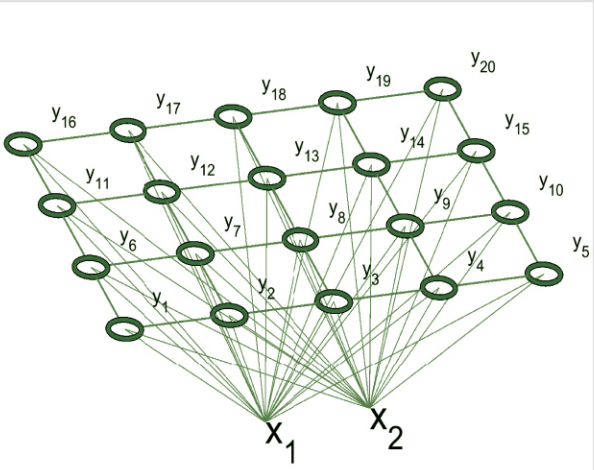


TERMÉSZETTUDOMÁNY

Brassai Sándor Tihamér

NEURÁLIS HÁLÓZATOK ÉS FUZZY LOGIKA

Sapientia Könyvek

Scientia Kiadó | 2019

BRASSAI SÁNDOR TIHAMÉR

NEURÁLIS HÁLÓZATOK ÉS FUZZY LOGIKA

SAPIENTIA KÖNYVEK

SAPIENTIA
ERDÉLYI MAGYAR
TUDOMÁNYEGYETEM

BRASSAI SÁNDOR TIHAMÉR

NEURÁLIS HÁLÓZATOK ÉS FUZZY LOGIKA

Scientia Kiadó
Kolozsvár·2019

SAPIENTIA KÖNYVEK

Természettudományok

Kiadja a
Scientia Kiadó
400112 Kolozsvár (Cluj-Napoca), Mátyás király (Matei Corvin) u. 4.
Tel./fax: +40-364-401454, e-mail: scientia@kpi.sapientia.ro
www.scientiakiado.ro

Felelős kiadó:
Kása Zoltán

Lektorálta:
Vörösházi Zsolt (Veszprém)

Első magyar nyelvű kiadás: 2019

Descrierea CIP a Bibliotecii Naţionale a României
BRASSAI, SÁNDOR-TIHAMÉR
Neurális hálózatok és Fuzzy logika / Brassai Sándor Tihamér. – Cluj-Napoca : Scientia, 2019
Conţine bibliografie
ISBN 978-606-975-021-6

004

TARTALOMJEGYZÉK

CONTENTS

CUPRINS

ELŐSZÓ

A könyv mérnökök, mérnökjelöltek és informatikusok számára nyújt bevezetést a mesterséges neurális hálózatok, a fuzzy logika világába, valamint az említett rendszerek újrakonfigurálható digitális áramkörön való megvalósításába. A könyv első részében széles körű áttekintést kapunk a neurális hálókról, a neurális hálok szerkezetéről és főként gradiens alapú tanítási módszerekről, valamint különböző típusú neurális hálók felépítéséről és tanításáról.

A második rész a fuzzy logikába és fuzzy következtető rendszerekbe vezeti be az olvasót. A harmadik rész az újrakonfigurálható áramkörök szerkezetét követően neuronhálók és fuzzy következtető rendszerek FPGA alapú hardveres megvalósítását tekinti át.

A neurális hálózatokról és fuzzy következtető rendszerekről számos szakkönyv érhető el akár magyar nyelven is. Prof. Roska Tamás és kutatótársai számos könyvet, cikket írtak a celluláris, neurális/nemlineáris hálózatok témakörben. Horváth Gábor *Neurális hálózatok és műszaki alkalmazásaik* és *Neurális hálózatok* című könyvei szintén szaknyelvi alapot biztosítanak az olvasók számára. Mindketten a neurális hálózatok méltán híres magyar kutatói voltak.

A könyvben nincs lehetőség a témakört teljes mértékben átfogni. A szerző a neurális hálókról és fuzzy következtető rendszerekről oktatás és kutatás terén szerzett tapasztalatait, a hardvermegvalósításokkal elért eredményeit foglalja össze. A könyv első és második része bevezeti a neurális hálózatok és fuzzy következtető rendszerek felépítésébe és különböző lehetséges gyakorlati alkalmazásába egyaránt a számítástechnika, automatizálás, mechatronika és informatika alapképzés szakos hallgatókat.

A könyv harmadik része újrakonfigurálható digitális áramkör ismeretekkel rendelkező mérnökjelölteket és akár mérnök kollégákat, kutatókat céloz meg, akik a neurális hálók hardveres megvalósítása terén tevékenykednek, valamint célja mesteri szakos hallgatók érdeklődését felkelteni. Közismert a neuronhálók, fuzzy következtető rendszerek, valamint egyéb

softcomputing módszerek, mint például a konvolúciós neuronhálók egyre szélesebb körben való, valós idejű gyakorlati alkalmazása.

A számítógépes irányítási rendszerek mesteri szakon, melyen a diákok már alapvető ismereteket szereztek az újrakonfigurálható digitális rendszerek terén, a könyvben tárgyalt neuronhálók és fuzzy következtető rendszerek megvalósításai alapján rálátást szereznek mind a neurális hálózatok, mind a fuzzy következtető rendszerek gyakorlati valós idejű megvalósítására. A szerzett tapasztalatok alapján bővíthetik ismereteiket a neuronhálók és egyéb algoritmusok újrakonfigurálható áramkörön való megvalósítása terén.

A neurális számítástechnika a természetes (biológiai) neurális rendszerek felépítésének és működésének mintájára épít számító rendszereket, melyek példákból nyert tapasztalatok felhasználásával tanulás útján oldják meg a feladatokat.

A mesterséges neuronok a biológiai neuron bizonyos részeit, dendriteket, sejttesteket és axonokat utánoznak egyszerű matematikai modellekkel.

Ha a sejttest a bejövő dendriteken keresztül elegendő ingert kap, az axonon egy jelet hoz létre. A kimenő jelek más neuronok bemeneteit képezik, ismételve ezt a folyamatot. A jelek (elektromos, kémiai) továbbítása a neuronok között a szinapszisokon keresztül történik.

A neuronok közötti kapcsolatok erősebbé vagy gyengébbé válhatnak, új kapcsolatok jelenhetnek meg, míg mások megszűnhetnek. Ezt a folyamatot utánozza a mesterséges neuron, amely egy olyan függvényt valósít meg, amely összegzi a súlyozott bemeneti jeleket. Ha az összegzett bemeneti jelek meghaladnak egy küszöbértéket, a neuron egy kimeneti jelet hoz létre.

Az egyszerűsített mesterséges neuronmodellek nem képesek utánozni sem a neuronok (dendritek vagy axonok) létrehozását, sem a megsemmisítését, és nem veszi figyelembe a jelek időzítését sem.

A biológiai neurális hálózatok sokkal bonyolultabbak, mint a mesterséges neurális hálózatokban használt matematikai modellek.

Habár egy egyszerű neuron feladatmegoldó képessége minimális, nagyon sok neuron együttes alkalmazásával komplex feladatok oldhatók meg.

Egyre szélesebb körben, különböző tudományágakban megoldandó feladatokra alkalmazzák a neurális hálókat: alakfelismerés, osztályozás, jelanalizálás, szűrés, adaptív irányítás.

E könyv keretében a mesterséges intelligenciához kapcsolódó neurális rendszerek, fuzzy következtető rendszerek alapvető elemeit, valamint hardver alapú megvalósítását mutatjuk be, törekedve arra, hogy az olvasó elsajátítsa az elméleti alapokat, és ugyanakkor tapasztalatot szerezzen az említett rendszerek gyakorlati alkalmazásában is. A fejezetek egymásra

épülnek, az elején bemutatva a neurális hálók alapvető elemeit, a mesterséges neuron felépítését, a PERCEPTRON és ADALINE típusú neuronok struktúráját és tanítását. Bemutatjuk a neuronhálók topológiáját és a neuronhálók gyakorlatban történő, szoftver és hardver alapú megvalósítását. Amint ismeretes, a többrétegű előrecsatolt perceptron (MLP) alapú neuronháló széles körben alkalmazható különböző típusú feladatok megoldására. Több rész keretében tárgyaljuk az MLP neuronhálók struktúráját, tanítását, megvalósítását, valamint a MATLAB programcsomag Neural Network Toolbox könyvtárában megtalálható MLP (és egyéb típusú) hálóval kapcsolatos függvények alkalmazását.

A neurális hálók alkalmazásának feltétele a háló tanításának a megfelelő módon való elvégzése. Tárgyaljuk a tanító és tesztelő halmazok előkészítését, a tanító algoritmusok megvalósítását, a hálók tanításának az elvégzését és a tanítás eredményének a kiértékelését. A neuronhálók alkalmazásának szemléltetésére például a radiális bázisfüggvényekből álló hálózat (RBF) és CMAC (Cerebellar Model Articulation Controller) hálókat használjuk függvények megközelítésére. Az említett neuronhálók az ismert és alkalmazott neuronhálóknak és tanítási algoritmusoknak csak egy töredékét képezik, de a könyv keretében csak az alapvető neurális hálókkal kapcsolatos ismeretek elsajátítására van lehetőség.

A könyv második részében a fuzzy következtető rendszereket mutatjuk be. A fuzzy halmazelmélet és fuzzy következtető rendszerek bemutatása során a lehető legegyszerűbb formában tárgyaljuk ezeket, a fő cél a gyakorlati alkalmazhatóság szemléltetése. A fuzzy rendszereknél bemutatjuk a fuzzy halmazelmélet alapjait, a fuzzy halmazműveleteket, az alkalmazott különböző típusú tagsági függvényeket, a MAMDANI és Takagi Sugeno típusú következtető rendszer felépítését, a szabálytáblázat kitöltésének módszerét, a fontosabb defuzzyfikálási eljárásokat. Szintén foglalkozunk a fuzzy halmazokon értelmezett relációkkal, különböző szerzőktől származó normáknak a bemutatásával.

A harmadik részben neurális hálózatok és fuzzy következtető rendszerek hardver alapú megvalósítását tárgyaljuk. Részletezzük a neurális hálózatok hardveres megvalósítására jellemző megszorításokat: mint például az alkalmazott aritmetika, pontosság, párhuzamosítási lehetőségek. Az utolsó fejezetekben egy RBF neuronháló és egy Takagi-Sugeno fuzzy következtető rendszer hardveres megvalósítását szemléltetjük. A hardveres megvalósításhoz kapcsolódó eredményeket az elvégzett kutatások során értem el. A

Takagi-Sugeno modellre épül az ANFIS Adaptiv Neuro fuzzy következtető rendszer. A tárgyalt modell alkalmazható ANFIS modellként is, mivel implementálva van egy valós időben működő tanító algoritmus.

Marosvásárhely, 2017. február 18. Brassai Sándor Tihamér

Rövidítésjegyzék

Rövidítés	Leírás
ADALINE	Adaptív lineáris elem
AMN	Asszociatív memória típusú neuronháló
ART	Adaptív rezonanciaelmélet
ASIC	Alkalmazásspecifikus integrált áramkör
BlockRAM, BRAM	FPGA áramkörben található memóriatömb. Az FPGA áramkörök alapvető alkotóelemei
CIFAR-100	Canadian Institute for Advanced Research tanítóhalmaza, 100 osztályba tartozó képeket tartalmaz, minden osztályban 600 képpel
CLB	Konfigurálható logikai tömb, FPGA áramkör alap alkotóeleme
CMAC	Cerebellar Model Articulation Controller típusú neuronháló
COA	Területközéppont-módszer defuzzyfikáló eljárás
COG	Súlypontmódszer defuzzyfikáló eljárás
COM	Maximumok közepe defuzzyfikáló eljárás
CORDIC	Koordináta geometriai módszer, amely aritmetikai és bit szintű műveletekre épül
CPLD	Komplex programozható logikai eszköz
CPS	Másodpercenkénti kapcsolatok, a neuronhálók feldolgozási sebességét jellemző paraméter
CRI	Rekurzív interpolációs módszer
CUPS	Másodpercenként frissített súlytényezők száma, a neurális hálózat tanulási sebességét jellemzi
DCM	Órajel-menedzselő modulok
DSP	Digitális jelfeldolgozó modul, amely főleg szorzás, de egyéb aritmetikai műveletek elvégzésére szolgál. A modern FPGA áramkörök alkotóelemei
EEPROM	Elektromosan törölhető és programozható ROM memória
ELU	Exponenciális lineáris egység, aktivációs függvény
EPROM	Törölhető és programozható ROM
FIR	Véges impulzus válasz típusú szűrő
FPGA	Helyszínen programozható, logikai kapukat tartalmazó tömb

Rövidítés	Leírás
FUSE	Félvezetőkben alkalmazott olvadóbiztosíték alapú technológia
GPU	Grafikai processzor
HDL	Integrált áramkörök működésének leírására szolgáló leíró nyelv
IOB	Ki-bemeneti tömb, az FPGA áramkörökbe a jelek be-, illetve kivezetése az IOB modulokon keresztül történik
IP	Szellemi tulajdonjog
Leaky ReLU	Szivárgó rektifikált lineáris elem, aktivációs függvény
LMS	Legkisebb négyzetek módszere
LUT	Kereső táblázat, az FPGA áramkörben logikai függvények megvalósítására szolgáló elem
LVQ	Tanulóvektor kvantálás
MIMD	Többszálú utasításáramlás, többszálú adatáramlás típusú rendszer
MLP	Előrecsatolt többrétegű perceptron típusú neuronháló
MOM	Maximumok átlaga defuzzyfikáló eljárás
MSE	Átlagos négyzetes eltérés költségfüggvény
MUL	FPGA áramkörökben megtalálható szorzómodul, a továbbfejlesztett változata a DSP modulok
NAM	Neurális asszociatív memória
NN	Neurális hálózatok
PCA	Főkomponens-analízis módszer
PCI	Perifériaegységek összeköttetésére szolgáló sínrendszer
PNN	Valószínűségi neuronháló
PReLU	Parametrizált rektifikált lineáris egység, aktivációs függvény
PROM	Programozható olvasható memória
PSO	Részecske-raj optimalizálási algoritmus
PWL	Szakaszonkénti lineáris megközelítés
PWM	Impulzus-szélesség moduláció
RAM	Véletlen hozzáférésű írható és olvasható memória
RBF	Radiális bázisfüggvényekből álló hálózat
ReLU	Rektifikált lineáris elem, aktivációs függvény
ROM	Csak olvasható, adatok tárolására alkalmas memória
SIMD	Egyszálú utasításáramlás, többszálú adatáramlás típusú rendszer
SNN	Spiking típusú neurális hálózatok
SoC	Rendszercsip, olyan integrált áramkör, amelyben egy teljes funkció minden kelléke megtalálható
SOM	Kohonen önszerveződő térkép
SRAM	Statikus RAM memória
SVM	Tartó vektor gép
TSP	Travelling Salesman probléma
VHDL	Nagyon nagy sebességű integrált áramkörök leírására szolgáló hardverleíró nyelv, rövidítése a VHSIC – Very High Speed Integrated Circuits Hardware Description Language kifejezésből származik

I. rész

Neurális hálózatok

1. fejezet

Mesterséges neuron

Ebben a részben a mesterséges neuron struktúrájának az ismertetése, neuronhálókkal kapcsolatos elemek, alapfogalmak, aktivációs függvénytípusok szemléltetése és a neuron matematikai modellje kerül bemutatásra.

1.1. A neuronok felépítése

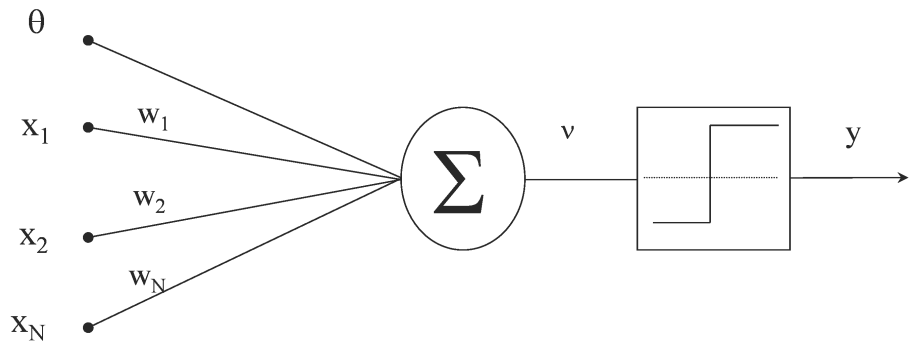
Egy neurális hálózat nagy számú, egyszerű felépítésű processzáló elemek, neuronok rendezett, reguláris felépítését tartalmazza. A neurális hálózat azonos vagy hasonló felépítésű, lokális feldolgozást végző műveleti elemekből épül fel. A feldolgozó elemeket neuronoknak nevezzük.

A neuronhálóban a feldolgozó elemek általában rendezett topológiájúak és jól meghatározott módon vannak összekapcsolva. A feladat végrehajtása során fontos szerepet játszik a neuronháló szerkezete. A neuronhálók belső párhuzamos felépítéséből adódóan a számítások párhuzamosan hajthatók végre, ezáltal nagy feldolgozási sebességet biztosítva. Így a neuronhálók kimondottan alkalmasak valós időben működő feladatok megoldására.

A neuronháló párhuzamos feldolgozást tesz lehetővé és lehetséges szoftver vagy hardver alapú megvalósításra. Speciális hardvereszközök alkalmazhatók a neuronhálók megvalósítására, amelyek kihasználják a neuronhálók párhuzamos szerkezeti felépítését. Számos neuronháló-típusnak megfelelően neuroprocesszor is rendelkezésre áll, amelyek többé-kevésbé kiaknázzák egy adott neuronhálóban rejlő párhuzamosságot. A valós idejű működés mellett a párhuzamos szerkezeti felépítésből adódóan a neuronhálók redundáns magas fokú hibatűrő rendszernek bizonyulnak [2].

A neurális hálózat azonos vagy hasonló felépítésű, lokális feldolgozást végző műveleti elemekből épül fel. A feldolgozó elemeket neuronoknak nevezzük. A neuronhálóban a feldolgozó elemek általában rendezett topológiájúak és jól meghatározott módon vannak összekapcsolva. A neuronháló párhuzamos feldolgozást tesz lehetővé és lehetséges szoftver vagy hardver alapú megvalósítása [2].

A neuron több bemenetű, egy kimenetű feldolgozó (műveletvégző) elem, amely rendelkezhet lokális memóriával, amelyben akár bemeneti, akár kimeneti értékeket tárol [3]. A bemeneti és tárolt értékekből az aktuális kimeneti értéket tipikusan nemlineáris (vagy lineáris) transzferfüggvény segítségével hozza létre, amelyet aktiváló függvénynek nevezünk.



1.1. ábra. Egy általános neuron szerkezete

A neuron bevezetése során a következő jelöléseket alkalmazzuk:
$x_1, x_2..x_i..x_N$ – a neuron bemenetei
$X = [x_1, x_2..x_i..x_N]$ – bemeneti vektor
N – neuron bemeneteinek a száma
θ – konstans bemenet (bias – eltolási érték)
ν – inger (súlyozott összeg)
y – a neuron kimenete
w_i – az i-edik bemenethez kapcsolódó súlytényező
$W = [w_1 w_2..w_i..w_N]$ – súlyvektor
φ – aktivációs függvény

A neuron bemenetei között (1.1. ábra) megkülönböztetünk konstans és változó értékű bemeneteket. A konstans bemenetnek a tanulás során van

szerepe. Az x_i bemenetek változó értékűek, míg a θ konstans marad, miután az értékét meghatároztuk.

Az x_i skalár-bemenetek w_i súlyozással kerülnek összegzésre, majd a súlyozott összeg egy nemlineáris elemre kerül. A bemeneti jelek súlyozott összegét, mely az aktivációs függvény bemenete, ingernek (excitation), míg a kimeneti jelet válasznak (activation) nevezzük. A φ függvényt aktiváló függvénynek (activation function) nevezzük. A neuron kimenete a következőképpen számolható ki:

$$y = \varphi\left(\sum_{i=1}^{N} x_i w_i - \theta\right).$$

A súlytényezők a neuronok lokális környezetében lévő más neuronokkal való kapcsolatok erősségét határozzák meg. A neuronhálók működésének szabályozása a súlytényezőkkel történik, az információ vagy az információfeldolgozó eljárás rögzítése a súlytényezőkben (hangolható paraméterekben) valósul meg a tanítási folyamat során. A neuronháló rendelkezik tanulási algoritmussal (learning algorithm), amely általában minta alapján való tanulást jelent.

A neurális hálózatok működése tipikusan két fázisra választható szét:

1. Tanulási fázis – a hálózatban valamilyen módon eltároljuk a kívánt információfeldolgozó eljárást.

2. Előhívási fázis (recall) – a tárolt eljárás felhasználásával elvégezzük az információfeldolgozást.

A két fázis a legtöbb esetben időben szétválik, rendszerint a tanulási fázis lassú, több iterációt, esetleg sikertelen tanulási szakaszokat is hordoz. Általában a tanítás korszakokba (epoch) van szervezve, és egy-egy korszak lefuttatása után lehetőség van a tanítási paraméterek újrahangolására. Az előhívási fázis gyors feldolgozást jelent [4] és rendszerint az alkalmazás pillanatában történik. Az előhívási fázis során a pillanatnyi bemeneti értékek alapján meghatározzuk a neuronháló kimenetét.

Adaptív rendszerek esetében a két fázis nem válik szét, az információ-előhívással párhuzamosan valósul meg a tanulás, az előhívási szakaszban történik a paramétereik módosítása is.

1.2. Aktivációs függvények

A neuronok működésében fontos szerepet töltenek be az aktivációs függvények. A neuronhálókban elsősorban a nemlineáris és folytonosan differenciálható aktivációs függvényeknek tulajdonítható fontosabb szerep. Megfelelő neuron mellett a nemlineáris aktivációs függvény alaklmazása lehetővé teszi bármilyen nemlineáris függvény neuronhálóval való modellezését. Lineáris aktivációs függvény alkalmazása eredményeként a neuronháló is nemlineáris. Ahhoz, hogy a neuronhálót nemlineárissá tegyük, legalább egy nemlineáris aktivációs függvényt kell alkalmazni. A differenciálhatóság is fontos, mert a gradiens alapú tanítás a leggyakrabban alkalmazott módszer a neuronhálók súlyainak a hangolására.

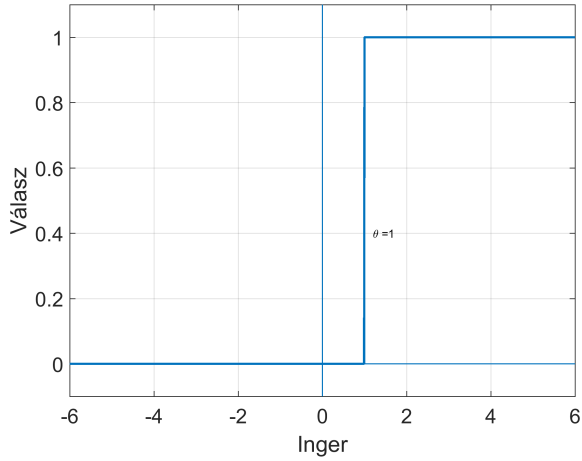
A mesterséges neuron esetében a következő aktivációs függvények alkalmazhatóak [2], [3], [4], [1]:

- Küszöbfüggvény vagy egységugrás-függvény (1.2. ábra)
- Lépcsőfüggvény (1.3. ábra)
- Lineáris aktivációs függvény (1.4. ábra)
- Logisztikus vagy szigmoid alakú karakterisztika
- Tangens hiperbolikus függvény (1.7., 1.8. ábra)
- Telítéses lineáris függvény (1.9. ábra)
- Gauss-függvény
- ReLU aktivációs függvény (1.11. ábra)
- Leaky ReLU (szivárgó ReLU)
- PReLU aktivációs függvény
- ELU aktivációs függvény (1.13. ábra)
- Softmax aktivációs függvény

1.2.1. Küszöbfüggvény vagy egységugrás-függvény

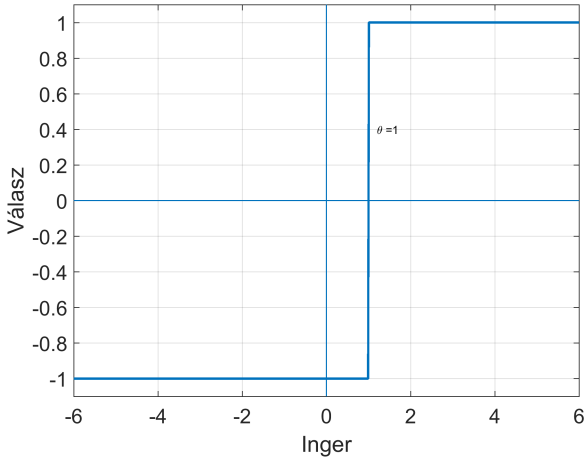
Egységugrás aktivációs függvényre, ha az inger értéke meghaladja a θ küszöböt, a neuron kimenete egyes, különben zérus értéket vesz fel (1.1).

$$\varphi(x) = \begin{cases} 0 & \text{ha} \quad x < \theta \\ 1 & \text{ha} \quad x \geq \theta \end{cases} \tag{1.1}$$



1.2. ábra. Küszöbfüggvény

1.2.2. Lépcsőfüggvény



1.3. ábra. Lépcsőfüggvény

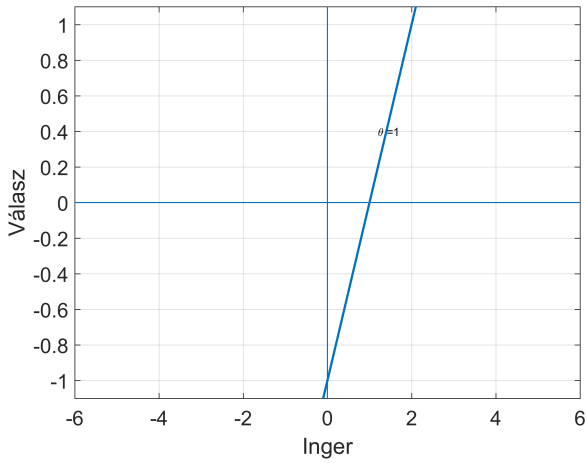
Lépcsőfüggvény aktivációs függvényre, ha az inger értéke meghaladja a θ küszöböt, a neuron kimenete egyes, különben mínusz egy (1.2).

$$\varphi(x) = \begin{cases} -1 & \text{ha} \quad x < \theta \\ 1 & \text{ha} \quad x \geq \theta \end{cases} \tag{1.2}$$

1.2.3. Lineáris aktivációs függvény

A neuron válasza, lineáris aktivációs függvény alkalmazása esetében, lineárisan növekszik az inger értékével. Az aktivációs függvényt az (1.3) egyenlet írja le.

$$\varphi(x) = x \tag{1.3}$$

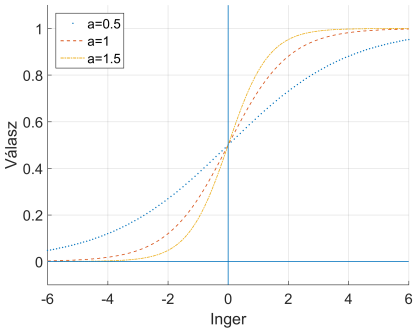


1.4. ábra. Lineárisan növekvő aktivációs függvény

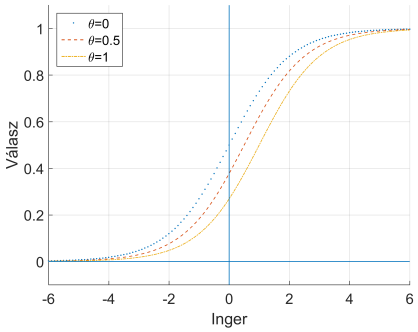
1.2.4. Logisztikus vagy szigmoid alakú karakterisztika

A szigmoid alakú aktivációs függvény értéke az egyenlet (1.4) alapján számolható ki. Az a paraméterrel lehet változtatni a függvény szaturációs részét. A függvény különböző a paraméter értékekre az 1.5. ábrán, valamint θ értékekre az 1.6. ábrán van szemléltetve.

$$\varphi(x) = \frac{1}{1 + e^{-ax-\theta}} \tag{1.4}$$



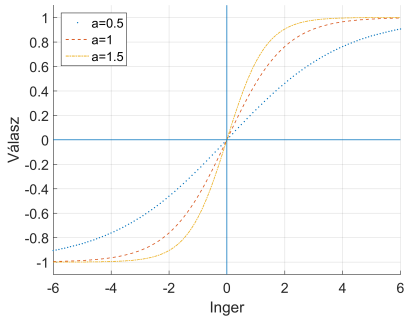
1.5. ábra. Szigmoid aktivációs függvény különböző a értékekre



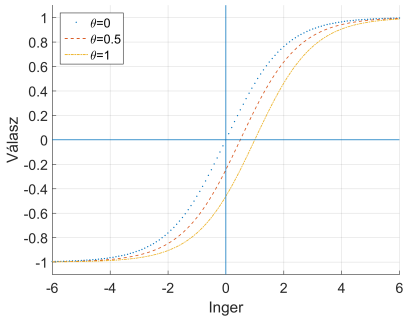
1.6. ábra. Szigmoid aktivációs függvény különböző θ értékekre

A szigmoid a biológiából inspirált aktivációs függvény, a teljes tartományban sima és differenciálható. Korábban a többrétegű perceptron típusú neuronhálókban nagyrészt szigmoid aktivációs függvényt alkalmaztak. A szigmoid függvény szaturálódik az inger értékeinek növelésével, vagyis a deriváltja nagyon kicsi lesz, aminek a hatására a súlytényezők hangolása elakad. Azt is szokás mondani, hogy az adott neuron elhal.

1.2.5. Tangens hiperbolikus függvény



1.7. ábra. Tangens hiperbolikus aktivációs függvény különböző a értékekre



1.8. ábra. Tangens hiperbolikus aktivációs függvény különböző θ értékekre

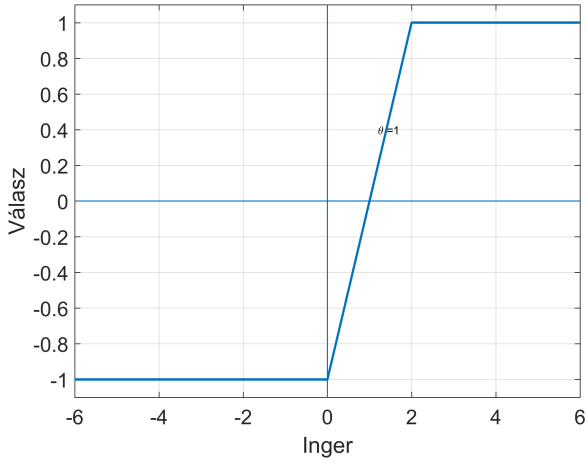
A tangens hiperbolikus aktivációs függvény értéke az (1.5) egyenlet alapján számolható ki. Az 1.7. ábrán a tangens hiperbolikus függvény különböző a, valamint θ értékekre van ábrázolva.

$$\varphi(x) = \frac{1 - e^{-ax-\theta}}{1 + e^{-ax-\theta}} \tag{1.5}$$

Előnye a szigmoid aktivációs függvényhez képest, hogy az origóhoz közeli értékekre a függvény értéke is zérus. Ez egy fontos szempont, mivel gyorsítja a gradiens módszer konvergenciáját. Azonban hasonlóan a szigmoid függvényhez, a tangens hiperbolikus is szaturálódik az inger abszolút értékeinek növelésével.

1.2.6. Telítéses lineáris függvény

A lineáris függvényre az aktiválás arányos a bemenettel. Ha egy többrétegű neuronháló mindenik rétege lineáris aktivációs függvényre épül, nem számít, hány rétege van a neuronhálónak, mert csak lineáris leképzést képes megvalósítani. A neuronháló egyetlen nemlineáris aktivációs függvényekre épülő réteggel ekvivalens. A telítéses lineáris függvény (1.9. ábra) a lineáris



1.9. ábra. Telítéses lineáris aktivációs függvény

függvényre épül, szaturálva egy-egy adott érték felett (és alatt) a kimenetet (1.6).

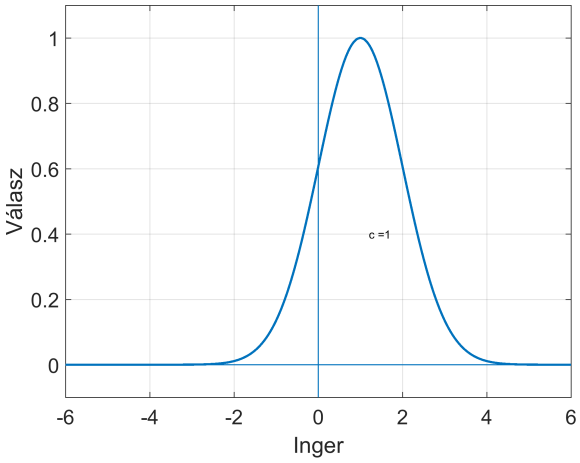
$$\varphi(x) = \begin{cases} -1 & \text{ha} \quad x < -1 \\ x & \text{ha} \quad -1 \leq x < 1 \\ 1 & \text{ha} \quad x \geq 1 \end{cases} \tag{1.6}$$

1.2.7. Gauss-függvény

A Gauss-függvényt (1.10. ábra) például az RBF típusú hálók és Kernel-hálók esetében alkalmazzák. Az RBF típusú hálót a 6. fejezet mutatja be részletesen. A Gauss-függvény az (1.7) egyenlet szerint számolható.

$$\varphi(\underline{x}, \underline{c}, \sigma) = e^{\frac{-\|\underline{x}-\underline{c}\|^2}{2\sigma^2}} \tag{1.7}$$

ahol
σ – szórás
c – középpont



1.10. ábra. Gauss aktivációs függvény c=1, $\sigma = 1$

$$\|\underline{x} - \underline{c}\| = \sqrt{(x_1 - c_1)^2 + (x_2 - c_2)^2 + .. + (x_N - c_N)^2}$$

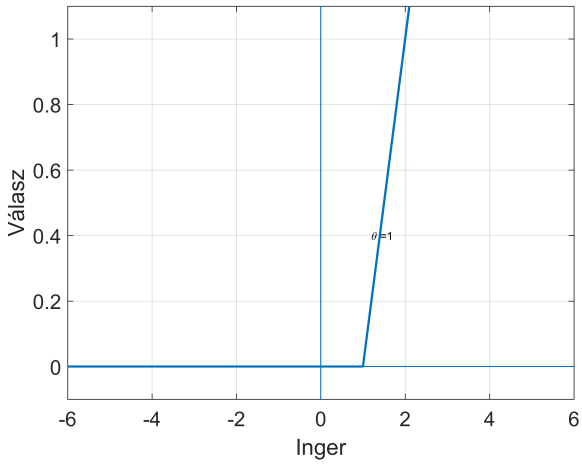
ahol
$\underline{x}$ – bemeneti vektor
$\underline{c}$ – középpont vektor
σ – szórás

1.2.8. ReLU rektifikált lineáris

Az utóbbi években az egyik legnépszerűbb aktivációs függvény [1]. A bevezetését követően helyettesíti a leggyakrabban alkalmazott szigmoid aktivációs függvényt, a gépi tanulási feladatokból származó pozitív hatás következményeként. A közelmúltban bebizonyosodott, hogy hatszor gyorsabb konvergenciát biztosít, összehasonlítva a tangens hiperbolikus aktivációs függvénnyel.
A ReLU-t és deriváltját az (1.8) és (1.9) egyenletek írják le.

$$\varphi(x) = max(0, x) \quad vagy \quad \varphi(x) = \begin{cases} 0 & \text{ha} \quad x < 0 \\ x & \text{ha} \quad x \geq 0 \end{cases} \tag{1.8}$$

A sok réteget tartalmazó neuronhálókat mély neuronhálóknak is szokás nevezni. A szigmoid és tangens hiperbolikus aktivációs függvények alkalmazása a kevés réteget tartalmazó neuronhálókra szűkül. Az említett függvények gradiense eltűnik a mély neuronhálók esetében, megakadályozva a neuronháló tanulását. A ReLU aktivációs függvény deriváltja mindig egy és nincs szaturálva a kimenet. Jó megoldást biztosít a mély neuronhálók esetében. Negatív ingerértékekre zérus kimeneti neuronokat eredményez. A neuronok, amelyek minden esetre zérus kimenetet eredményeznek, egyszerűen kivághatók a neuronhálóból, csökkentve a számításigényt.



1.11. ábra. ReLU aktivációs függvény

ReLU aktivációs függvény deriváltja:

$$\varphi'(x) = \begin{cases} 0 & \text{ha} \quad x < 0 \\ 1 & \text{ha} \quad x \geq 0 \end{cases} \tag{1.9}$$

Egyes gradiens alapú tanulási módszerek során a ReLU aktivációs függvény is a neuron elhalásához, elvesztéséhez vezethet. Egy olyan súlytényező-adaptálást eredményezhet, amelyet követően a neuron többet egyetlen adatpontra sem aktiválódik. A neuronháló szempontjából az említett neuronok elvesznek, sőt pazarolják a számítási erőforrásokat.

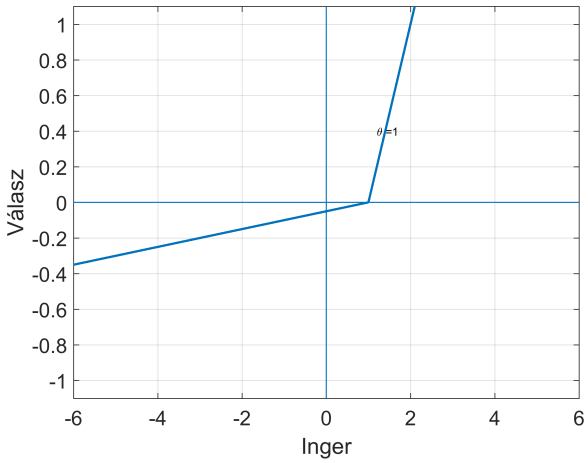
1.2.9. Leaky ReLU (szivárgó ReLU)

A szivárgó ReLU (1.10) aktivációs függvényt (1.12. ábra) [1] a ReLU hátrányának kiküszöbölésére vezették be. A függvénynek a negatív részekre is egy lejtést biztosít, lehetővé téve ezen a szakaszon is az adaptív súlytényező-beállítást.

$$\varphi(x) = \begin{cases} \alpha x & \text{ha} \quad x < 0 \\ x & \text{ha} \quad x \geq 0 \end{cases} \tag{1.10}$$

LReLU deriváltja (1.11)

$$\varphi'(x) = \begin{cases} \alpha & \text{ha} \quad x < 0 \\ 1 & \text{ha} \quad x \geq 0 \end{cases} \tag{1.11}$$



1.12. ábra. Leaky ReLU aktivációs függvény, $\alpha = 0{,}05$

1.2.10. PReLU aktivációs függvény

A lényeges különbség a LReLu és a PReLu (1.12) között, hogy a parametrizált rektifikált lineáris egység esetében az α_i a neuronháló egy tanítható paramétere [1]. Az α_i paraméter hangolása a neuronháló súlytényezőihez hasonlóan valósítható meg [1].

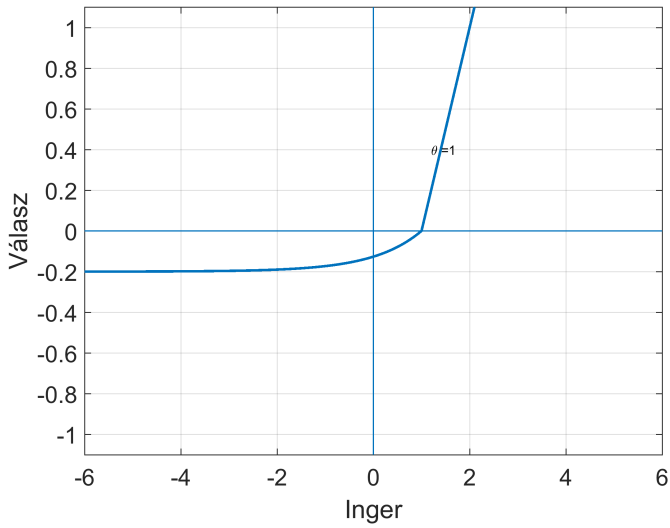
$$\varphi(x) = \begin{cases} \alpha_i x & \text{ha} \quad x < 0 \\ x & \text{ha} \quad x \geq 0 \end{cases} \tag{1.12}$$

PReLU deriváltja (1.13)

$$\varphi'(x) = \begin{cases} \alpha_i & \text{ha} \quad x < 0 \\ 1 & \text{ha} \quad x \geq 0 \end{cases} \tag{1.13}$$

1.2.11. ELU aktivációs függvény

Az exponenciális lineáris egység (ELU) egy újabb, a ReLU-ra épülő aktivációs függvény (1.13. ábra). Felgyorsítja a tanulást és enyhíti az eltűnő gradiens problémát [1]. Egy egyszerű konvolúciós neuronhálót alkalmazva, a CIFAR-100 adatbázist használva, összehasonlították a ReLu és ELU aktivációs függvényeket. Az eredmények alapján a neuronháló jobban teljesített az ELU aktivációs függvényt alkalmazva a ReLU-hoz képest [5]. Az ELU-t



1.13. ábra. ELU aktivációs függvény, $\alpha = 0{,}2$

az (1.14) egyenlet írja le.

$$\varphi(x) = \begin{cases} \alpha\,(e^x - 1) & \text{ha} \quad x < 0 \\ x & \text{ha} \quad x \geq 0 \end{cases} \tag{1.14}$$

1.2.12. Softmax aktivációs függvény

A softmax aktivációs függvény (1.15) a szigmoid függvényhez hasonlóan minden kimenetet a 0 -1 tartományba vetít (normalizálja). Viszont a kimeneteket elosztja a rendszer kimeneteinek összegével, úgy, hogy a normalizált kimenetek összege 1. A Softmax normalizált exponenciális függvényként is értelmezhető. Osztályozási feladatokban a neuronháló kimenetén általában softmax aktivációs függvényt alkalmaznak. A softmax kimeneti aktivációs függvény több osztály diszkrét valószínűségi eloszlását határozza meg. Nagyon fontos szerepe van a mai sokrétegű konvolúciós neuronhálózatokban, mint például a VGGNet, AlexNet, GoogleNet.

$$\varphi_i = \frac{e^{y_i}}{\sum\limits_{j=1}^{N} e^{y_j}} \tag{1.15}$$

y_i – csomópontok kimenete,
φ_i – normalizált kimenetek.

Az aktivációs függvény típusának a megválasztása attól függ, hogy a neuron kimenete milyen értékeket vehet fel. Ha a kimenet egy bináris érték, aktivációs függvénynek küszöbfüggvényt vagy lépcsőfüggvényt választunk, 0, 1 értékek esetében küszöbfüggvényt, -1, 1 értékek esetében pedig lépcsőfüggvényt. Ha a kimenet folytonos érték, akkor egy folytonos aktivációs függvényt alkalmazunk, annak függvényében, hogy a kimenet csak pozitív, vagy pozitív/negatív értékeket is felvehet. Csak pozitív értékek esetében szigmoid függvényt, míg pozitív-negatív értékek esetében tangens hiperbolikus függvényt alkalmazunk.

A szigmoid, tangens hiperbolikus aktiváló függvényeket a kevés réteget tartalmazó neuronhálókban alkalmaznak. Csak lineáris aktivációs függvény nem alkalmazható egy neuronhálóban, mert csak lineáris kimenetet eredményez. Az egyenirányító karakterisztika alapú aktivációs függvényeket előnyösebb alkalmazni a mély neuronhálókban, mivel nem eredményezik a gradinesből származó neuronelhalást.

2. fejezet

Perceptron típusú neuron

A fejezet célja a tanító algoritmusok osztályozása, a tanító és tesztelő halmaz szerepének meghatározása a neuronhálók tanításában, a Perceptron és Adaline feldolgozó elemek struktúrája, tanítása és a köztük lévő hasonlóságoknak és különbségeknek az ismertetése.

2.1. Tanítási módszerek

A neurális hálózatok legfőbb jellemzője az adaptációs tanulási képesség. A neurális hálózatokban a tanulás egyszerűen a rendszer valamilyen képességének javítását jelenti. Így tanulásról beszélhetünk, amikor olyan hálózatarchitektúrát, illetve súlytényezőket keresünk, amelyek mellett egy hálózat egy adott függvénynek a minél jobb approximációjára lesz képes, de tanulás során egy hálózat azon képessége is fejleszthető, amely a bemenetére kerülő minták közötti hasonlóság megállapítását teszi lehetővé [4]. A neurális hálózatok főbb tanulási formái [6]:

1. Tanítóval történő tanulás (ellenőrzött, felügyelt vagy irányított tanulásnak is nevezik)
2. Megerősítéses tanulás
3. Tanulás tanító nélkül (nem ellenőrzött vagy felügyelet nélküli tanulás)
4. Analitikus tanulás

2.1.1. Ellenőrzött tanulás

Ellenőrzött tanulásnál a hálózat paramétereinek hangolására összetartozó be- és kimeneti értékek, úgynevezett tanítópárok állnak rendelkezésre. A tanítás azon alapszik, hogy ismertek a hálózatnak valamely bemenetekre adandó kívánt válaszai, így a hálózat tényleges válasza minden esetben közvetlenül összehasonlítható a kívánt válasszal (2.1. ábra). Az összehasonlítás eredménye – az elvárt kimenet és a számított válasz különbsége (hiba) – felhasználható a hálózat paramétereinek a módosítására. A paraméterek hangolásának az a célja, hogy a számított válaszok a kívánt válaszokkal minél jobban megegyezzenek, a hálózat kimenete és a kívánt kimenet közötti különbség csökkenjen. Ellenőrzött tanulásról beszélünk akkor is, amikor a kívánt válasz pontosan nem ismert, csupán annyit tudunk, hogy a neurális hálózat válasza helyes vagy nem. Ezt megerősítéses (reinforcement) tanulásnak nevezzük.

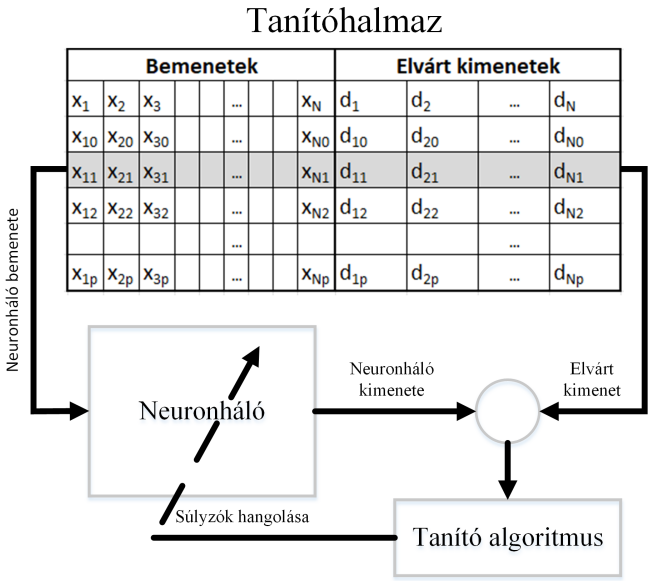
A neurális hálózatok tanításakor meg kell említeni a tanítási folyamat fontosabb lépéseit. Fontos kiemelni azt, hogy az információ a neurális háló súlytényezőiben van eltárolva. A tanítási folyamat ezen súlytényezők meghatározását, beállítását, módosítását jelenti.

A tanítás egy ciklikusan ismétlődő folyamat, amely során a tanítóhalmazból egyenként vesszük a bemeneteket, megmutatjuk a hálónak, az adott bemenetre kiszámoljuk a háló kimenetét, felhasználva az aktuális súlytényezőket, hangolható paramétereket, majd az elvárt érték és a számított érték között számolunk egy hibát. A kapott hibát majd alkalmazzuk a súlytényezők módosítására.

A tanítási algoritmust két különböző osztályba sorolhatjuk, attól függően, hogy milyen módon határozzuk meg a hibát: minden bemenetre egyenként számolunk egy hibát vagy csak a teljes tanítóhalmazra számolunk egy hibavektort.

Első esetben a tanítás a hiba pillanatnyi értéke alapján történik. A második esetben a tanítóhalmazból nem egy, hanem egy blokknyi mintára számított globális hiba alapján valósul meg a súlytényezők újraszámolása. Ezt a tanítási módot szokás batch tanításnak nevezni. Grafikai processzorokkal (GPU) való mély neuronhálózat tanítása során ezzel a paraméterrel lehet szabályozni a tanulási képességet, nyilván többletmemória-igénnyel. A pillanatnyi hiba alapján történő tanításhoz kevesebb erőforrásra, memóriára van szükség.

Több tanítási algoritmust lehet alkalmazni egy sokváltozós optimalizálási feladatra, mint például a neuronhálók paramétereinek a hangolása.



2.1. ábra. Tanítóhalmaz elemeinek a neuronháló bemeneteire való kapcsolása

A legegyszerűbb megoldás a gradiensalapú módszerek alkalmazása. A neuronháló tanítására alkalmazhatóak genetikus módszerek, rajelméletre épülő optimalizáló algoritmusok.

Tanító halmaz – be- és kimeneti elempárok, amelyek rendelkezésünkre állnak. Általában ezt egy táblázatban tároljuk.

Tanítási ciklus – a tanítás során a tanító halmazból veszünk egy be- és kimeneti elempárt. Az adott bemenetre kiszámoljuk a neuron (neurális háló) kimenetét. Az elvárt érték és számított értékből számolt hibát felhasználjuk a súlytényezők (paraméterek) módosítására. A tanítás során vesszük egyenként a tanító halmazból az elempárokat, és minden elempárra elvégezzük a paraméter-módosítást. Egy számítási ciklust – a háló tanítását az összes elempárra – tanítási ciklusnak nevezünk.

Egy neurális háló tanítása a tanítási ciklusok sorozatából tevődik össze. A tanítás befejeződését annak alapján dönthetjük el, hogy közben leellenőrizzük, hogy a háló teljesíti-e az elvárt feltételeket. A felügyelt tanítás esetében például minden tanítási ciklusra számolunk egy globális hibát, egy előre meghatározott kritériumfüggvény (költségfüggvény) alapján, melyet

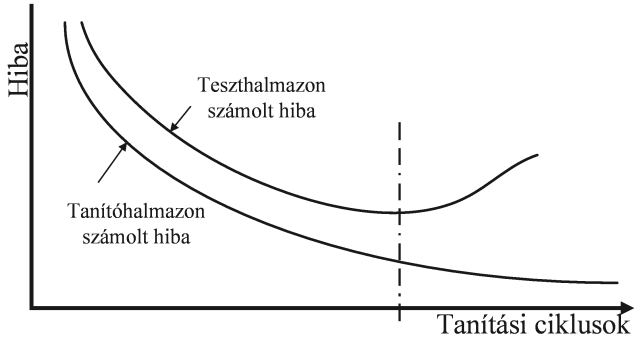
ábrázolunk, és ha a hiba egy bizonyos küszöbérték alá kerül, leállítjuk a tanítást.

A tanítás lényegében két fázisból tevődik össze:

- első fázis a tanítási fázis, amikor tanítjuk a hálót, és erre rendelkezésre áll egy tanító halmaz, amely ki-bemeneti elempárokat tartalmaz;
- tesztelő fázis, amikor leellenőrizzük a háló viselkedését olyan bemenetekre is, amelyekre nem tanítottuk a neuronhálót, nem végeztünk súlytényező-módosítást.

Előfordulhat, hogy a tanítás sikerrel jár, de a tesztelés sikertelen. Ebben az esetben akár újra kell gondolni a neuronháló szerkezetét, módosítani egyes paramétereket, újra kell tanítani a hálót. Mindkét esetben százalékban szokták kifejezni, hogy a tanító halmaz hány százalékára viselkedik helyesen a háló. A sikeres tanítás és tesztelés után elmentjük a súlytényezőket és alkalmazhatjuk a hálót. Ahhoz, hogy egy adott pontossággal meg tudjuk határozni, hogy hány százalékban ad helyes eredményt a háló, a tanító halmaznak megfelelő számú elemet kell tartalmaznia. Ha 10 darab elemre tanítjuk a hálót, csak 10%-os pontossággal tudjuk meghatározni a helyes működést. Ha 10 elemből egy elemre téved, azt jelenti, hogy 90%-ban helyesen működik a háló.

Ha minden tanítási ciklus végén mind a tanító halmazra, mind a teszthalmazra kiszámoljuk a hibát és egy grafikonon ábrázoljuk, következtetni lehet a neuronháló túltanítására.



2.2. ábra. Túltanítás

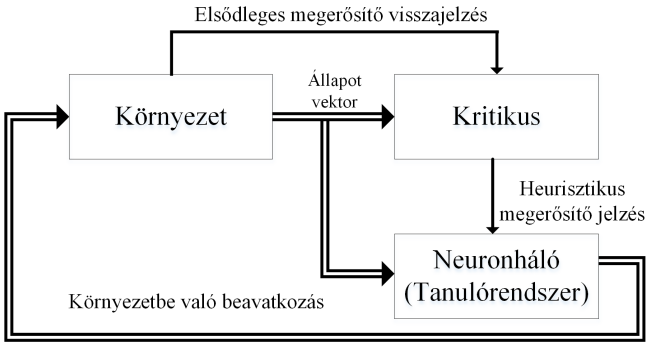
Akkor beszélünk túltanításról (2.2. ábra), amikor a tanító halmazon a hiba csökken, viszont a teszthalmazon a hiba elkezd növekedni. A tanítás első fázisában, hasonló mintákra, a tanító és teszthalmazban mindkét hiba

elkezd csökkenni. A tanítás során a neuronháló a teszthalmazban található mintákra egyre jobban kezd illeszkedni, viszont egy adott ponttól a teszthalmazban található mintáktól elkezd távolodni (a teszthalmazon számított hiba elkezd növekedni). Ettől a pillanattól mondjuk azt, hogy túltanítottuk a neuronhálót. Ha megtörténik a túltanítás, abban az esetben le kell állítani a tanítást és módosítani a neuronháló paramétereit vagy a tanító algoritmus beállításait, és újra kell kezdeni a tanítást.

2.1.2. Megerősítéses tanulás

A megerősítéses tanulás során a hálózat logikai {0,1} vagy egy valós értéket kap egy tanítási ciklust követően, amely meghatározza, hogy az eredmény helyes-e vagy hibás.

A tanítókészlet bemeneti mintázatokból áll, a tanítási ciklus befejezése után egy megerősítéses érték jelzi, hogy az eredmény helyes volt-e vagy nem, egyes esetekben visszajelzés van arról is, hogy mennyire rossz vagy jó a neuronháló döntése. Ez alapján történik a súlytényezők tanítása (2.3. ábra). Egy lehetséges tanítás, hogy a helyes válaszokra kapott bemeneteket és a neuronháló kimenetén kapott eredményt hozzáadjuk a tanító halmazhoz. A rossz válaszok nem kerülnek be a tanítóhalmazba. A megerősítéses tanulás részletes ismertetése és a gyakorlati feladatok szemléltetése a [7] szakirodalomban van részletezve.



2.3. ábra. Megerősítéses tanulás [2]

2.1.3. Nem ellenőrzött tanulás

A nem ellenőrzött tanulás esetében nem állnak rendelkezésünkre adott bemenethez tartozó kívánt válaszok. A hálózatnak a bemenetek és a kimenetek alapján kell valamilyen viselkedést kialakítania, a környezetből nincs semmiféle visszajelzés, ami a hálózat viselkedésének helyességére utalna. Ebben az esetben is rendelkezésünkre áll egy tanító halmaz, de csak bemeneteket tartalmaz, és nem tartalmazza az elvárt kimeneteket.

2.1.4. Analitikus tanulás

Az analitikus tanítás során a megfelelő viselkedést biztosító hálózat kialakítása elméleti úton, a feladatból határozható meg. Ebben az esetben nem is beszélhetünk tanulásról, a hálózat megfelelő kialakítása nem lépésenként, a környezetből szerzett információ fokozatos felhasználása révén, hanem analitikus módszerekkel végezhető el.

2.2. Perceptron és adaptív lineáris elem

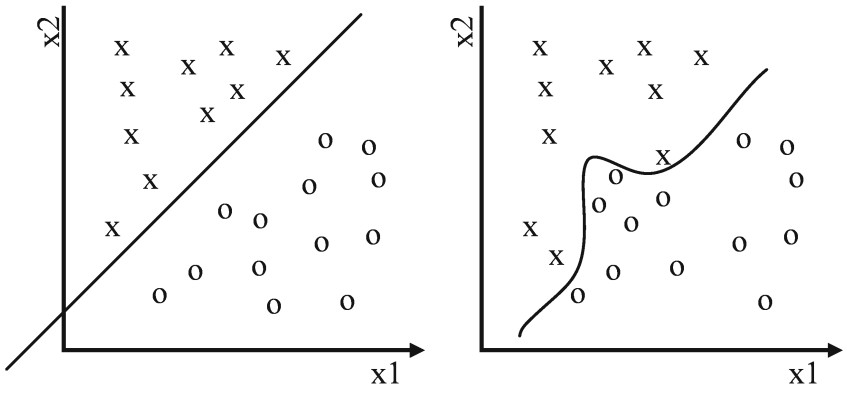
A következőkben bemutatásra kerül az egyszerű Perceptron és ADALINE hálók struktúrája és az LMS (Least Mean Square) tanítási algoritmus. Az LMS legkisebb négyzetek módszere a mérések matematikai feldolgozásában használt eljárás, a neuronhálók esetében az elvárt kimenet és számított kimenet különbségének a négyzetösszegét igyekszik minimalizálni.

2.2.1. Perceptron

Az egyszerű perceptron csak egyszerűbb gyakorlati feladatok megoldására alkalmazható (a bináris kimenetből és lineáris szeparálási képességéből adódóan). Az egyszerű perceptron kiindulási pont a többrétegű perceptron típusú neuronhálók (MLP) tanulmányozásában.

A perceptron az egyike a legelső és legegyszerűbb mesterséges neurális hálózatoknak. Az előrecsatolt többrétegű hálózatok olyan speciális változatának tekinthető, amely egyetlen rétegből és azon belül egy vagy több processzáló elemből áll. Az egy processzáló elemből álló változatot, lépcsőugrás aktivációs függvénnyel, szokás egyszerű perceptronnak is nevezni. A perceptront eredetileg Rosenblatt javasolta egy olyan hálózatként, amely

képes arra, hogy megfelelő beállítás, tanítás után két lineárisan szeparálható mintahalmazt szétválasszon [8]. A lineáris szeparálhatóság azt jelenti (2.4. ábra), hogy a bemeneti mintateret egy síkkal (hipersíkkal) két diszjunkt tartományra tudjuk bontani úgy, hogy a két tartomány eltérő osztályba tartozó bemeneti mintapontokat tartalmazzon.



2.4. ábra. Lineárisan szeparálható, valamint lineárisan nem szeparálható osztályok

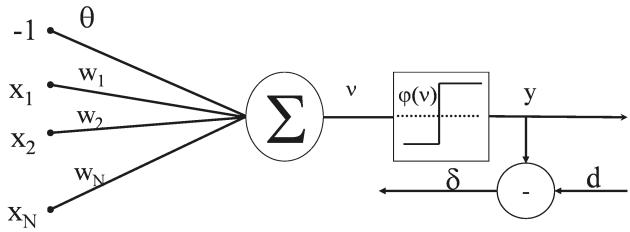
Az egyszerű perceptron (2.5. ábra) tehát képes lineárisan szétválasztható mintákat két osztályba sorolni. A XOR feladat, amint a 2.6. ábra is szemlélteti, lineárisan nem szétválasztható, Minsky-problémaként is említik [9]. A későbbiekben az egyszerű perceptront továbbfejlesztették többelemű, illetve többrétegű hálózatokká, amelyek képességei az egyszerű perceptron képességeit messze felülmúlják. Az *egyszerű perceptron* egy partikuláris esete a McCulloch-Pitts mesterséges neuron modellnek, amikor az aktiváló függvény egy lépcsőfüggvény.

Az egyszerű perceptron típusú neuron szerepe osztályozni a bemeneteket a két lehetséges osztály egyikébe ($y = +1$ vagy $y = -1$).

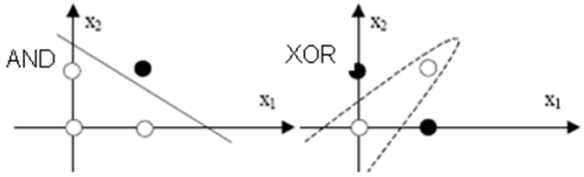
A kimenetet a következőképpen kell kiszámítani (2.1), (2.2):

$$\nu = \sum_{i=1}^{N} w_i x_i - \theta \tag{2.1}$$

$$y = \varphi(\nu) = \begin{cases} -1 & \text{ha} \quad \nu < 0 \\ 1 & \text{ha} \quad \nu \geq 0 \end{cases} \tag{2.2}$$



2.5. ábra. Egyszerű perceptron



2.6. ábra. AND, XOR példa

Az osztályok a (2.3) egyenlet által megadott hipersíkkal vannak szétválasztva:

$$\sum_{i=1}^{N} w_i x_i - \theta = 0 \tag{2.3}$$

Ennek egy partikuláris esete $N = 2$.

$$w_1 x_1 + w_2 x_2 - \theta = 0 \tag{2.4}$$

A (2.4) egy egyenes egyenletét írja le. Ebben az esetben a bemeneti vektorok egy egyenessel vannak elválasztva. Egy példa egy ilyen típusú feladatra az ÉS logikai függvény, ellenpélda pedig a XOR, ami nem szeparálható egy egyenessel (2.6. ábra).

$N = 3$ esetében az elválasztás egy síkkal, $N > 3$ esetében hipersíkkal történik. Az egyszerű perceptron csak lineárisan szeparálható feladatok megoldására alkalmazható.

Változók, paraméterek jelölése:
$\underline{X} = [1, x_1, x_2, \ldots, x_i, \ldots, x_N]$ – bemeneti vektor
$\underline{W} = [-\theta\ w_1, w_2, \ldots, w_i, \ldots, w_N]$ – súlyzóvektor
w_i – az i-edik bemenethez kapcsolódó súlytényező
N – neuron bemeneteinek a száma

θ – küszöb (referenciabemenet)
s – inger (súlyozott összeg)
y – a neuron kimenete
μ – tanítási együttható, $0 < \mu \leq 1$.
φ – aktivációs függvény
d – elvárt kimenet
δ – a hiba az elvárt kimenet és a neuron kimenete közötti különbség $\delta = d - y$.

2.2.1.1. Perceptron típusú neurális háló tanításának lépései

1. súlytényezők inicializálása: $w_i = 0$, $i = 1, \ldots, N$, vagy véletlenszerűen;

2. a tanítóhalmazból egy (a következő) minta választása;

3. a háló (neuron) kimenetének kiszámolása a tanítóhalmazból választott mintára (2.5), (2.6)

$$s = w_1 x_1 + w_2 x_2 + \cdots + w_N x_N = \underline{W}^T \underline{X} \tag{2.5}$$

$$y = \varphi(s) = \varphi\left(\underline{W}^T \underline{X}\right) \tag{2.6}$$

4. a súlytényezők módosítása (2.7) – vektoriális forma, (2.8) – skaláris forma)

$$\underline{W}[k+1] = \underline{W}[k] + \mu(d - y) \tag{2.7}$$

$$w_i[k+1] = w_i[k] + \mu(d - y) \quad i = 1 \cdots N, \tag{2.8}$$

ahol az elvárt kimenet megválasztása a (2.9) szerint történik:

$$y = d(n) = \begin{cases} -1 & \text{ha} \quad x(n) \in C_1 \\ 1 & \text{ha} \quad x(n) \in C_2 \end{cases}; \tag{2.9}$$

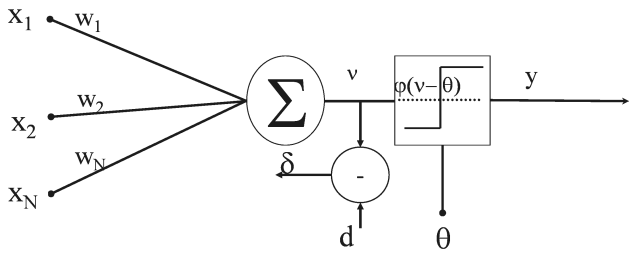
ahol k a tanítási lépés szám, C_1, C_2 a két halmaz, amelybe a tanítóminták sorolhatóak;

5. vesszük a következő elemet a tanító halmazból.

Addig ismételjük a ciklust, amíg az osztályozás minden bemeneti vektorra helyes.

2.2.2. Az ADALINE adaptív lineáris elem

Az ADALINE (adaptív lineáris elem) neuron (2.7. ábra) abban különbözik a perceptrontól, hogy a hibát nem a kimeneten, hanem a lineáris kimenet alapján számoljuk ki [10]. Az ADALINE felépítését a következő ábra szemlélteti:



2.7. ábra. Az ADALINE szerkezete

Az ADALINE típusú neuron tanítása

1. a súlytényezők inicializálása a 0-dik tanítási ciklusban $w_j[0] = 0$, $j = 1, \ldots, N$;

2. a háló kimenetének a kiszámolása egy bemeneti vektorra a tanító halmazból (2.10)

$$y = \varphi\left(\underline{W}^T \underline{X}\right) = \varphi\left(\sum_{j=0}^{N} w_j x_j\right); \tag{2.10}$$

3. kiszámoljuk a hibát (δ), mint a várt kimenet, d és a lineáris kimenet vagy inger (s) közötti különbséget (2.11)

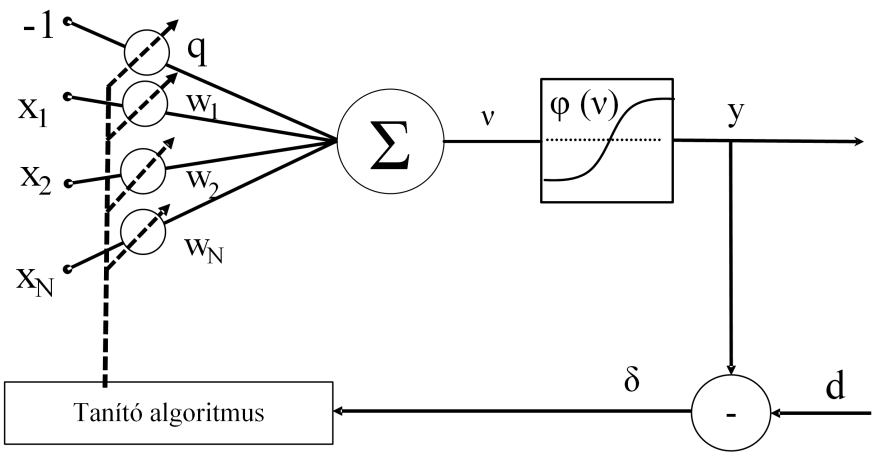
$$\delta = d - s; \tag{2.11}$$

4. módosítjuk a súlytényezőket a delta szabályt alkalmazva (2.12)

$$w_i\left[k+1\right] = w_i\left[k\right] + \mu\left(d - s\right); \tag{2.12}$$

5. vesszük a következő elemet a tanító halmazból; szintén addig ismételjük a tanítási ciklusokat, amíg a hiba egy előírt küszöbérték alá nem csökken.

2.3. Feldolgozó elem nemlineáris aktiváló függvénnyel



2.8. ábra. Nemlineáris függvényt tartalmazó neuron

A súlytényezők módosítása abban az esetben, ha a neuron nemlineáris aktivációs függvényt tartalmaz (2.8. ábra), a (2.13) képlet alapján alakul.

$$\underline{W}\left[k+1\right] = \underline{W}\left[k\right] + \mu\delta f(s)'\underline{X}\left[k\right] \tag{2.13}$$

A (2.14) képlet szerint számolt hibát alkalmazzuk a (2.13) egyenletben.

$$\delta\left[k\right] = d\left[k\right] - y\left[k\right] \tag{2.14}$$

ahol az elvárt érték és a háló kimenetén kapott érték közötti különbség, *f(s)'* az aktivációs függvény deriváltja, $\underline{W}$ a súlyvektor, $\underline{X}$ a bemeneti vektor, μ a tanítási együttható, a k lépések száma.

2.4. Perceptron típusú neuron példával való szemléltetése

Tervezzünk egy neuronhálót, amelyet 3×3 mátrixon ábrázolt karakterek azonosítására alkalmazunk. A feladat megoldásának fontosabb lépései:

1. a tanító halmaz felépítése, a tanítandó karakterek tervezése (meghatározása) vagy egyszerűen egy adatbázisból való letöltése
2. a háló be- és kimeneteinek a meghatározása;
3. a neuronháló struktúrájának a meghatározása;
4. a háló tanítása;
5. a háló tesztelése csak előhívási fázist alkalmazva;
6. a neuronháló alkalmazása.

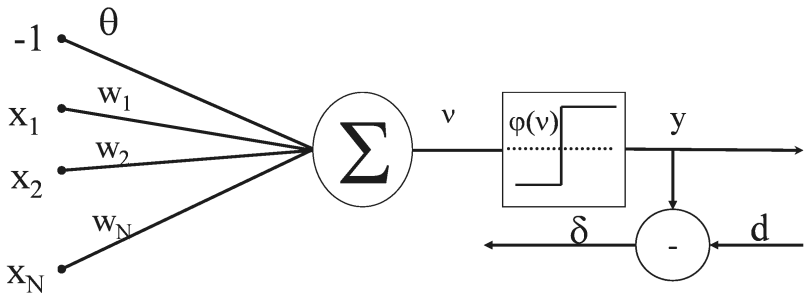
Végezzük el a háló tanítását két esetre, majd hasonlítsuk össze a kapott megoldásokat. Első változatban két karakterre (2.9. ábra), majd négy karakterre (2.12. ábra) végezzük el a háló tanítását. Két karakteres megoldásnál alkalmazzuk a következő karaktereket: T és H.

2.9. ábra. Tanítóhalmaz 3x3 bemeneti mátrixon T és H betűkre

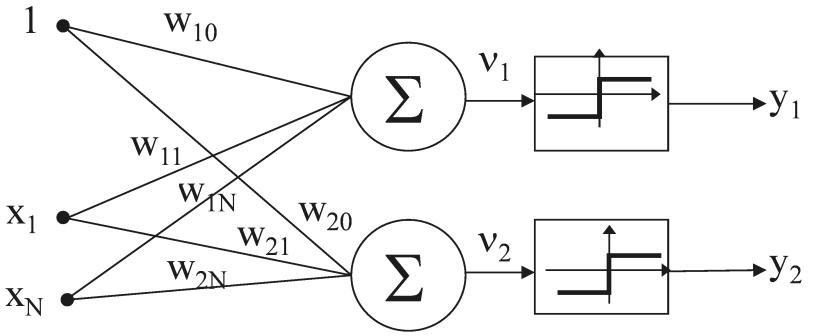
A neuronháló bemeneteinek a számát karakterpixeleinek száma (a neuronháló változó bemenetei), valamint egy konstans értékű bemenet együttesen adja. A neuronháló kimenetei számának a meghatározására két megoldást javasolunk: a hálónak egy (2.10. ábra), illetve két kimenete (2.11. ábra) legyen.

Ha két karakterünk van, a háló kimenetét binárisan lehet kódolni. Például a T karakternek feleljen meg logikai igaz, a H karakternek pedig logikai hamis. Ennek megfelelően a háló egyetlen PERCEPTRON típusú neuronnal megvalósítható. Ebben az esetben a háló tanítása könnyen elvégezhető. Igen ám, de ennek a megoldásnak van egy hátránya. Ha hibás karaktert viszünk

a bemenetre, attól függően, hogy melyik karakterhez hasonlít jobban, vagy T, vagy H karakternek ismeri fel a háló, a valóságban pedig egy harmadik karakterről van szó.



2.10. ábra. Két karaktert osztályozó neuron

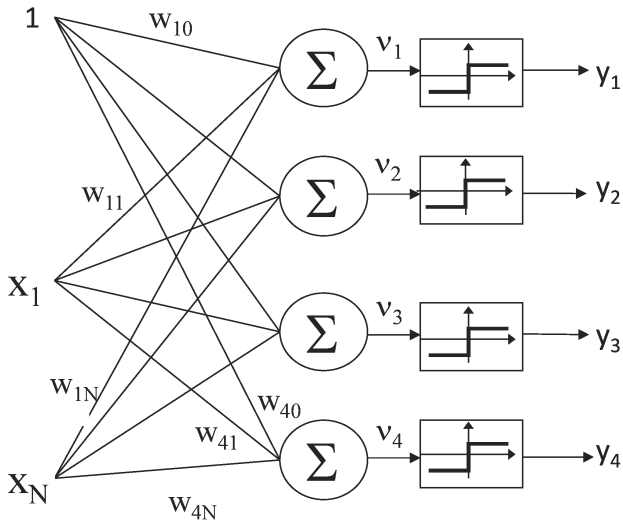


2.11. ábra. Két karakter osztályozása két kimenetű neuronhálóval

A hibás karaktereknek a felismerésére a megoldás a következő: annyi kimenete legyen a neuronhálónak, mint az osztályozandó karakterek száma. Ebben a megközelítésben minden egyes karakterhez hozzárendelünk egy perceptron típusú neuront. Ha a helyes karakter kerül a háló bemenetére, a karakternek megfelelő neuron kimenete aktív lesz, míg hibás karakter esetében a háló kimenetei nem lesznek aktívak.

Ha több karaktert kell osztályozni a hálónak, annyi kimenete lesz, ahány karakter, tehát annyi, mint az osztályozandó mintáknak a száma.

2.12. ábra. Példa 4 karakterre 3x3-as mátrixon



2.13. ábra. Négy kimenetű neuronháló

2.1. táblázat. Tanítóhalmaz 1 kimenetű változatra

Bemenetek ($\underline{x}$)									**Elvárt kimenetek**
x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	d
1	1	1	0	1	0	0	1	0	1
1	0	1	1	1	1	1	0	1	0

2.2. táblázat. Tanítóhalmaz 2 kimenetű változatra

Bemenetek									**Elvárt kimenetek** ($\underline{d}$)	
x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	d_1	d_2
1	1	1	0	1	0	0	1	0	1	0
1	0	1	1	1	1	1	0	1	0	1

A következő 2.1., 2.2., 2.3. táblázatokban a tanítóhalmazok vannak feltüntetve a három esetnek megfelelően. A táblázatokban a fekete '1' az alakzatpont, a fehér '0' pedig a háttérpontot jelenti. A 3x3-as sablon a 2.1., 2.2., 2.3. táblázatokban sorvektorként van felírva.

A kimenetek kódolása függ az alkalmazott aktivációs függvény típusától. Küszöbfüggvényt (vagy szigmoidot) alkalmazva a kimeneteket 0, 1-gyel kódoljuk, lépcsőfüggvény (vagy tangens hiperbolikus) esetében pedig -1, 1-gyel. A tanítás több tanítási ciklust tartalmaz.

Egy tanítási ciklus a következő lépéseket tartalmazza:

2.3. táblázat. Tanítóhalmaz 4 kimenetű változatra

Bemenetek ($\underline{x}$)									**Elvárt kimenetek** ($\underline{d}$)			
x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	d_1	d_2	d_3	d_4
1	1	1	0	1	0	0	1	0	1	0	0	0
1	0	1	1	1	1	1	0	1	0	1	0	0
1	0	0	1	0	0	1	1	1	0	0	1	0
1	1	1	0	0	1	0	0	1	0	0	0	1

1. a tanító halmazból a következő karakter bevitele a neuronháló bemenetére;
2. a háló kimenetének kiszámolása;
3. a hiba kiszámolása az elvárt kimenet és a háló kimenete között;
4. a súlytényezők tanítása a tanítóhalmaz minden elemére;
5. a hiba összegzése (és ábrázolása) a teljes tanítóhalmazra;
6. a tanítási ciklusok megismétlése, amíg a hiba egy meghatározott küszöbérték alá nem csökken.

A tanítási ciklusokat addig kell ismételni, amíg a hiba egy meghatározott küszöbérték alá nem csökken. Mivel a tanítási ciklusoknak a száma ismeretlen, egyszerűbb, ha előre meghatározzuk a tanítási ciklusok számát, majd ellenőrizzük, hogy a hiba eléggé lecsökkent-e. Ha a hiba a tanítás után a meghatározott küszöbérték alá csökkent, a tanítást befejezettnek minősítjük. Ellenkező esetben a tanítási együtthatót módosítva, a tanítási ciklusok számát növelve megismételjük/folytatjuk a neuronháló tanítását. A tanító halmazon kívül egy teszthalmazt is fel kell építeni. A teszthalmaz nem csak ideális, hanem zajos, hibás karaktereket is tartalmaz. A teszthalmaz minden elemére kiszámoljuk a háló kimenetét. Minden egyes tanítási ciklusban minden egyes karaktert beviszünk a háló bemenetére. A tanítóhalmaz két külön mátrixban van tárolva. A T mátrix tartalmazza a karakterek, a d mátrix pedig az elvárt kimenetek kódolását. A T mátrix méretét tanulmányozva azt látjuk, hogy eggyel több oszlopot tartalmaz, mint a 3-as táblázatban bemutatott tanító halmaz. Ennek oka, hogy a konstans értékű bemenet is bekerült a tanító halmazba. Amint látható, a T mátrix első oszlopának minden egyes eleme 1-es. Az első oszlop reprezentálja a konstans értékű bemeneteket. A tanítás során a konstans értékeket is tanítani kell. A konstans értéknek a tanító halmazba való bevitelével azt érjük el, hogy ehhez a bemenethez kapcsolódó súlytényező reprezentálja a konstans értéket, aminek a tanítása egyszerre fog történni a változó bemenetekhez kapcsolódó súlytényezőkkel.

3. fejezet

Mesterséges neuronháló szerkezete

Ebben a fejezetben a mesterséges neuronhálók topológiájával kapcsolatos ismereteket mutatunk be, neuronok összekapcsolását, teljesen összekötött, illetve parciálisan összekötött topológiákat szemléltetünk.

3.1. Neuronhálók topológiája

A neuronhálókban a neuronok elrendezése nagyon változatos. Ebben a részben főként a neuronok lehetséges összekapcsolásának módjaival foglalkozunk. A neuronhálók összekapcsolásának elsajátítása alapot biztosít a könyv következő részének a megértéséhez, a különböző típusú neuronhálók tanításához. A mesterséges neurális hálózatok alapjában véve absztrakt entitások. Leírhatóak matematikailag és megvalósíthatóak különböző változatban: szoftveresen vagy hardveresen. A neuronhálóban a neuronok közötti kapcsolatokat a neuronháló topológiája határozza meg. A mesterséges neuronhálók topológiája úgy tekinthető, mint a neuronok közötti reláció kapcsolataik révén. A topológia kifejezésre alternatíva a neuronháló szerkezete vagy a neuronháló architektúrája. Általában a neuronhálózat topológiája irányított gráffal írható le. A neuronok a gráf csomópontjai, míg a neuronháló bemeneteit, kimeneteit és a neuronok közötti kapcsolatokat a gráf élei reprezentálják. Az élek a neuron bemenetétől a kimenet felé vannak irányítva. Általában egy neuronhálóban három típusú neuront különböztethetünk meg (3.1. ábra):

1. **Bemeneti neuronok**: a bemeneti neuronoknak általában nincsen jelfeldolgozó, processzáló szerepük. A bemeneti neuronok a hálózat bemenetéről kapják a jeleket és más neuronok bemenetére továbbítják.

2. **Kimeneti neuronok**: a neuronháló kimenetén a környezet felé közvetítik az információt. A kimeneti neuronok bemenetét általában a neuronhálóban található egyéb neuronok szolgáltatják.

3. **Rejtett neuronok**: bemeneteikkel és kimeneteikkel szigorúan csak más neuronokhoz kapcsolódnak.

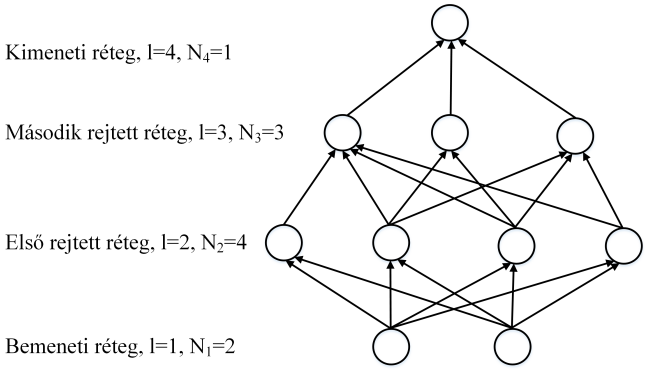
A legtöbb neuronháló esetében a neuronok rétegekbe vannak szervezve. Van egy pár kivétel, amikor a neuronhálóban a neuronok nem rétegekbe vannak szervezve, viszont ezek a neuronhálók is értelmezhetőek többrétegű topológiájú rendszerekként [11].

Egy többrétegű neuronháló esetében, a neuronok osztályozása szerint megkülönböztetünk:

1. bemeneti réteget: bemeneti neuronokat tartalmaz;

2. kimeneti réteget: kimeneti neuronokat tartalmaz;

3. rejtett réteget: rejtett neuronokból épül fel.



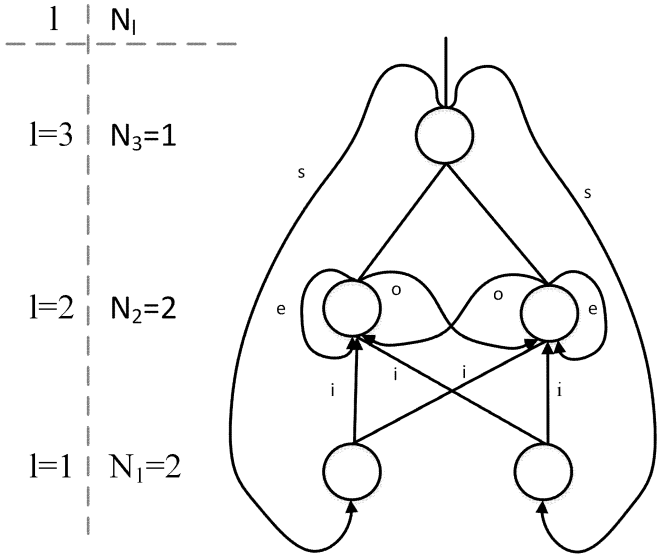
3.1. ábra. Többrétegű neuronháló szemléltetése

Többrétegű neuronháló esetében a rétegek rendezettek és megszámlálhatók, a bemeneti réteg indexe 1, első rejtett réteg indexe 2, következő rejtett

réteg indexe 3, és az utolsó kimeneti réteg indexe L, mely a neuronhálóban található rétegeknek a száma. Rétegenként ismerjük a neuronok számát $N^{<1>}, N^{<2>}, \ldots, N^{<L>}$. A 3.1. ábrán egy többrétegű neuronháló van szemléltetve.

3.1.1. Neuronok összekapcsolása

A neurális hálózatok topológiája, összekapcsolási struktúrája (3.2. ábra) meghatározza a neuronok összekapcsolódásának módját.



3.2. ábra. Neuronok közötti kapcsolatok

A többrétegű neuronhálókra alapozva, különböző típusú kapcsolatok lehetségesek [4], [6], [11].

- rétegek közötti kapcsolat (i – interlayer connection): szomszédos rétegekben található neuronok közötti kapcsolat;
- laterális (o), intralayer connection: egy rétegen belüli neuronok közötti kapcsolat, a réteg kimenete a rétegben található neuronok bemenetére van csatolva;
- elemi visszacsatolás (e): egy neuron kimenetének saját bemenetére való visszacsatolása (ez egy speciális esete a rétegen belüli kapcsolatoknak);

– supralayer connection (s): nem szomszédos rétegekben található neuronok közötti kapcsolat; általában egy réteg kimenetének a visszacsatolása egy előbbi réteg bemenetére, de nem a saját réteghez tartozó neuronok bemeneteire.

Minden egyes kapcsolathoz egy súlytényező van rendelve, egy súlyozási tényező, amely meghatározza a bemenet fontosságát. A súlytényező egy skalár érték, pozitív érték esetén gerjesztő, negatív érték esetén gátló szerepe van. Ha a kapcsolathoz tartozó súlytényező értéke zérus, úgy tekinthető, hogy az adott időpontban a kapcsolat nem létezik.

Teljesen összekötött topológia: akkor beszélünk teljesen összekötött topológiáról, ha a neuronhálóban a lehetséges kapcsolatok közül az összes jelen van. A különböző neuronháló-topológia összehasonlítására fontos, hogy ismerjük, hogy egy adott topológia összesen hány kapcsolatot tartalmaz. A legtöbb neuronháló esetében a kapcsolatoknak a száma megegyezik a súlytényezőknek a számával, vagyis minden egyes kapcsolathoz egy súlytényező van rendelve. Egyes típusú neuronháló, mint például a konvolúciós neuronhálók [1] esetében, beszélhetünk súlytényező-megosztásról, amikor egy kapcsolatcsoport oszt meg egy súlytényezőt. Ebben az esetben több kapcsolat ugyanazt a súlytényezőt osztja meg, a súlytényezőknek a száma sokkal kevesebb a kapcsolatok számához képest. Amikor megszámoljuk a súlytényezők számát egy hálóban, szintén el kell dönteni, hogy a neuron bias (eltolási, offset) értékeit is figyelembe vegyük-e vagy nem. Amint a neuron bemutatásánál már szemléltettük, a bias is lényegében egyfajta súlytényezőnek is tekinthető, és gyakran a bias érték adaptálása is ugyanolyan módszerrel történik, mint a súlytényezők tanítása.

Parciálisan összekapcsolt topológia: egy érdekes alternatívát jelenthet a teljesen összekapcsolt neuronhálókhoz képest; csökkentett mértékű redundancia mellett fokozott hatékonyságot biztosít.

3.1.2. Előrecsatolt topológiájú neuronhálók

Többrétegű előrecsatolt neuronhálók esetében minden egyes réteghez tartozik:

– súlymátrix $\left(\underline{W}^{<l>}\right)$,
– réteg bemenete (réteg bemeneti vektora) $\left(\underline{X}^{<l>}\right)$,
– réteghez tartozó ingervektor $\left(\underline{S}^{<l>}\right)$,
– réteg kimenete $\left(\underline{Y}^{<l>}\right)$,

– a réteg kimenetén a hibavektor $\left(\underline{\delta}^{<l>}\right)$, ahol l a réteg száma.

Minden egyes réteghez egy rétegszámot rendelünk, és a réteg számával indexeljük a változókat. Ezen index alapján különböztetjük meg, hogy az adott változó melyik réteghez tartozik. Egy több neuront tartalmazó rétegben, ahhoz, hogy egy bemenet és egy neuron között azonosítani lehessen egy kapcsolatot, a súlytényezőket két indexszel kell ellátni. Az első index meghatározza a neuron számát, amelyhez a bemenet kapcsolódik, illetve a második index meghatározza a bemenet számát. Például egy kétrétegű neuronháló esetében a következőképpen alakul a változók jelölése:

Első réteg változóira alkalmazott jelölések:
$\underline{X}^{<1>}$ $\underline{W}^{<1>}$ $\underline{S}^{<1>}$ $\underline{Y}^{<1>}$ $\underline{\delta}^{<1>}$

A tanítás során minden réteghez szükség van egy kimeneti hibavektorra, amelyet δ-val jelölünk. A második réteghez kapcsolódó változók:
$\underline{X}^{<2>}$ $\underline{W}^{<2>}$ $\underline{S}^{<2>}$ $\underline{Y}^{<2>}$ $\underline{\delta}^{<2>}$

Általánosan az l-edik réteghez tartozó változók:
$\underline{X}^{<l>}$ $\underline{W}^{<l>}$ $\underline{S}^{<l>}$ $\underline{Y}^{<l>}$ $\underline{\delta}^{<l>}$

Egy réteghez tartozó súlymátrix a következőképpen van felépítve: a rétegben található neuronok súlyvektorait oszloponként egymás mellé helyezzük a mátrixban.

$\underline{X}^{<l>} = x_0^{<l>}, x_1^{<l>}, x_2^{<l>} \cdots x_i^{<l>} \cdots x_{N^{<l-1>}}^{<l>}]$ – bemeneti vektor

$\underline{S}^{<l>} = [s_1^{<l>}, s_2^{<l>} \cdots s_i^{<l>} \cdots s_{N^{<l>}}^{<l>}]$ – ingervektor

$\underline{Y}^{<l>} = [y_1^{<l>}, y_2^{<l>} \cdots y_i^{<l>} \cdots y_{N^l}^{<l>}]$ – kimeneti vektor

$\underline{\delta}^{<l>} = [\delta_0^{<l>}, \delta_1^{<l>}, \delta_2^{<l>} \cdots \delta_i^{<l>} \cdots \delta_{N^l}^{<l>}]$ – réteg kimenetén a hiba

$$\underline{W}^{<l>} = \begin{bmatrix} -\theta_1^{<l>} & -\theta_2^{<l>} & \cdots & -\theta_j^{<l>} & \cdots & -\theta_{N^{<l>}}^{<l>} \\ w_{1,1}^{<l>} & w_{2,1}^{<l>} & \cdots & w_{j,1}^{<l>} & \cdots & w_{N^{<l>},1}^{<l>} \\ w_{1,2}^{<l>} & w_{2,2}^{<l>} & \cdots & w_{j,2}^{<l>} & \cdots & w_{N^{<l>},2}^{<l>} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{1,N^{<l-1>}}^{<l>} & w_{2,N^{<l-1>}}^{<l>} & \cdots & w_{j,N^{<l-1>}}^{<l>} & \cdots & w_{N^{<l>},N^{<l-1>}}^{<l>} \end{bmatrix}$$

– súlymátrix

Többrétegű topológiájú neuronháló esetében a neuronháló kimenetét a bemenettől a kimenet felé számoljuk. Első lépésben kiszámoljuk az első

réteg ingerét, majd kimenetét. Az első réteg kimenete a második réteg bemenetét képezi. Kiszámoljuk a következőben a második réteg ingerét, illetve a második réteg kimenetét, amely jelen esetben a neuronháló kimenetét is jelenti.

3.1.3. Visszacsatolást tartalmazó neronhálók

Az előrecsatolt neuronhálók esetében a feldolgozás a bemenettől a kimenet irányába történik. A neuronháló kimenete csak a neuronháló bemenetére kapcsolt pillanatnyi értékektől függ. A visszacsatolt vagy rekurrens neuronhálók memóriaként viselkednek. Összehasonlítva egy sorrendi hálózattal, mely a visszacsatoló ágban memóriát, tároló elemet tartalmaz, a visszacsatolással lényegében állapotokat iktatunk a neuronhálóba. A visszacsatolt neuronhálókat bonyolultabb, dinamikus rendszerek modellezésére, zajos jelminták visszaállítására vagy akár osztályba való sorolására is alkalmazzák. A visszacsatolt neuronhálók közül a Hopfield-hálók szerkezeti felépítését, tanítását és alkalmazását a 8. fejezet keretében részletezzük. A visszacsatolt neurális hálózatok a természetes nyelvfeldolgozásban az egyik leggyakrabban alkalmazott hálózatnak számítanak az ígéretesnek bizonyuló elért eredményeknek köszönhetően.

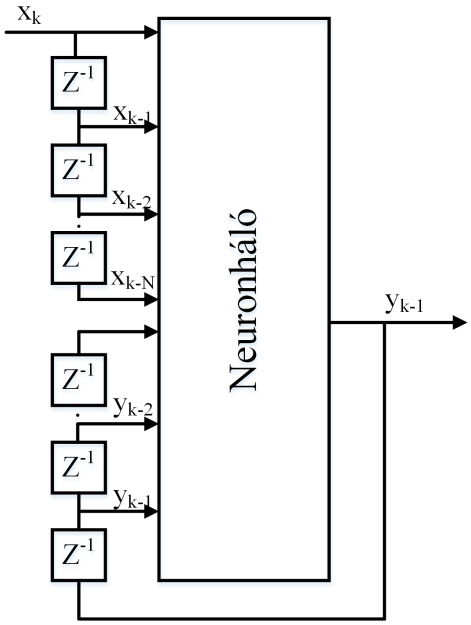
A rekurrens neurális hálóknak a természetes nyelvfeldolgozásban való alkalmazására két fő megközelítés áll:

- a hálóval becsülhetjük vagy kitalálhatjuk a mondatokat és javíthatjuk a hibát a becslés során,
- egy adott partikuláris műfajnak megfelelően tanítható a neuronháló és a műfajnak megfelelő szöveg generálható.

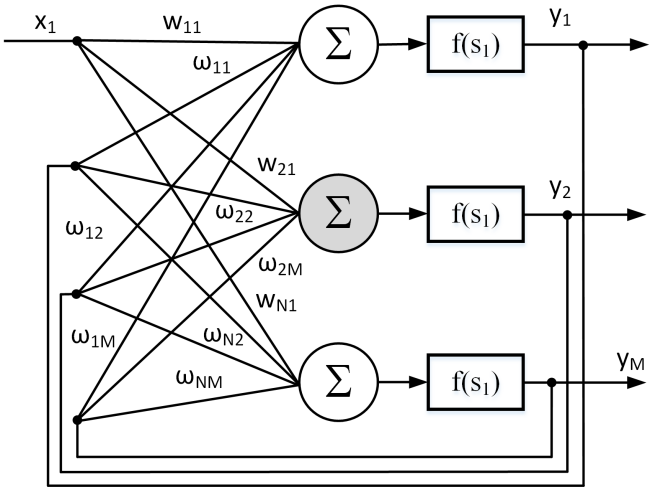
3.1.4. Időfüggő vagy idővariáns neuronhálók

Az időfüggő hálók többféleképpen alakíthatóak ki. Egy lehetséges változat, hogy a neuronháló bemenetére a modellezendő rendszer időben késleltetett bemeneteit, illetve a késleltetett kimeneteit vezetjük be (3.3. ábra).

Egy másik lehetséges változat, hogy például egy többrétegű perceptron típusú neuronháló kimeneteit visszacsatoljuk a neuronháló bemeneteire (3.4. ábra).

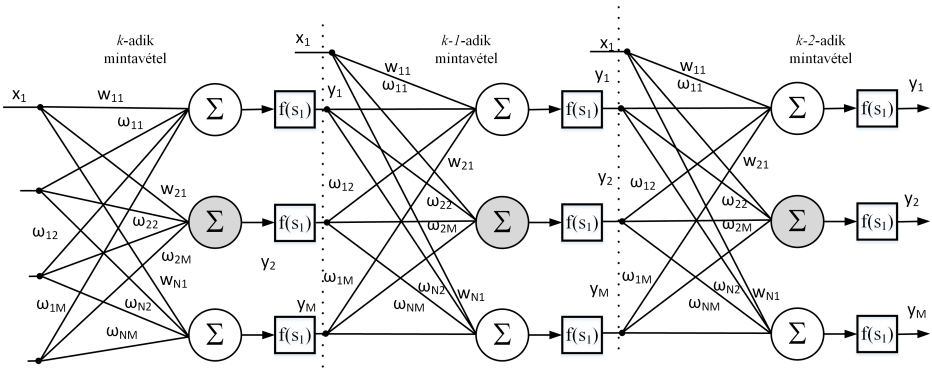


3.3. ábra. Időfüggő neuronháló, késleltetett bemenetek



3.4. ábra. Időfüggő neuronháló kialakítása a kimenet visszacsatolásával

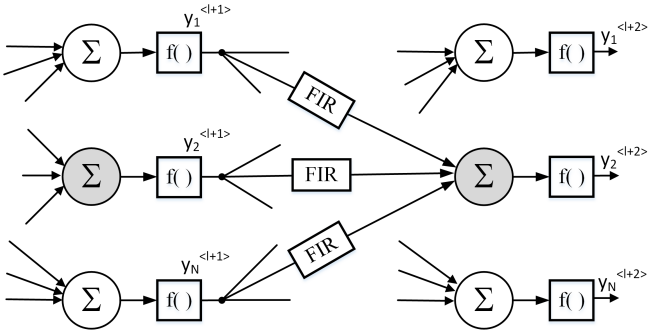
Ebben az esetben a kimenetszámítást, de főleg a tanítás során a visszacsatolt rendszermodellt kitekerjük és úgy kezeljük, mint egy előrecsatolt neuronhálót. A kitekerés során a különböző részekben ugyanazok a súlytényezők vannak megosztva. Az alábbi példában egy egyrétegű neuronhálóval szemléltetjük a kiterített neuronháló szerkezetét (3.5. ábra). A neuronhálónak a szemléltetett ábrán egy bemenete van, és mindegyik kimenete vissza van csatolva a neuronháló bemeneteire. A bemenetekhez kapcsolódó súlytényezőket w-vel, míg a visszacsatolt bemenetekhez kapcsolódó súlytényezőket ω-val jelöltük.



3.5. ábra. Kiterített neuronháló

Az előreszámolás, vagyis a bemenettől a kimenet felé való számolás során a megosztott súlytényezők értéke megegyezik az egyes részekben. A visszaterjesztéskor, amely a kimenetszámítástól a bemenet felé történik, a súlytényezők más és más értékeket vesznek fel, ugyanis az utolsó rétegbeli értékek alapján számoljuk az új súlytényezők értékeit a hiba-visszaterjesztés módszerével. Az előírt hibát az utolsó rész kimenetén határozzuk meg, és innen áramoltatjuk vissza a kiterített neuronhálón. Az x bemeneteket az egyes kitekert részekben az adott mintavételbeli értékek határozzák meg. A következő kimenetszámítás során a legutolsó lépésben meghatározott súlytényezőpárokat alkalmazzuk.

Egy harmadik lehetséges kialakítását az időfüggő hálóknak a súlytényezőknek véges impulzus válasz (FIR) szűrővel való helyettesítése jelenti (3.6. ábra). A FIR szűrök digitális szűrők, amelyek impulzusválasza időben véges. A szűrő algoritmusa csak az aktuális és az előző bemeneti értéket veszi figyelembe. Nem rekurzív, konvolúciós vagy mozgó átlag szűröknek is nevezik [12].



3.6. ábra. Időfüggő neuronháló kialakítása FIR szűrővel

Az időfüggő hálók alkalmazása és tanítása jóval bonyolultabb, összehasonlítva az előrecsatolt neuronhálókkal. Alaposabb betekintést az időfüggő hálók szerkezetébe, tanításába és alkalmazásába a szakirodalom [4], [2] tanulmányozását követően nyerhetünk.

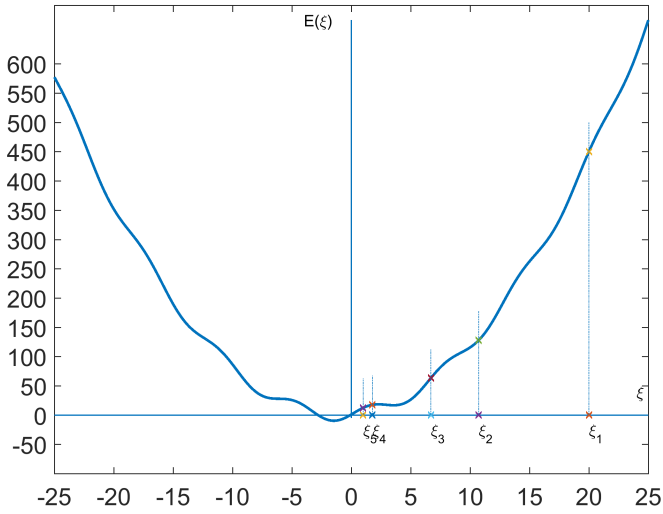
4. fejezet

Neuronhálók tanítása

4.1. Gradiens alapú tanítás

Ebben a részben a gradiens alapú optimalizálási eljárását mutatjuk be röviden. A gradiens alapú optimalizálási módszerek esetében nagyon fontos az alkalmazott költségfüggvény megválasztása. A fejezet keretében tárgyaljuk a neurális hálózatok tanítására alkalmazható fontosabb költségfüggvényeket. A neuronhálók tanítása során fontos a rendelkezésre álló adathalmaz megfelelő előkészítése, normalizálása. Ismertetjük a fontosabb normalizálási módszereket. A fejezet utolsó részében a regularizációs módszerekre világítunk rá.

A neuronhálók tanítása nem más, mint egy többváltozós optimalizálási eljárás egy előre jól meghatározott kritériumfüggvény $E(\underline{\xi})$ (költségfüggvény) alapján. A neuronhálók tanítására széles körben alkalmaznak különböző optimalizálási eljárásokat: gradiens alapú stratégiákat [13], [4], [6], evolúciós módszereket, genetikus algoritmusokat [14], részecske-raj optimalizálási (PSO) algoritmusokat [15], [16], [17]. A könyv keretében nincs lehetőség részletesen kitérni a különböző optimalizálási eljárásokra. Az egyik legegyszerűbb tanítási módszer a gradiens alapú módszerek alkalmazása a neuronháló paramétereinek hangolására. A gradiens nem más, mint a függvények deriválásának általánosítása többváltozós függvényekre. Aszerint, hogy minimumot vagy maximumot keresünk, a ξ paramétert úgy kell változtatni, hogy a költségfüggvény értéke, $E(\xi)$ csökkenjen vagy növekedjen (4.1. ábra).



4.1. ábra. Gradiens alapú tanítás, költségfüggvény ábrázolása a hangolható paraméter függvényében

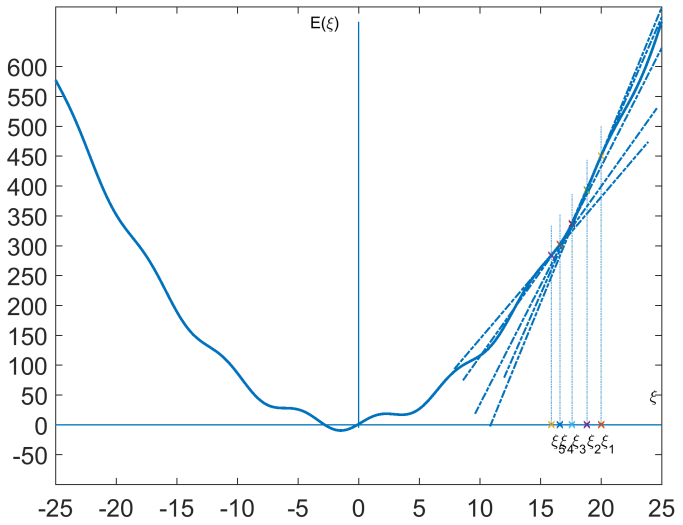
A gradiens alapú módszer alapján a ξ paraméter hangolása a (4.1) képlet szerint alakul a minimumkeresésre, illetve (4.2) szerint a maximumkeresés esetében.

$$\xi[k+1] = \xi[k] - \mu \frac{\partial E(\xi)}{\partial \xi} \tag{4.1}$$

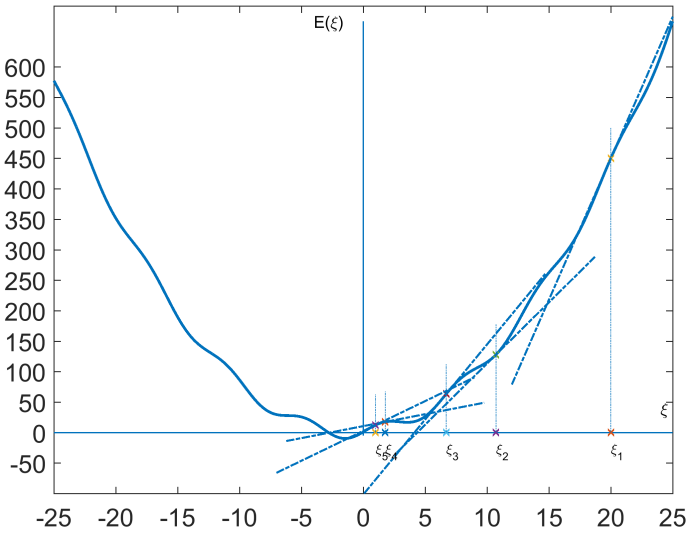
$$\xi[k+1] = \xi[k] + \mu \frac{\partial E(\xi)}{\partial \xi} \tag{4.2}$$

Az algoritmust a következőképpen értelmezhetjük: a $\xi[k]$ pontban egy érintőt húzunk a kritérium függvényhez, ami nem más, mint a függvény deriváltja az adott pontban. Ha a függvény deriváltja pozitív, növekvő ξ értékekre a költségfüggvény értéke is növekszik, illetve ha a függvény deriváltja negatív, növekvő ξ értékekre a költségfüggvény értéke csökken. Az említettek alapján, ha minimumot szeretnénk számolni, a gradiens változásával (deriválttal) ellentétes előjellel kell változtatni a ξ paraméter értékét.

A μ paramétert az optimalizálási eljárásnál lépésnek, neuronhálók tanításánál tanítási együtthatónak nevezzük. Ha a μ értéke kicsi, a ξ csak nagyon kis értékekkel változik és nagyon lassan közeledünk a megoldáshoz. Nagyon kicsi értékű tanítási együtthatók alkalmazása esetében a gradiens módszer nagyon könnyen elakadhat lokális minimumban, nagyon lassan, kicsi léptekkel közelít a megoldáshoz (4.2. ábra).

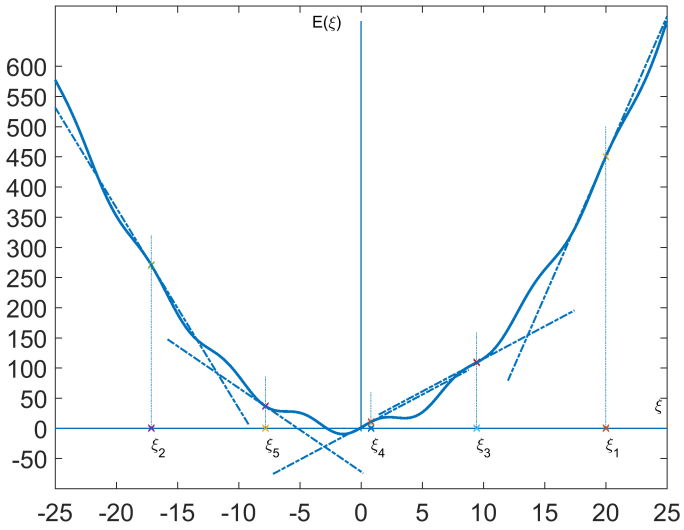


4.2. ábra. Nagyon kis értékű tanítási együttható

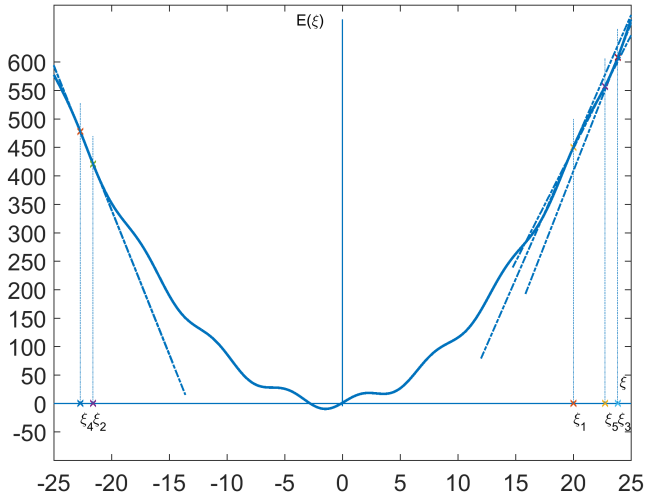


4.3. ábra. Költségfüggvényhez húzott érintő abban a pontban, amely körül keressük a szélsőértéket

Egy nagyobb értékű tanítási együttható alkalmazása esetén nagyobb léptekkel közelít a megoldáshoz (4.3. ábra), ha azonban a tanítási együttható túl nagy, a megoldás divergenssé válik.

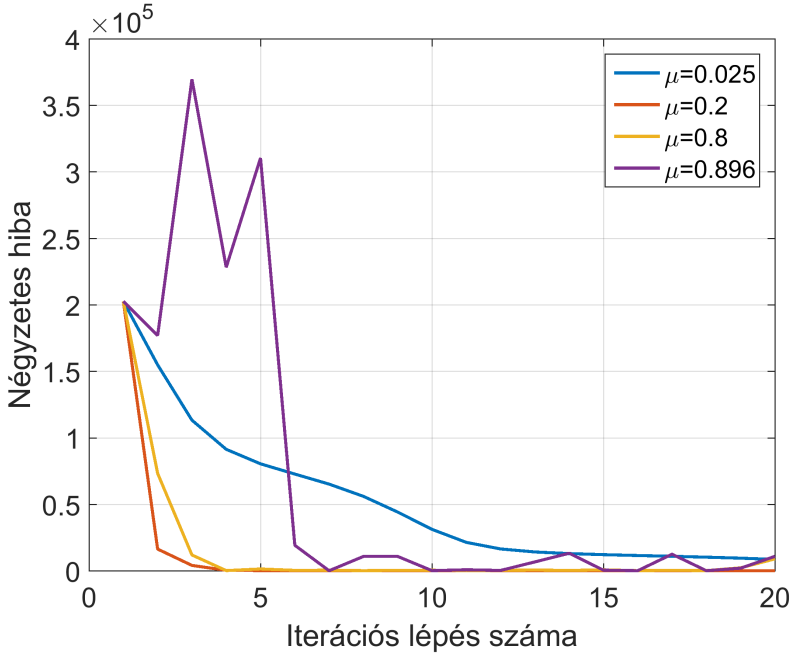


4.4. ábra. Túl nagy értékű tanítási együttható



4.5. ábra. A hiba értéke oszcillál

Egy adott μ tanítási együtthatót alkalmazva előfordulhat, hogy a ξ paraméter elkezd oszcillálni két határérték között (4.5. ábra). A hibagörbét ábrázolva az oszcilláció a hibagörbében is megfigyelhető. A 4.6. ábrán a kritériumfüggvény ábrázolása látható különböző μ értékekre.



4.6. ábra. Hiba alakulása különböző μ értékekre

4.2. Veszteségfüggvények/költségfüggvények

A gradiens alapú optimáláshoz szükség van egy olyan függvényre, költségfüggvényre (C – cost function), amely deriválható és számszerűen felméri a neuronháló hibáját a neuronháló számított kimenete és az elvárt kimenet ismeretében. A következő részben egy pár, a neurális hálózatoknál alkalmazott költségfüggvényt ismertetünk: [4], [18], [19], [20].

4.2.1. Átlagos abszolút eltérés

$$C_{ABS}(\xi) = \frac{1}{2}\sum_{i}^{N} |d_i^r - y_i(\xi)| \tag{4.3}$$

Az átlagos abszolút eltérés (4.3) nem alkalmazható gradiens alapú súlytényező-adaptációra, mivel nem deriválható, viszont alkalmazható például genetikus algoritmusok esetében a fitneszfüggény számításához. A genetikus algoritmusoknál a fitneszérték dönti el, hogy egy adott egyed mennyire jó megoldást biztosít egy vizsgált feladatra. A nagyobb fitneszértékű, jó képességű egyedek továbbfejlődnek, a kicsi fitneszértékűek, a rosszak viszont elpusztulnak.

4.2.2. Átlagos négyzetes eltérés

Az átlagos négyzetes eltérés (mean squared error) (4.4) az egyik leggyakrabban alkalmazott módszer a neuronháló hibájának a kiértékelésére.

$$C_{MSE}(\xi) = \frac{1}{2}\sum_{i}^{N} (d_i^r - y_i(\xi))^2 \tag{4.4}$$

Az átlagos négyzetes eltérés költségfüggvény-gradiense (4.5) a neuronháló kimenetére és a d^r mintára vonatkoztatva:

$$\nabla_{y_i} C_{MSE} = -(d_i^r - y_i(\xi)) \tag{4.5}$$

4.2.3. Bináris kereszt-entrópia

A kereszt-entrópia a (4.6) képlet szerint számolható.

$$C_{CE}(\xi) = -\frac{1}{n}\sum_{i} [y_i(\xi)\ln(d_i^r) + (1 - y_i(\xi))\ln(1 - d_i^r)] \tag{4.6}$$

A kereszt-entrópia költségfüggvény-gradiense (4.7) a neuronháló kimenetére és a d^r mintára vonatkoztatva [18]:

$$\nabla_{y_i} C_{CE} = -\frac{(d_i^r - y_i(\xi))}{(1 - y_i(\xi))(y_i(\xi))} \tag{4.7}$$

4.2.4. Exponenciális költségfüggvény

Az exponenciális költségfüggvény a (4.8) képlet szerint számolható.

$$C_{EXP}(\xi) = \tau \ \exp(\frac{1}{\tau}\sum_i (d_i^r - y_i(\xi))^2) \tag{4.8}$$

Az exponenciális költségfüggvény gradiense (4.9) a neuronháló kimenetére és a d^r mintára vonatkoztatva:

$$\nabla_{y_i} C = -\frac{2}{\tau}(d_i^r - y_i) C_{EXP}(\xi) \tag{4.9}$$

4.2.5. Hellinger-távolság

A Hellinger-távolság a (4.10) egyenlet szerint írható fel.

$$C_{HD}(\xi) = \frac{1}{\sqrt{2}}\sum_i (\sqrt{d_i^r} - \sqrt{y_i})^2 \tag{4.10}$$

A Hellinger-távolságra épülő költségfüggvény gradiense (4.11) a neuronháló kimenetére és a d^r mintára vonatkoztatva:

$$\nabla_{y_i} C = -\frac{\sqrt{d_i^r} - \sqrt{y_i(\xi)}}{\sqrt{2}\sqrt{y_i(\xi)}} \tag{4.11}$$

A felsorolt költségfüggvényeken kívül még számos változatot alkalmaznak a neuronhálók hibájának kiértékelésére, mint például a Kullback–Leibler-divergencián, általánosított Kullback–Leibler-divergencián, valamint Itakura–Saito-távolságon alapuló költségfüggvények alkalmazása. A ξ általánosan alkalmazott változó, amely a neuronháló hangolható paramétereit, súlytényezőit jelenti.

4.3. Lokális minimumokba való ragadás elkerülése

4.3.1. Adatok normalizálása

A neuronháló hatékony alkalmazása és tanítása szempontjából fontos a neuronháló bemeneteinek (kimeneteinek) normalizálása. Több módszer ismeretes, amelyeket a neuronhálóknál alkalmaznak az adathalmazok normalizálására [21], [4].

4.3.1.1. Min-Max normalizálás

A min-max normalizálás (4.12) a mért adatokat átalakítja egy $min_Q - max_Q$ intervallumra:

$$x' = \frac{x - min_x}{max_x - min_x} (min_Q - max_Q) + min_Q \tag{4.12}$$

4.3.1.2. Standard normalizálás

A standard normalizálás (4.13) (standardizálás) esetében úgy normalizáljuk az adathalmazt, hogy középértéke 0 legyen, a szórása pedig 1 minden egyes attribútumra:

$$x' = \frac{x - \overline{X}}{\sigma_X} \tag{4.13}$$

ahol x a bemenet, amelyet szeretnénk normalizálni, $\overline{X}$ a mintahalmaz átlaga, σ_X a mintahalmaz szórása.

4.3.1.3. Batch-normalizálás

A batch-normalizálás esetében [22] a hálózat folyamatosan számol egy futó átlagot és egy futó szórást a bemenetekről, amelyet felhasznál az új bemenetek normalizálására. A módszer alkalmazásával nagymértékben csökkenthető a konvergencia sebessége és csökken a túlillesztés esélye is. Akkor van szó túlillesztésről, amikor a neuronháló nagyon jó megoldást eredményez a tanítóhalmazban levő mintákra, és a tanítás előrehaladtával egyre rosszabb az eredmény azon mintákra, amelyekre nem történt tanítás.

4.3.1.4. Kiugró adatok kiejtése

A kiugró adatok gyakori jelenség és sok gondot okoz a feldolgozás során. Egyszerű és hatékony módszer a kiugró adatok kiejtése, figyelmen kívül hagyása.

4.4. Regularizáció

A neurális hálózatok, különösen a többrétegű rendszerek nagymértékben ki vannak téve a túlillesztés problémájának. A túlillesztés kiküszöbölésére alkalmazhatóak a különböző regularizációs technikák. A regularizáció alatt szabályozást értünk és több módon valósítható meg: a paraméterek számának a csökkentésével, a paraméterek értékének korlátozásával vagy a paraméterek jelentőségének csökkentésével [23]. A regularizáció során a költségfüggvényt ki kell bővíteni egy büntetőfüggvénnyel a (4.14) képlet szerint [1].

$$L\left(\xi\right) = E_s\left(\xi\right) + \lambda E_c\left(\xi\right) \tag{4.14}$$

ahol $E_s\left(w\right)$ megadja a rendszer működésének a hibáját, az $E_c\left(w\right)$ büntetőfüggvény, a λ regularizációs paraméter meghatározza, hogy a regularizációs tag milyen mértékben bünteti a költségfüggvényt.

4.4.1. L_2 Regularizáció

Egy egyszerű, viszont hatékony változat egy neuronháló regularizációjára az, hogy kiszámoljuk az L2 normát a súlytényezőkre és hozzáadjuk a költségfüggvényhez [1].

$$L\left(\underline{W}\right) = E_s\left(\underline{W}\right) + \lambda \left\|\underline{W}\right\|^{<2>} \tag{4.15}$$

A büntetőtag (4.15) ösztönzi, hogy a súlytényezők kis értéket vegyenek fel. Ha a λ értéke nagy, a súlytényezők értéke zérus körüli értéket vesz fel, csökkentve a modell varianciáját és növelve a bias értékeket. Ha a λ értéke kicsi, lehetővé tesszük, hogy a súlytényezők nagy értéket vegyenek fel, ezáltal növelve a modell varianciáját. Csak a működési hibát figyelembe véve, a paraméteradaptáció (4.16) a képlettel írható le:

$$\xi\left[k+1\right] = \xi\left[k\right] - \mu\frac{\partial E_s\left[k\right]}{\partial \xi\left[k\right]}, \tag{4.16}$$

ha alkalmazzuk a kettes norma alapú regularizációt, a paraméteradaptálás a (4.17) egyenlet szerint számítható ki:

$$\xi\left[k+1\right] = \xi\left[k\right] - \mu\frac{\partial E_s\left[k\right]}{\partial \xi\left[k\right]} - \frac{\mu\lambda}{n}\xi, \tag{4.17}$$

ahol n az egy lépésben alkalmazott tanító mintáknak a száma. A pillanatnyi hiba alapján való tanulás esetében $n = 1$.

4.4.2. L_1 Regularizáció

Az L_1 regularizáció (4.18) bünteti a költségfüggvényt a súlytényezők L_1 normájával.

$$L\left(\underline{W}\right) = E_s\left(\underline{W}\right) + \lambda\left|\underline{W}\right| \tag{4.18}$$

Ellentétben az L_2 regularizációval, ritka súlytényező mátrixokat (vektorokat) eredményezhet, amelyben a súlytényezők értéke zérus vagy nagyon közel van zérushoz.

A paraméteradaptációra általánosan az L_1 norma alapú regularizációt alkalmazva a (4.19) egyenlettel számolható:

$$\xi\left[k+1\right] = \xi\left[k\right] - \mu\frac{\partial E_s\left[k\right]}{\partial\xi\left[k\right]} + \mu\lambda sgn\left(\xi\right) \tag{4.19}$$

Lehetőség van az L_2 és L_1 regularizációs módszerek együttes alkalmazására (4.20):

$$L\left(\underline{W}\right) = E_s\left(\underline{W}\right) + \lambda_1\left|\underline{W}\right| + \lambda_2\left\|\underline{W}\right\|^{<2>} \tag{4.20}$$

4.4.3. Max-norma regularizáció

Az előbbi regularizációs változatokban a költségfüggvényt bővítettük ki egy büntető taggal. A max-norma regularizáció (4.21) nem bünteti a költségfüggvényt, hanem a $\|\underline{W}\|$ egy c sugarú körön belül tartja. A gradienssámolást, majd a súlytényező-adaptációt követően ki kell számolni a $\|\underline{W}\|$ súlyzók L2 normáját. Ha a számított norma meghaladja a c küszöbértéket, a súlytényezőket a c sugarú gömb felületére vetítjük a következő egyenletet alkalmazva [1].

$$\underline{W} = \frac{\underline{W}}{\|\underline{W}\|}c \tag{4.21}$$

4.4.4. Kiejtéses regularizáció

Kiejtéses (Dropout) regularizáció: a neuronok egy véletlenszerűen kiválasztott halmazát kiejtjük a tanulási körből.

5. fejezet

Többrétegű előrecsatolt neuronhálók

A fejezet célja a többrétegű előrecsatolt neuronhálók struktúrájának, a hiba visszaterjesztéses tanító algoritmusnak a bemutatása, valamint a háló bemeneti, kimeneti neuronjainak és a rejtett rétegben levő neuronok számának a meghatározása.

5.1. Többrétegű perceptron típusú neuronhálók szerkezete

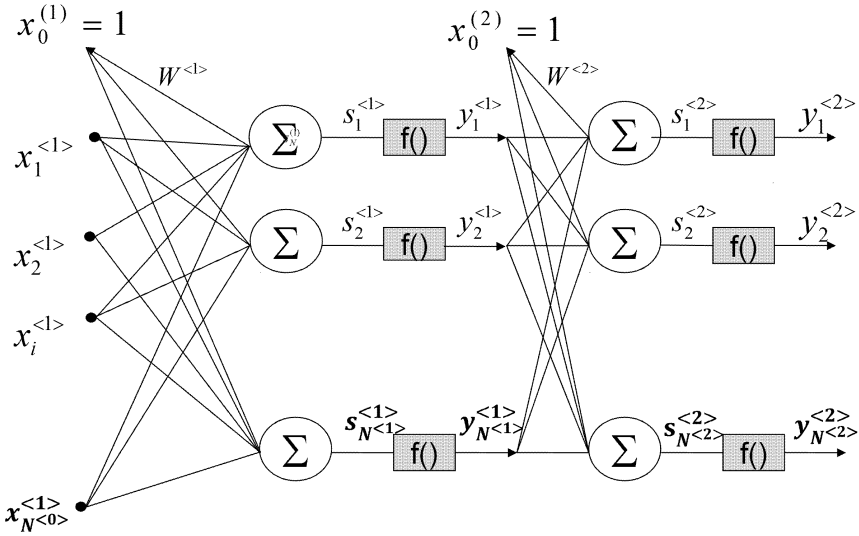
Topológiai szempontból az egy neuronhálót alkotó neuronokat rétegekbe rendezzük (5.1. ábra), amint a 3. fejezetben tárgyaltuk. A többrétegű neuronhálók esetében alapvetően három réteg különböztethető meg [2], [4], [6]:

- a bemeneti réteg,
- a kimeneti réteg,
- rejtett réteg.

A bemeneti réteghez tartozó neuronok bemenete képezi a hálózat bemeneteit. A bemeneti rétegnek jelentősebb szerepe nincsen a neuronháló működése szempontjából. A bemeneti réteg szétosztja a neuronháló bemeneteit a rejtett rétegben található neuronok bemenetére. Általában nem tartalmaznak hangolható paramétereket. A kimeneti réteg egyben a neuronháló kimenete is. A rejtett réteghez tartozó neuronok csak más rejtett réteg neuronjaival

vagy bemeneti neuronokkal, kimenetei pedig más rejtett réteg neuronjaival, illetve kimeneti réteghez tartozó neuronokkal vannak kapcsolatban. A rejtett rétegek számát, valamint a rejtett rétegen belüli neuronok számát a megoldandó feladat milyensége határozza meg.

A perceptron típusú neuron csak lineáris szeparációt képes megvalósítani [2], [10] [24]. A többrétegű neuronháló lehetővé teszi a lineárisan nem szeparálható minták szétválasztását, osztályozását [2], [25].



5.1. ábra. Többrétegű neuronháló szerkezete

Minden egyes réteghez egy sorszámot rendelünk a neuronháló bemenetétől a kimenet felé haladva. Minden egyes réteghez tartozó változót indexelünk a rétegnek a számával. A rétegnek a számát jelöljük l-lel, egy rétegben található neuronoknak a számát $N^{<l>}$-lel. Általánosan az l-dik réteghez tartozó változók:

$$\underline{X}^{<l>} = \left[x_0^{<l>}, x_1^{<l>}, x_2^{<l>} \cdots x_i^{<l>} \cdots x_{N^{<l-1>}}^{<l>}\right]^T \text{ – bemeneti vektor}$$

$$\underline{S}^{<l>} = [s_1^{<l>}, s_2^{<l>} \cdots s_i^{<l>} \cdots s_{N^{<l>}}^{<l>}]^T \text{ – ingervektor}$$

$$\underline{Y}^{<l>} = [y_1^{<l>}, y_2^{<l>} \cdots y_i^{<l>} \cdots y_{N^l}^{<l>}]^T \text{ – kimeneti vektor}$$

$\underline{\delta}^{<l>} = [\delta_0^{<l>}, \delta_1^{<l>}, \delta_2^{<l>} \cdots \delta_i^{<l>} \cdots \delta_{N^{<l>}}^{<l>}]^T$ – réteg kimenetén a hiba

$\underline{W}^{<l>} =$

$$\begin{bmatrix} -\theta_1^{<l>} & -\theta_2^{<l>} & \cdots & -\theta_j^{<l>} & \cdots & -\theta_{N^{<l>}}^{<l>} \\ w_{1,1}^{<l>} & w_{2,1}^{<l>} & \cdots & w_{j,1}^{<l>} & \cdots & w_{N^{<l>},1}^{<l>} \\ w_{1,2}^{<l>} & w_{2,2}^{<l>} & \cdots & w_{j,2}^{<l>} & \cdots & w_{N^{<l>},2}^{<l>} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{1,N^{<l-1>}}^{<l>} & w_{2,N^{<l-1>}}^{<l>} & \cdots & w_{j,N^{<l-1>}}^{<l>} & \cdots & w_{N^{<l>},N^{<l-1>}}^{<l>} \end{bmatrix}$$

– súlymátrix

Két réteget tartalmazó neuronháló esetében a következőképpen alakulnak a jelölések:

$\underline{X}^{<1>} = x_0^{<1>}, x_1^{<1>}, x_2^{<1>} \cdots x_i^{<1>} \cdots x_{N^{<0>}}^{<1>}]^T$ – bemeneti vektor

$\underline{S}^{<1>} = [s_1^{<1>}, s_2^{<1>} \cdots s_i^{<1>} \cdots s_{N^{<1>}}^{<1>}]^T$ – ingervektor

$\underline{Y}^{<1>} = [y_1^{<1>}, y_2^{<1>} \cdots y_i^{<1>} \cdots y_{N^{<1>}}^{<1>}]^T$ – kimeneti vektor

$\underline{\delta}^{<l>} = [\delta_0^{<1>}, \delta_1^{<1>}, \delta_2^{<1>} \cdots \delta_i^{<1>} \cdots \delta_{N^{<1>}}^{<1>}]^T$ – réteg kimenetén a hiba

$\underline{W}^{<1>} =$

$$\begin{bmatrix} -\theta_1^{<1>} & -\theta_2^{<1>} & \cdots & -\theta_j^{<1>} & \cdots & -\theta_{N^{<1>}}^{<1>} \\ w_{1,1}^{<1>} & w_{2,1}^{<1>} & \cdots & w_{j,1}^{<1>} & \cdots & w_{N^{<1>},1}^{<1>} \\ w_{1,2}^{<1>} & w_{2,2}^{<1>} & \cdots & w_{j,2}^{<1>} & \cdots & w_{N^{<1>},2}^{<1>} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{1,N^{<0>}}^{<1>} & w_{2,N^{<0>}}^{<1>} & \cdots & w_{j,N^{<0>}}^{<1>} & \cdots & w_{N^{<1>},N^{<0>}}^{<1>} \end{bmatrix}$$

– súlymátrix

A második rétegre, jelen esetben a kimeneti rétegre a változók jelölése:

$\underline{X}^{<2>} = x_0^{<2>}, x_1^{<2>}, x_2^{<2>} \cdots x_i^{<2>} \cdots x_{N^{<1>}}^{<2>}]^T$ – bemeneti vektor

$\underline{S}^{<2>} = [s_1^{<2>}, s_2^{<2>} \cdots s_i^{<2>} \cdots s_{N^{<2>}}^{<2>}]^T$ – ingervektor

$\underline{Y}^{<2>} = [y_1^{<2>}, y_2^{<2>} \cdots y_i^{<2>} \cdots y_{N^{<2>}}^{<2>}]^T$ – kimeneti vektor

$\underline{\delta}^{<l>} = [\delta_0^{<2>}, \delta_1^{<2>}, \delta_2^{<2>} \cdots \delta_i^{<2>} \cdots \delta_{N^{<2>}}^{<2>}]^T$ – réteg kimenetén a hiba

$\underline{W}^{<2>} =$

$$\begin{bmatrix} -\theta_1^{<2>} & -\theta_2^{<2>} & \cdots & -\theta_j^{<2>} & \cdots & -\theta_{N^{<2>}}^{<2>} \\ w_{1,1}^{<2>} & w_{2,1}^{<2>} & \cdots & w_{j,1}^{<2>} & \cdots & w_{N^{<2>},1}^{<2>} \\ w_{1,2}^{<2>} & w_{2,2}^{<2>} & \cdots & w_{j,2}^{<2>} & \cdots & w_{N^{<2>},2}^{<2>} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{1,N^{<1>}}^{<2>} & w_{2,N^{<1>}}^{<2>} & \cdots & w_{j,N^{<1>}}^{<2>} & \cdots & w_{N^{<2>},N^{<1>}}^{<2>} \end{bmatrix}$$

– súlymátrix

5.2. MLP neuronháló tanítása

Többrétegű háló tanításának és alkalmazásának lépései:
1. A háló topológiájának meghatározása (rétegek száma, bemenetek száma, kimenetek száma, a rétegekben található neuronok száma)
– a tanító halmaz és tesztelő halmaz felépítése,
– a súlytényezők kezdeti értékeinek meghatározása, általában véletlenszerűen generáljuk.
2. A tanítás során a tanítóhalmazból minden egyes mintára ki kell számítani a neuronháló kimenetét, az elvárt kimenet alapján a hibát, és kell hangolni a súlytényezőket. A tanítóhalmazból kivesszük a következő mintát és bevezetjük a neuronháló bemenetére, $\underline{X}$, vagyis az első réteg bemenetére.

A neuronháló bemenetei: $\underline{X} = [x_0, x_1, x_2 \cdots x_i \cdots x_{N^{l-1}}]^T$ – bemeneti vektor .

Ha megfigyeljük a bemeneti vektort és a súlyzómátrixot, a bemeneti vektorban az első elem konstans egy, és kapcsolódik a súlymátrix első sorában megadott biasértékhez. A biasértéknek a tanítás során van jelentősége, és lényegében az aktivációs függvénynek egy küszöbértéket határoz meg. A biasnak az értéke is tanítható, hasonlóan a súlytényezőkéhez. Hogy ne kelljen külön megoldani a biastanítást, integráltuk a biasértékeket a súlymátrix első sorában. A tanító halmazból egy bemenetet, amelyet most $\underline{X}$-szel jelölünk, bevezetjük az első réteg bemenetére, vagyis $\underline{X}^{<1>} = \underline{X}$.

Kiszámoljuk az első rétegen az ingervektor értékeit (5.1):

$$\underline{S}^{<1>} = \left(\underline{W}^{<1>}\right)^T \underline{X}^{<1>} \tag{5.1}$$

Kiszámoljuk az első réteg kimenetét (5.2):

$$\underline{Y}^{<1>} = f\left(\underline{S}^{<1>}\right) \tag{5.2}$$

Az első réteg kimenete a második réteg bemenetét képezi. Kiszámoljuk a második rétegen az ingervektor értékeit (5.3):

$$\underline{S}^{<2>} = \left(\underline{W}^{<2>}\right)^T \underline{X}^{<2>} \tag{5.3}$$

ahol $\underline{X}^{<2>} = \begin{bmatrix} 1 \\ \underline{Y}^{<1>} \end{bmatrix}$ vagyis az első sorba behelyeztük a konstans bemenetet.

Kiszámoljuk a neuronháló kimenetét (5.4):

$$\underline{Y}^{<2>} = f\left(\underline{S}^{<2>}\right) \tag{5.4}$$

3. A neuronháló kimenetén a hibát az elvárt kimenet és a kiszámolt kimenet különbségeként határozzuk meg (5.5):

$$\delta_i^{<2>} = d_i - y_i^{<2>}, i = 1..N^{<2>} \tag{5.5}$$

Költségfüggvényként (5.6) a neuronhálók kimenetén az egyes kimeneten a hibák négyzetének az összege van alkalmazva:

$$E = \left(\delta_1^{<2>}\right)^2 + \left(\delta_2^{<2>}\right)^2 + \cdots + \left(\delta_{N^{<2>}}^{<2>}\right)^2 = \sum_{i=1}^{N^{<2>}} \left(\delta_i^{<2>}\right)^2, i = 1..N^{<2>} \tag{5.6}$$

4. A kimeneti réteg súlyzóinak módosítását a gradiens módszerből levezetve a (5.7) eredményezi:

$$w_{i,j}^{<2>}[k+1] = w_{i,j}^{<2>}[k] - \mu^{<2>} \frac{\partial E[k]}{\partial w_{i,j}^{<2>}[k]} \tag{5.7}$$

A következő (5.8) egyenletet kapjuk a kimeneti réteg súlytényezőinek a hangolására:

$$w_{i,j}^{<2>}[k+1] = w_{i,j}^{<2>}[k] + \mu^{<2>} x_j^{<2>}[k] f'\left(s_i^{<2>}\right) \delta_i^{<2>} \tag{5.8}$$

A kimeneti réteg súlytényezőinek a hangolása vektoriálisan az (5.9) formában írható fel:

$$\underline{W}^{<2>}[k+1] = \underline{W}^{<2>}[k] + \mu^{<2>} \underline{X}^{<2>}[k] \left(f'\left(\underline{S}^{<2>}\right) \odot \underline{\delta}^{<2>}\right)^T \tag{5.9}$$

5. A rejtett réteg súlytényezőinek módosítására a gradiens módszert a (5.10) szerint lehet felírni:

$$w_{i,j}^{<1>}[k+1] = w_{i,j}^{<1>}[k] - \mu^{<1>} \frac{\partial E[k]}{\partial w_{i,j}^{<1>}[k]} \tag{5.10}$$

A gradiens módszerből kiindulva az (5.11) egyenletet kapjuk az első (rejtett) réteg súlytényezőinek a hangolására:

$$w_{i,j}^{<1>}[k+1] = w_{i,j}^{<1>}[k] + \mu^{<1>} x_j^{<1>}[k] f'\left(s_i^{<1>}\right) \delta_i^{<1>} \tag{5.11}$$

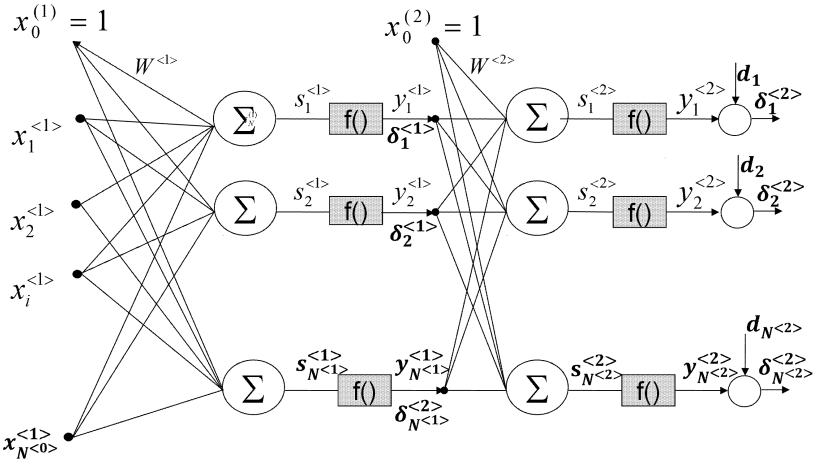
A rejtett réteg súlytényezőinek a hangolása vektoriálisan az (5.12) formában írható fel:

$$\underline{W}^{<1>}[k+1] = \underline{W}^{<1>}[k] + \mu^{<1>} \underline{X}^{<1>}[k] \left(f'\left(\underline{S}^{<1>}\right) \odot \underline{\delta}^{<1>}\right)^T \tag{5.12}$$

ahol a $\delta_i^{<1>}$ a visszaterjesztett hiba, $\odot$ pedig két vektor elemenkénti szorzata.

5.3. Hiba-visszaterjesztés

Ha összehasonlítjuk a kimeneti és rejtett rétegek súlytényezőinek hangolására alkalmazott egyenleteket, észrevesszük, hogy a lényeges különbséget a két képletben alkalmazott hibavektor jelenti. A kimeneti rétegen (a neuronháló kimenetén) egyértelmű a hibaszámítás $\delta_i^{<2>}$, viszont nem annyira a rejtett réteg kimenetén $\delta_i^{<1>}$ (5.2. ábra), ugyanis a rejtett réteg kimenetére nem határozható meg előírt érték [26]. A rejtett réteg kimenetén felírható a virtuális hiba, amelyet a kimeneti réteg hibájából származtatunk, a módszert hiba-visszaterjesztésnek nevezzük.



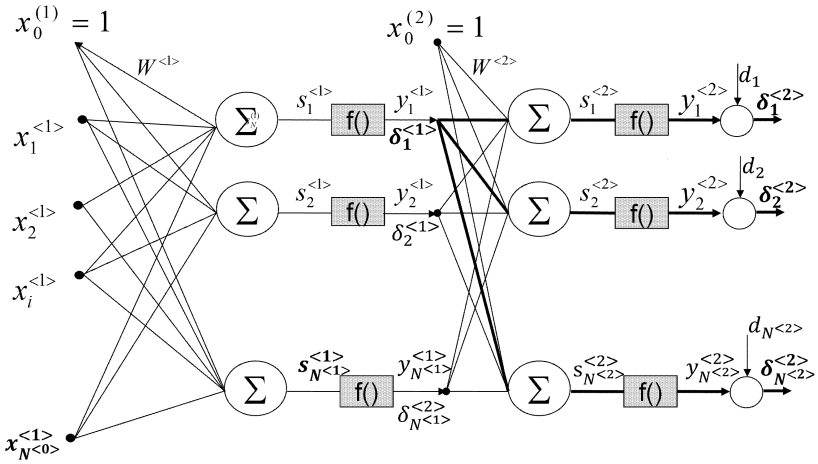
5.2. ábra. Hiba-visszaterjesztés, a kimeneti rétegen és a rejtett rétegen a δ szemléltetése

Egyenletrendszer formájában a visszaterjesztett hiba a következőképpen írható fel (5.13):

$$\begin{bmatrix} \delta_1^{<1>} \\ \delta_2^{<1>} \\ \vdots \\ \delta_{N^{<1>}}^{<1>} \end{bmatrix} =$$

$$= \begin{bmatrix} \delta_1^{<2>} f'\left(s_1^{<2>}\right) w_{1,1}^{<2>} + \delta_2^{<2>} f'\left(s_2^{<2>}\right) w_{2,1}^{<2>} + \cdots + \delta_{N^{<2>}}^{<2>} f'\left(s_{N^{<2>}}^{<2>}\right) w_{N^{<2>},1}^{<2>} \\ \delta_1^{<2>} f'\left(s_1^{<2>}\right) w_{1,2}^{<2>} + \delta_2^{<2>} f'\left(s_2^{<2>}\right) w_{2,2}^{<2>} + \cdots + \delta_{N^{<2>}}^{<2>} f'\left(s_{N^{<2>}}^{<2>}\right) w_{N^{<2>},2}^{<2>} \\ \vdots \\ \delta_1^{<2>} f'\left(s_1^{<2>}\right) w_{1,N^{<1>}}^{<2>} + \delta_2^{<2>} f'\left(s_2^{<2>}\right) w_{2,N^{<1>}}^{<2>} + \cdots + \delta_{N^{<2>}}^{<2>} f'\left(s_{N^{<2>}}^{<2>}\right) w_{N^{<2>},N^{<1>}}^{<2>} \end{bmatrix} \tag{5.13}$$

és a következőképpen értelmezhető: a rejtett réteg első kimenetére a visszaterjesztett hiba (5.3. ábra) egyenlő a kimeneti réteg első kimenetén a hiba $\delta_1^{<2>}$ szorozva az aktivációs függvény deriváltjával $f'\left(S_1^2\right)$, szorozva a rejtett réteg első kimenetét a második rétegben található első neuronnal összekapcsoló súlytényezővel $w_{1,1}^{<2>}$ és összegezve hasonló módon a hibát az összes kimenetről.



5.3. ábra. Hiba visszaterjesztése az első réteg első neuronjának a kimenetére

A kiterjesztett egyenletrendszer felírható a következő formában (5.14):

$$\delta_i^{<1>} = \sum_{j=1}^{N^{<2>}} \left(\delta_j^{<2>} f' \left(s_j^{<2>} \right) w_{i,j}^{<2>} \right) \tag{5.14}$$

és vektoriálisan is (5.15):

$$\underline{\delta}^{<1>} = \underline{W}^{<2>} \left(\underline{\delta}^{<2>} \odot f' \left(\underline{S}^{<2>} \right) \right) \tag{5.15}$$

ahol $\odot$ jelen esetben a két vektor $f' \left(\underline{S}^{<2>} \right)$ és $\underline{\delta}^{<2>}$ elemenkénti szorzatát jelenti (5.16).

$$\underline{\delta}^{<2>} \odot f' \left(\underline{S}^{<2>} \right) = \begin{pmatrix} \underline{\delta}_1^{<2>} f' \left(\underline{s}_1^{<2>} \right) \\ \underline{\delta}_2^{<2>} f' \left(\underline{s}_2^{<2>} \right) \\ \vdots \\ \underline{\delta}_{N^{<2>}}^{<2>} f' \left(\underline{s}_{N^{<2>}}^{<2>} \right) \end{pmatrix} \tag{5.16}$$

Összegezve egy többrétegű neuronháló tanítási lépéseit:

1. vesszük a tanító halmazból a mintát és bevezetjük a neuronháló bemenetére;
2. kiszámoljuk a rejtett rétegen az ingert;

3. kiszámoljuk a rejtett réteg kimenetét, amely a kimeneti réteg bemenetét képezi;
4. kiszámoljuk a kimeneti rétegben az ingert;
5. meghatározzuk a neuronháló kimenetét;
6. kiszámoljuk a hibát a tanítóhalmazból vett előírt érték, valamint a neuronháló kimenetének különbségeként;
7. tanítjuk a kimeneti réteg súlytényezőit;
8. visszaterjesztjük a hibát;
9. tanítjuk a rejtett réteg súlytényezőit.

A tanítási lépéseket (2–7) megismételjük minden egyes ki-bemeneti elempárra a tanító halmazból (tanítási ciklus). A háló tanítása több lépésből, tanítási korszakból (epoch) tevődik össze. Többször ismételjük meg a tanítást, amíg a tanítási hiba (pontosabban a neuronháló kimenetén a hibák négyzetének az összege) egy elfogadható érték alá csökken. Általában nem a hiba alapján állítjuk le a tanítást, mivel a program futtatása során végtelen ciklushoz vezethet, ha a hibát a tanítás során nem sikerül egy küszöbérték alá szorítani. Előre meghatározzuk a tanítási ciklusok számát. A tanítási ciklusok befejezése után ellenőrizzük a hibát. Sikeres végkimenetel esetében elmentjük a háló paramétereit, ellenkező esetben tovább tanítjuk. Általában a tanítás korszakokba van szervezve, és egy-egy korszak lefuttatása után lehetőség van a tanítási paraméterek újrahangolására.

5.4. MLP neuronháló – beépített Matlab függvényekkel

Alapvető Matlab függvények az előrecsatolt többrétegű hálók tanítására [27]:

- newff – egy új többrétegű előrecsatolt perceptron típusú háló létrehozása,
- sim – a háló kimenetének előállítása egy adott bemenetre (előhívási fázis),
- train – a háló tanítása.

NEWFF – létrehoz egy előrecsatolt neuronhálót

```
net = newff(PR,[S1 S2 ... SNl], {TF1 TF2 ... TFNl},
  BTF, BLF, PF)
```

A *newff* függvény paraméterezése:
- PR – R × 2 mátrix, tartalmazza a minimális és maximális értékeket egy R elemű bemeneti vektorra;
- Si – az i-edik réteg neuronjainak száma, az utolsó (kimeneti réteg) neuronjainak a számát a feladat határozza meg;
- TFi – i-edik réteghez tartozó aktivációs függvények típusa: tansig (tangens hiperbolikus), logsig (logisztikus), purelin (lineáris);
- BTF – a tanítási módszer;
- BLF – súlytényezők és bias hangolására alkalmazott tanítófüggvény;
- PF – költségfüggvény.

A tanító algoritmusként a következő módszerek közül lehet választani:
- *trainb* – súlytényező és eltolási érték kötegelt tanítása;
- *trainbfg* – kvázi-Newton módszeren alapuló hiba-visszaterjesztés;
- *trainbr* – Bayes alapú hiba-visszaterjesztés;
- *trainbu* – nem felügyelt súlytényező és eltolási érték kötegelt tanítása;
- *trainc* – ciklikus súlytényező és eltolási érték tanítás;
- *traincgb* – konjugált gradiens alapú hiba-visszaterjesztés;
- *traincgp* – konjugált gradiens alapú hiba Polak-Ribiere alapú paraméterfrissítés;
- *traingd* – Gradient descent alapú hiba-visszaterjesztés;
- *traingda* – Gradient descent hiba-visszaterjesztés adaptív *lr* tanítási együtthatóval;
- *traingdm* – Gradient descent alapú tanítás momentum taggal;
- *traingdx* – Gradient descent alapú tanítás momentum taggal és adaptív *lr* tanítási együtthatóval;
- *trainlm* – Levenberg-Marquardt alapú hiba-visszaterjesztés;
- *trainr* – véletlen sorrendű súlytényező/eltolás tanítás;
- *trainru* – nem felügyelt véletlen sorrendű súlytényező/eltolás tanítás;
- *trains* – szekvenciális sorrendű súlytényező/eltolás tanítás;
- *trainscg* – skálázott konjugált gradiens alapú hiba-visszaterjesztés.

Súlytényezők és eltolási érték (BLF) hangolására a következő tanító függvények alkalmazhatóak:
- *LEARNGD* – Gradient descent módszer alkalmazása súlytényezők/eltolási érték hangolására;

– *LEARNGDM* – súlytényezők/eltolási érték tanítása momentummal kibővített gradient descent módszerrel.

A tanítás során a következő költségfüggvények használhatók:
– *MSE* – átlagos négyzetes eltérés;
– *MSEREG* – átlagos négyzetes eltérés plusz regularizációs tag;
– *MAE* – átlagos abszolút eltérés.

A következő egyszerű példa szemlélteti a tanító halmaz, az elvárt kimenet létrehozását, a neuronháló felépítését. A neuronháló két aktív réteget tartalmaz, amelyekben tangens hiperbolikus, valamint lineáris aktivációs függvény használ. A *newff* első paramétere tartalmazza a bemenetekre a minimum és maximum értékeket. A második paraméter a rétegbeli neuronok száma: 5, illetve 1. A view net utasítással megjeleníthető a neuronháló felépítése. A *train* függvénnyel valósul meg a neuronháló tanítása a P bemeneti és T kimeneti elempárokra. A tanítást követően a P bemenetekre a *sim* függvénnyel valósul meg bemeneti értékekre a szimuláció.

```
clear all
P = [0 1 2 3 4 5 6 7 8 9 10]; %tanítóhalmaz bemenet
T = [0 1 2 3 4 3 2 1 2 3 4]; %tanítóhalmaz elvárt
    kimenet
net = newff([0 10],[5 1],{'tansig' 'purelin'});% a
    háló felépítése
view(net)
net = train(net,P,T); %a háló tanítása
Y = sim(net,P); %a háló kimenetének kiszámolása
```

Az újabb Matlab verziókban a *newff* helyett a feedforwardnet változatot alkalmazzák a neuronhálók létrehozására.

A következő egyszerű példa a *feedforwardnet* alkalmazását mutatja be:

```
clear all
[x,t] = tanito_halmaz(); %kell készíteni egy függvényt
    , amely felépíti a tanítóhalmazt
net = feedforwardnet(10); %felépíti a neuronhálót
net = train(net,x,t); %elvégzi a neuronháló tanítását,
     ha a súlyzók nem voltak inicializálva, a tanítás
    előtt inicializálja a súlyzómátrixokat (és egyéb
    paramétereket).
view(net)% egy ablakban megjeleníti felépített
    neuronháló szerkezetét
```

```
y = net(x); %kiszámolja a neuronháló kimenetét az x
    bemenetre
perf = perform(net,t,y) % kiszámolja a neuronháló
    teljesítményét (költségfüggvény értékét)
```

A példaprogramban a neuronháló a *feedforwardnet* függvénnyel van felépítve. A költségfüggvény értékének a kiszámítása a *perform* függvénnyel valósítható meg.

A neuronháló hangolására alkalmazható paraméterek: trainParam: .showWindow, .showCommandLine, .show, .epochs, .time, .goal, .min_grad, .max_fail, .mu, .mu_dec, .mu_inc, .mu_max

A fontosabb paraméterek értelmezése:
- epoch: tanítási ciklusok száma (ennyi ciklus után áll le a tanítás);
- goal: a kritérium függvény értéke az alkalmazott kritérium függvénynek megfelelően;
- mu: tanítási együttható;
- mu_inc: a tanítási együttható értékének a növelése, ha a hiba csökken;
- mu_dec: a tanítási együttható értékének a csökkentése, ha a hiba elkezd növekedni a tanítási ciklusok során.

A *net* osztályra alkalmazható metódusok:
- adapt: folytonos működés közbeni adaptáció;
- configure: bemenetek és kimenetek konfigurálása;
- gensim: Simulink modell létrehozása;
- init: súlytényezők és bias inicializálása;
- perform: költség kiszámítása;
- sim: meghatározott bemenetre a neuronháló kimenetének a kiszámítása;
- train: a neuronháló tanítása a megadott mintákra;
- view: kiértékelő diagramok megjelenítése;
- unconfigure: bemenetek és kimenetek leválasztása.

Miután létrehoztuk a neuronhálót a *newff* vagy *feedfordwardnet* függvényekkel, ha a parancs ablakba beírjuk a *net* változót, kiírja, milyen alapértelmezett beállításokkal van létrehozva a neuronháló. Az alapértelmezett paramétereken általában kell változtatni. A parancs ablakban megjelenített változók alapján áttekinthetők a paraméterek. Az 5.1. példa két paraméter,

a tanítási együttható és a tanítási lépések számának paramétermódosítását mutatja be:

5.1. példa. *Tanítási paraméterek beállítása*

```
net.trainParam.mu=0.01;
net.trainParam.epochs=2000;
```

5.5. Ötletek a tanítási algoritmus gyorsítására

Fontos, hogy minél optimálisabb, kisebb méretű és kevesebb rétegszámú is legyen, kevesebb neuronból álló neuronhálót kapjunk egy adott feladat megoldására, és, ha lehetséges, minimális lépésszámból végezzük el a neuronháló tanítását. A következő bekezdésekben egy pár ötletet foglalunk össze a tanítási együttható megválasztására, a lokális minimumokba való ragadás elkerülésére, a neuronháló méretének az optimalizálására.

5.5.1. Adaptív tanulási együttható

A tanítási együttható megválasztásánál érdemes a következő javaslatokat figyelembe venni:

1. Igen kicsi tanítási együttható esetében a tanulás lassú, és gyakran elakad lokális minimumokban.

2. Túl nagy tanítási együttható esetében az algoritmus nem konvergens.

3. Javasolt a változó értékű tanítási együtthatónak az alkalmazása, kezdetben nagyobb együttható biztosítja a gyors tanítást, a tanítás végén pedig egyre kisebb értékű együttható alkalmazása eredményezi a pontos, „finom” tanulást. Egy javaslat a változó értékű tanítási együttható tanítási ciklusonként való csökkentésére $\mu[k] = \mu_0 \frac{1}{1+\alpha k}$, ahol μ_0 a tanítási együttható kezdeti értéke, k a tanítás ciklusszáma, α egy paraméter, melynek segítségével a tanítási együttható változásának meredekségét lehet hangolni, $\mu[k]$ a tanítási együttható értéke a k-adik tanítási ciklusban.

4. Egy lehetséges módszer a tanítás gyorsítására a hiba alakulásának függvényében módosítani a tanítási együtthatót:

– ha a tanulás stagnál, duplázzuk meg a tanítási együtthatót;
– ha a tanulás elég gyors, marad a tanítási együttható értéke az előbbi tanítási ciklusból;
– ha a hiba növekedni kezd, felezzük a tanítási együtthatót.

5.5.2. Lokális minimumokban való elakadás elkerülése

A lokális minimumokban való elakadás elkerülésére javasolt a momentum módszer alkalmazása (5.17). A Widrow Hoff (delta) szabályt egy újabb taggal, úgynevezett momentummal kell kiegészíteni. A momentumos tagnak a lényege, hogy figyelembe veszi a súlytényezők módosításának az irányát az előbbi lépésből. Amint az algoritmus nevéből is kiszűrhető, egy tehetetlenségi tagról van szó.

$$w_{i,j}^{<l>}[k+1] = w_{i,j}^{<l>}[k] + \mu^{<l>}\delta_i^{<l>} f'\left(s_i^{<l>}\right) x_j^{<l>}[k] + m\Delta w_{i,j}^{<l>}[k] \quad (5.17)$$

A hiba változását (5.18) alkalmazzuk az (5.17) egyenletben.

$$\Delta w_{i,j}^{<l>}[k] = w_{i,j}^{<l>}[k] - w_{i,j}^{<l>}[k-1] \quad (5.18)$$

ahol $w_{i,j}^{<l>}[k+1]$ az l-edik réteg i-edik neuron j-edik bemenetre kapcsolódó súlytényező értéke;
l-edik réteghez tartozó tanítási együttható;
$\delta_i^{<l>}$ a réteg kimenetén a hiba, a kimeneti réteg esetében a háló kimenetén számolt hiba, rejtett rétegek esetében pedig a visszaterjesztéses hiba;
$x_j^{<l>}[k]$ az l-edik réteg j-edik bemenete a k-edik tanítási ciklusban;
m momentum tag, 0 és 1 között vehet fel értéket.

5.5.3. A neuronháló méretének az optimalizálása

Nem elég a feladatot megoldani, hanem a leggyorsabb megoldást szeretnénk elérni. Ha a neuronháló szoftveres megvalósításáról van szó, a háló kimenetének a kiszámítása (és tanítása) annál gyorsabb, minél kevesebb számítást kell elvégeznünk. A háló méretének az optimalizálása az elvégzendő számítások csökkentését eredményezi.

5.5.3.1. A háló méretének növelésével

Egy kisebb méretű hálóból kiindulva végezzük el a tanítást. A tanítás végén ellenőrizzük, hogy elfogadható-e a megoldás. Ha nem elfogadható, újabb neuronokat építünk a hálóba. Egy rejtett réteget és egy kimeneti réteget tartalmazó háló esetében a rejtett rétegben újabb neuronnak a bevitele a súlymátrixoknak a következő módosításait igényli:

- a rejtett réteg súlytényező mátrixához hozzá kell adni egy újabb oszlopot,
- a kimeneti réteghez hozzá kell adni egy újabb sort.

Újra és újra elvégezzük a háló tanítását, amíg a hiba a megfelelő érték alá nem csökken. Az új neuron bevitele után a hiba csökkenése lesz észlelhető. Ha a hiba csökkenése stagnál, egy újabb neuron bevitelére kerül sor.

5.5.3.2. A háló méretének a csökkentésével

Egy nagyobb méretű hálóból kiindulva (amely biztosan megoldja a feladatot) optimalizáljuk a háló méretét, csökkentve a rejtett rétegben levő neuronok számát. A neuronok kivágásánál felmerül a kérdés, hogy mely neuronokat lehet kivágni a hálóból. A neuronhálóból kivághatók azok a neuronok, amelyek teljesítik a következő feltételeket:

- ha két neuronnak a kimenete hasonló a tanítóhalmaz minden elemére,
- a tanítóhalmaz minden elemére egyes neuronok kimenete nulla (vagy nullához közeli).

Egy neuron kivágása után továbbtanítjuk a neuronhálót. Első fázisban a hiba ugrásszerű növekedése lesz észlelhető, de a tanítás során, ha a háló képes megoldani a feladatot, a hiba az elvárt érték alá csökken. Tehát egy újabb neuront kell kivágni. Addig ismételjük a neuronok kivágását, amíg a hiba az elvárt érték alá nem csökken. A súlytényezőknek zérus érték körüli tartásához alkalmazhatóak a regularizációs eljárások.

A súlymódosításnál be lehet iktatni egy büntető, regularizációs tagot $\lambda w_{i,j}^{<l>}$, melynek hatására a tanítás után a súlytényezők értéke kicsi lesz (közel nullához). A büntetőtag alkalmazásával a súlymódosítás az (5.19) képlet szerint alakul:

$$w_{i,j}^{<l>}[k+1] = w_{i,j}^{<l>}[k] + \mu^{<l>}\delta_i^{<l>} f'\left(s_i^{<l>}\right) x_j^{<l>}[k] - \lambda w_{i,j}^{<l>}[k] \quad (5.19)$$

A büntetőtagot csak azon súlytényezőkre alkalmazzuk, melyek abszolút értéke egy küszöb alatt van $\left|w_{i,j}^{<l>}[k]\right|$, ellenkező esetben a háló nem fog tanulni. Az egyenletben a λ a felejtési együttható (regularizációs együttható). A büntetőtag alkalmazásának a célja, hogy a tanítás elvégzése után egyszerűsítsük a háló struktúráját, kiejtve a hálóból azokat a neuronokat, melyek kimenete a teljes tanítási ciklusra nulla körüli értéket vesz fel. A végső cél tehát az, hogy a tanítás elvégzése után kapjunk egy méretben is optimális neuronháló-topológiát, amely összehasonlítva a kezdeti neuronhálóval kevesebb számítást vesz igénybe.

5.6. MLP neuronháló működésének szemléltetése Matlab programmal

Tervezzünk egy többrétegű előrecsatolt MLP neuronhálót, 3×3 karakter felismerésére. (A karakterek egyénileg megválaszthatóak, az osztályozandó karakterek száma 4.)

1. Tanulmányozzuk a háló tanítását (tanulási sebességet, konvergenciát) különböző tanítási együtthatókra. A tanítási együtthatókat külön-külön változtatjuk a két rétegben.

2. Tanulmányozzuk a háló viselkedését a rejtett rétegben levő neuronok számának a függvényében.

3. Figyeljük meg, hogy mi a hátránya annak, ha a rejtett rétegben kevés neuront (1, 2) használunk?

4. Mennyi az optimális neuronok száma a rejtett rétegben?

5. Hasonlítsuk össze a háló viselkedését, ha az tartalmaz, illetve ha nem tartalmaz aktiváló függvényeket.

5.2. példa. *MLP neuronháló tanítása*

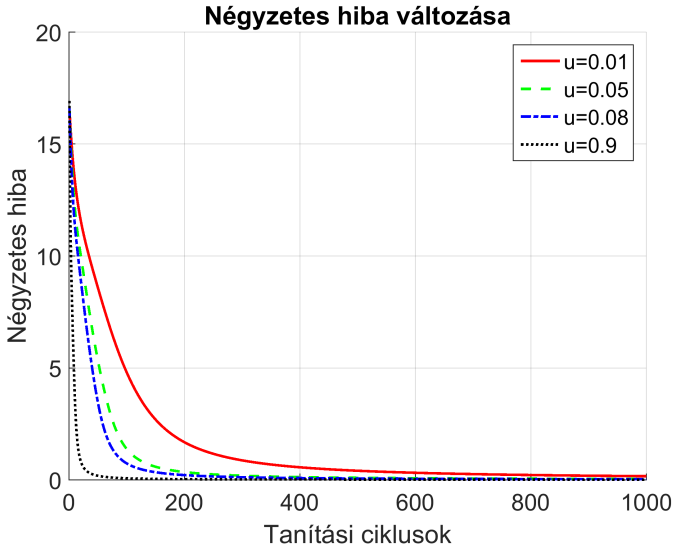
```
close all
clear all
tang_hip=@(s,a,teta) (1-exp(-a*s+teta))./(1+exp(-a*s+
    teta))
tang_hip_der=@(s,a,teta) 2*a*exp(-a*s+teta)./((1+exp(-
    a*s+teta)).^2)
```

```
% a tanító halmaz felépítése
%T -tartalmazza a bemeneteket
%d -tartalmazza a bemenetnek megfelelő elvárt
    kimenetet. Ebben az esetben
%mely karakterre mely kimenet kell aktiválodjon. A T
    mátrix minden egyes sorának megfelel egy sor a d
    mátrixból

T=[1 1 1 1 0 1 0 0 1 0;
    1 1 0 1 1 1 1 1 0 1;
    1 1 1 1 1 0 0 1 0 0;
    1 0 0 1 0 0 1 1 1 1];

d=[1 -1 -1 -1;
    -1 1 -1 -1;
    -1 -1 1 -1;
    -1 -1 -1 1];
N=10;      % -bemenetek_szama
M=10;      % -rejtett rétegben levő neuronok száma
P=4;       % -kimeneti réteg neuronainak száma
a=1        %- az aktivációs függvény meredekségét
    meghatározó paraméter
u2 =0.05; % tanítási együttható a kimeneti rétegre
u1 =0.05; % tanítási együttható a rejtett rétegre
Tanitasi_ciklusok_szama =1000;

w1_s=(rand(N,M) -0.5);
w2_s=(rand(M,P) -0.5);

figure(1)
hold on

u=[0.01 0.05 0.08 0.9];
szin={'r','g--','b-.','k:'}
%a külső ciklus különböző tanítási együtthatókra
    megismétli a tanítást
for l=1:4
w1=w1_s;
w2=w2_s;
```

```
%az alábbi ciklus meghatározza a tanítási ciklusok
    számát
for i=1:Tanitasi_ciklusok_szama
    E(i)=0;
%az alábbi ciklus szerepe a tanítás elvégzése minden
    egyes elempárra
    for j=1:4
%- a j-ik elem kiválasztása a tanítóhalmazból
x1=T(j,:)';
%az inger kiszámolása rejtett rétegre
s1=w1'*x1;
%a rejtett réteg kimenetének a kiszámolása
y1=tang_hip(s1,a,0);%
%a rejtett réteg kimenete a kimeneti réteg bemenete
x2=y1;
%az inger kiszámolása rejtett rétegre
s2=w2'*x2;
%a rejtett réteg kimenetének a kiszámolása
y2=tang_hip(s2,a,0); %
%a hiba kiszámolása a háló kimenetén
delta_2=d(j,:)'-y2;
%a kimeneti réteg súlyzóinak a tanítasa

w2=w2+u2*x2*(delta_2.*tang_hip_der(s2,a,0))';
u2=u(l);

%a hiba visszaáramoltatása a kimenetről a rejtett
    rétegre
delta_1=w2*(delta_2.*tang_hip_der(s2,a,0));
%a súlyzók tanítása a rejtett rétegre
        w1=w1+u1*x1*(delta_1.*tang_hip_der(s1,a,0))';
%a hiba összegzése egy tanítási ciklusra
E(i)=E(i)+delta_2'*delta_2; %   . A hibát egy
    vektorban tároljuk, a vektor minden egyes eleme
    egyes tanítási ciklusokban kapott hibákat
    tartalmazza. Grafikusan ábrázolva a hibát,
    következtetni lehet a háló tanulásának alakulására.
    end
end
```

```
plot(E,szin{l},'Linewidth',2);
leg_text{l}=strcat('u=',num2str(u2));
end;

grid on;
title('Négyzetes hiba változása')
xlabel('Tanítási ciklusok')
ylabel('Négyzetes hiba')
legend(leg_text);
set(gca,'FontSize',18)
print('-dpng','-r300','hiba_alakulasa_MLP');
```

Az 5.2. példa saját Matlab programmal megvalósított MLP neuronháló megvalósítását és tanítását mutatja be. A bemenetet négy darab 3×3 méretű rácson meghatározott forma képezi. A programrész négy különböző tanítási együtthatóra fut le.



5.4. ábra. Négyzetes hiba alakulása különböző tanítási együtthatókra

A négyzetes hiba alakulását az 5.4. ábra szemlélteti. Kisebb tanítási együtthatóra a neuronháló lassabban konvergál, a négyzetes hiba kisebb mértékben csökken.

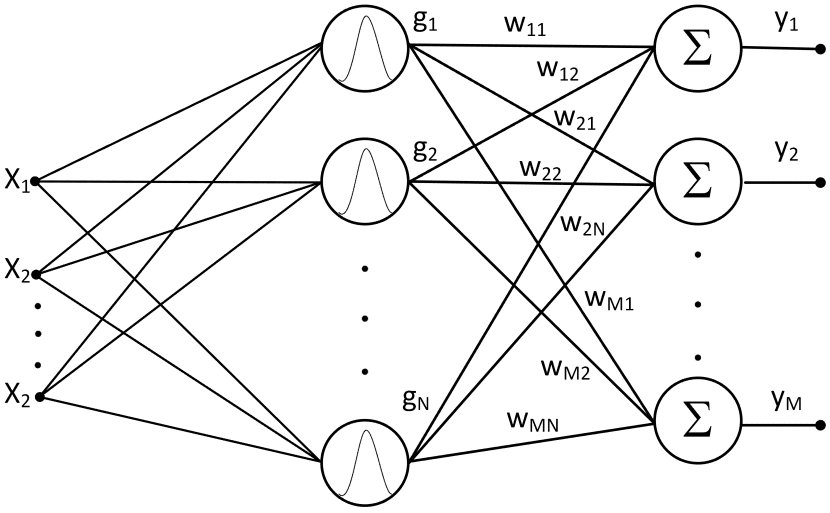
6. fejezet

Radiális bázisfüggvényekből álló hálózat

A fejezet bemutatja a radiális bázisfüggvényekből álló hálózat (RBF) struktúráját, a bázisfüggvények elhelyezésének módjait, a bázisfüggvények középpontjának meghatározását, a pillanatnyi és a globális hiba szerinti tanulás összehasonlítását.

6.1. Bevezető

Bázisfüggvényeken alapuló hálózatokat Broomhead és Love vezettek be 1988-ban [28], majd Moody, Darken 1989-ben [29]. Az RBF típusú neurális hálóknál a nemlineáris rész átkerül az összegző elem elé, ezek lesznek a bázisfüggvények. Minden bázisfüggvényhez és kimenethez tartozik egy súlytényező. Az információ a súlytényezőkben van tárolva. A kimenet egy lineáris összegzőt tartalmaz. A hálónak egy tanítható rétege van, ez az oka annak, hogy a kimenetek számítása és a tanítási folyamat gyorsabb, mint többrétegű hálók esetén.



6.1. ábra. RBF neuronháló szerkezete

6.2. RBF hálózat szerkezete

Az RBF hálónak két aktív rétege van (6.1. ábra), a bemeneti réteg, mely egy nemlineáris leképezést végez, és a kimeneti réteg, amely a bázisfüggvények és a súlytényezők lineáris kombinációját valósítja meg. A súlytényezők mátrixformába foglalhatók, amely sorainak száma megegyezik a bázisfüggvények számával és oszlopainak száma a kimenetek számával. Így minden bázisfüggvény-kimenet pároshoz tartozik egy súlytényező [30], [4]. A megtanult információ ezekben a súlytényezőkben van tárolva. A háló bemeneteinek száma adja meg a bázisfüggvények terének dimenzióját. A háló kimenetét az alábbi (6.1), (6.2) képletek alapján lehet kiszámolni:

$$y_i = \sum_{k=0}^{N} g\left(\underline{X}, \underline{c}_k, \sigma_k\right) w_{i,k} \tag{6.1}$$

$$y_i = \prod_{k=0}^{N} g\left(\underline{X}, \underline{c}_k, \sigma_k\right) w_{i,k} \tag{6.2}$$

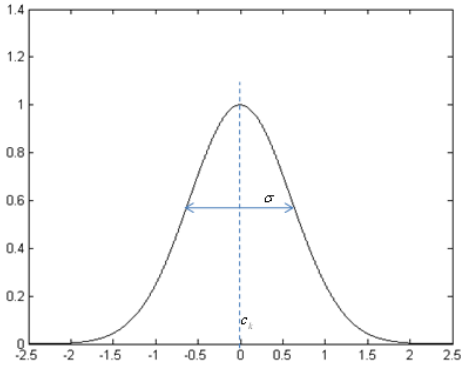
ahol: $g\left(\underline{X}, \underline{c}_k, \sigma_k\right)$ – a bázisfüggvény kimenete $w_{i,k}$ súlytényező k-adik bázisfüggvényről az i-edik kimenetre kapcsolódó súlyozó,
i – a kimenet száma,

k – a bázisfüggvény száma,
$\underline{c}_k$ – k-dik bázisfüggvény középpont vektora,
σ_k – k-dik bázisfüggvény szórása.

Összegzés esetében a háló érzékenyebb, viszont a kimenete nem normalizált érték. Szorzat használata azért előnyös, mert a háló kimenete a [0, 1] intervallumban van normalizálva. Ebben az esetben egyik bázisfüggvény kimenete sem lehet nulla.

6.2.1. RBF neuronhálókban használt bázisfüggvények

Bázisfüggvényként alkalmazható minden körszimmetrikus nemlineáris függvény [4]. Az RBF hálózatokban elterjedt körszimmetrikus függvény például a multikvadratikus vagy az inverz multikvadratikus függvény, azonban legtöbbször a Gauss-függvényeket (6.3) alkalmazzák (6.2. ábra). A hálózatban alkalmazott bázisfüggvények nagymértékben befolyásolhatják a tanulás minőségét.



6.2. ábra. Gauss-függvény

$$g\left(\underline{X}, \underline{c}_k, \sigma_k\right) = e^{\frac{-\|\underline{X}-\underline{c}_k\|^2}{\sigma^2}} \tag{6.3}$$

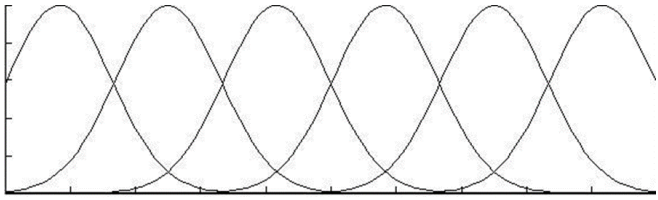
$\underline{X}$ – bemeneti vektor
$\underline{c}_k$ – a bázisfüggvény középpontja
σ_k – a bázisfüggvény szórása

$$\underline{W} = \begin{bmatrix} w_{1,1} & w_{2,1} & \cdots & w_{j,1} & \cdots & w_{M,1} \\ w_{1,2} & w_{2,2} & \cdots & w_{j,2} & \cdots & w_{M,2} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{1,N} & w_{2,N} & \cdots & w_{j,N} & \cdots & w_{M,N} \end{bmatrix} \text{ – súlymátrix}$$

N – bázisfüggvények száma

M – RBF neuronháló kimeneteinek a száma.

A szélességparaméterek megválasztására Gauss-függvények esetén az interpoláció nem érzékeny, ebből adódóan minden RBF neuronnak lehet ugyanaz a szélességparamétere. A középpontok is taníthatók, ezáltal a pontosság jelentősen növelhető, vagy ugyanolyan pontosság mellett csökkenthető a neuronok száma. A 6.3. ábra egy bemenetű hálóra szemlélteti a bázisfüggvények elhelyezését a bemeneti intervallumon.



6.3. ábra. Bázisfüggvények egyenletes elosztása egyváltozós bemenetre

Az RBF hálót függvényapproximációra alkalmazták, viszont klaszterezési feladatokban is hasznosnak bizonyul. Klaszterezési feladatokban hasonló módon kell alkalmazni az RBF neuronhálót, mint az előrecsatolt többrétegű neuronhálót. Függvényapproximációra való alkalmazáskor fontos, hogy a bázisfüggvények a teljes intervallumot lefedjék, a tér azon részén, ahol nem található bázisfüggvény, a háló nem képes az approximációra. A bázisfüggvények átfedésének mértéke is nagyban befolyásolja a háló approximációs minőségét. A tér azon részére, ahol a tanító halmazban hirtelen változások vannak, több bázisfüggvényt helyezhetünk. Ha a háló bemeneteinek száma n, akkor egy n dimenziós teret kell lefednünk, figyelve a fent említett feltételekre. A 6.4. ábrán például két dimenziós tér bázisfüggvényekkel való lefedése van szemléltetve.

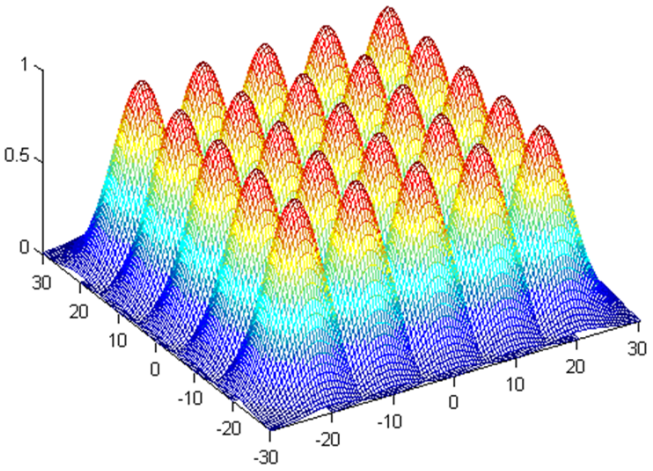
Más példák RBF neuronhálókban használt bázisfüggvényekre [2]:

$g(r) = r$ szakaszonkénti lineáris approximáció,

$g(r) = r^3$ köbös approximáció.

Multikvadratikus függvény: $g(x,c) = \left(x^2 + c^2\right)^{\frac{1}{2}} \quad c > 0, x \in R.$

Inverz multikvadratikus függvény: $g(x,c) = \frac{1}{\left(x^2+c^2\right)^{\frac{1}{2}}} \quad c > 0, x \in R.$



6.4. ábra. Példa egy kétdimenziós tér bázisfüggvényekkel való lefedésére

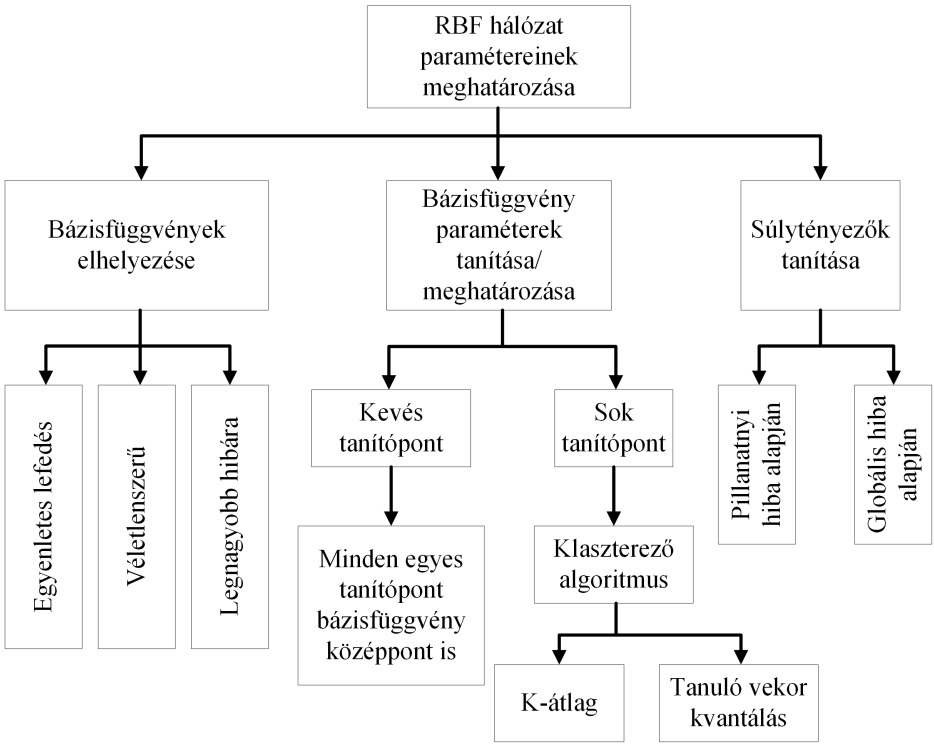
6.3. RBF neuronhálók tanítása

Az RBF hálók tanítóval való tanítást (felügyelt tanítást) igényelnek, tehát szükség van egy tanító halmazra. Ha szabályzásra használjuk a hálót, a tanító halmaz a referencia és kimenet párosból állítható össze. Mivel ezeknél a hálózatoknál a súlytényezőkön kívül a bázisfüggvények középpontjai és ezek szórása is tanítható, a tanítás több részből is összetevődhet. Az RBF hálózat tanítását a 6.5. ábra foglalja össze.

6.3.1. Bázisfüggvények elhelyezése

Érdemes a bázisfüggvények középpontjait olyan helyre tenni, ahol a tanító halmaz több pontot tartalmaz, vagy ahol a tanítandó felület változása nagy. A bázisfüggvények szórását úgy kell megválasztani, hogy az egész intervallumot lefedjék. A középpontok és a bázisfüggvények tanítása a súlytényezőkkel analóg módon is történhet, a hiba minimumát keresve. Egyszerűbb esetekben a bázisfüggvények egyenlő távolságra helyezkednek el egymástól, és a szórásuk úgy van megválasztva, hogy az egész teret lefedjék. Ebben az esetben csak a súlytényezők tanításával kell foglalkozzunk.

A bázisfüggvények elhelyezésére több megoldás is alkalmazható:



6.5. ábra. RBF hálózat paraméter hangolásának összefoglalása

1. A bemeneti tartomány bázisfüggvényekkel való egyenletes lefedése: meghatározzuk, hogy egy dimenzió szerint hány bázisfüggvényt szeretnénk elhelyezni, és ennek alapján határozzuk meg a bázisfüggvény középpontját és szélességparaméterét.

2. A bázisfüggvények középpont paraméterének a meghatározására klaszterező algoritmust alkalmazunk.

3. Egyenként elkezdünk elhelyezni egy-egy bázisfüggvényt. A bázisfüggvényt a legnagyobb hibapontban helyezzük el, elkezdjük tanítani a neuronhálót, amikor tanítási ciklusok során a hiba már nem csökken, egy újabb bázisfüggvényt helyezünk el szintén a legnagyobb hibának megfelelő bemeneti pontra. Addig ismételjük a bázisfüggvények elhelyezését, amíg a hiba egy előre meghatározott küszöbérték alá nem

esik. A bázisfüggvények elhelyezésére különböző kritériumokat lehet meghatározni: például egy metrika alapján, ha az elhelyezendő bázisfüggvény túl közel esik egy már elhelyezett bázisfüggvényhez, akkor az új bázisfüggvényt nem helyezzük el, vagy kivágunk egy már korábban elhelyezett bázisfüggvényt.

4. A bázisfüggvények elhelyezésére és a háló paramétereinek a hangolására genetikus algoritmusokra épülő módszereket alkalmazunk.

6.3.2. Bázisfüggvények paramétereinek meghatározása

A különböző rétegbeli processzáló elemek paramétereit eltérő módon választjuk, illetve határozzuk meg. A kimeneti réteg paraméterei, a wi súlyok, ellenőrzött tanítás során kapják meg a megfelelő értékeiket. Az aktivációs függvény paramétereinek (c_i középpont és esetenként a σ_i szélességparaméter) a megválasztására különböző eljárásokat alkalmazunk:

1. Kevés tanító pont esetén minden egyes tanító pont egyben függvényközéppontot is képviselhet. Ilyenkor az RBF processzáló elemek száma megegyezik a tanító pontok számával.

2. Nagyszámú tanító pont mellett minden tanító pont középpontként való felhasználása nem alkalmazható. A kellő pontosságú approximációhoz elegendő, ha a tanító pontoknak valamilyen összetartozó csoportjához egyetlen középpontot és így egyetlen processzáló elemet rendelünk. A tanító pontok csoportosítására, klaszterek képzésére bármilyen klaszterkialakító algoritmus használható:
   - tanulóvektor-kvantálás (LVQ-learning vector quantization);
   - K-átlagképző (K-mean) eljárás.

6.3.2.1. K átlagképző eljárás

A standard K-átlagképző eljárás (K-mean) klaszterkialakító algoritmus feltételezi, hogy az összes tanító pont a rendelkezésünkre áll. A K-mean algoritmus célja, hogy K olyan klaszter-középpontot (c_k, $k = 1, 2, \ldots, K$) határozzon meg, hogy a tanító pontok és a hozzájuk legközelebb eső klaszterközéppont négyzetes távolságainak összege minimális értéket vegyen fel. Ha U az u_i, $i = 1, 2, \ldots, P$ tanító vektorok halmazát, $U(k)$ pedig azon halmazt

jelöli, melynek pontjai a c_k $(k = 1, 2, \ldots, K)$ középpontokhoz lesznek a legközelebb, akkor a következő (6.4) kifejezés megoldását keressük [4]:

$$\min_{c_k} \sum_{k=1}^{K} \sum_{u_i \in U^{(k)}} \|u_i - c_k\|^2 \tag{6.4}$$

A K-mean algoritmus lépései:

1. Véletlenszerűen válasszunk meg egy K középpontot c_k, $k = 1, 2, \ldots, K$.

2. Határozzuk meg minden tanító pontra, hogy mely középponthoz van a legközelebb, vagyis alakítsuk ki a K klasztert, ha u_i-től a legkisebb távolságra lévő középpont c_k, akkor u_i a k-adik klaszter $U(k)$ eleme lesz.

3. Határozzuk meg az így kialakított klaszterek új középpontjait. Egy klaszter új középpontja legyen a klaszterekbe tartozó mintapontok átlaga.

4. Akkor fejezzük be az eljárást, ha a mintapontok klaszterbe sorolása már nem változik.

Sok esetben a c_k középpontokat egyszerűen a tanító pontokból véletlenszerűen kiválasztott K pont képezi. A középpont-paraméterek ellenőrzött tanítással is kialakíthatók [4]. Az RBF hálózatok nem érzékenyek a szélességparaméter megválasztására, a középpontok meghatározása után az alábbi heurisztikus, R legközelebbi szomszéd módszerrel határozható meg (6.5):

$$\sigma_j = \frac{1}{R} \sum_{k=1}^{R} \|c_k - c_j\| \tag{6.5}$$

ahol az R különböző c_j középpont, c_k R legközelebbi szomszédja. A szélességparaméter szintén meghatározható ellenőrzött tanítással is (6.6).

$$c_k = \frac{1}{P^{(k)}} \sum_{u_i \in U^{(k)}} u_i \tag{6.6}$$

ahol $P^{(k)}$ az $U^{(k)}$ klaszterbe tartozó tanító pontok száma.

A súlytényezők tanítása történhet a pillanatnyi hiba vagy a tanítási ciklus globális átlaghibája alapján:

– online tanítás – minden egyes be-kimeneti tanítópárra kiszámoljuk a hibát és tanítjuk a súlytényezőket;
– batch tanítás – kiszámoljuk a hibát minden be-kimeneti tanítópárra, de a súlymódosítást csak egyetlenegyszer végezzük el egy tanítási cikluson belül.

6.3.3. Súlytényezők hangolása

6.3.3.1. Pillanatnyi hiba alapján való tanítás

A pillanatnyi hiba alapján történő tanulásnál a súlymódosításra alkalmazzuk a Widroff–Hoff-(delta) szabályt (6.7):

$$w_{i,n}\left[k+1\right] = w_{i,n}\left[k\right] + \mu\left(\delta_i\right) g\left(\underline{X}, \underline{c}_k, \sigma_k\right) \;\; n = 1, \ldots, N, i = 1, \ldots, M. \tag{6.7}$$

A tanítási lépések:

1. Meghatározzuk a háló topológiáját, bázisfüggvények típusát, bázisfüggvények számát, a bázisfüggvények elhelyezésének módját, véletlenszerűen meghatározzuk a súlytényezők kezdőértékeit.

2. Kiszámoljuk a háló kimenetét (y) az x_k bemenet függvényében.

3. Ismerve a bemeneti-kimeneti tanító párokat (x_k, d_k), a második lépésben kiszámolt kimenetre kiszámoljuk a hibát (6.8):

$$\delta = d_k - y. \tag{6.8}$$

4. Módosítjuk a súlytényezőket a delta-szabállyal.

5. Megismételjük minden egyes ki-bemeneti párra a tanító halmazból.

6. Egy teljes ciklus végén kiszámolunk egy négyzetes hibát (6.9):

$$E = \sum_{i=1}^{M} \delta_i^2. \tag{6.9}$$

Ha a hiba egy adott szint alá csökken, vége a tanításnak, ha nem, újra vesszük a be-kimeneti párokat, és megismételjük az algoritmust a 2. ponttól.

6.3.3.2. Globális hiba alapján való tanítás

Kevés tanító pont esetén a kimeneti lineáris réteg tanítása egy lineáris egyenletrendszer megoldását jelenti, ahol az egyenletek és ismeretlenek száma megegyezik. Ha az u_i tanító pontokra adott kívánt válaszokat most d_i-vel jelöljük, akkor a megoldandó egyenlet (6.10):

$$\underline{G}\ \underline{W} = \underline{d} \tag{6.10}$$

ahol $G = [g_{ij}]$ az interpolációs mátrix és d a d_i kívánt válaszokból álló oszlopvektor. A kvadratikus mátrix elemei $g_{ij} = g_i(u_j)$. Ha G nemszinguláris, a megoldásvektor létezik (6.11) [4]: [28]:

$$\underline{W} = \underline{G}^{-1}\underline{d}. \tag{6.11}$$

Amennyiben a tanító pontok száma nagyobb, mint a rejtett rétegbeli processzáló elemek száma, a G mátrix nem lesz kvadratikus, a megoldandó egyenletrendszerben az egyenletek száma nagyobb, mint az ismeretlenek száma. A megoldást ilyenkor a legkisebb négyzetes eltérés értelmében kereshetjük (6.12):

$$C(\underline{W}) = \min_{\underline{W}} \|\underline{d} - \underline{GW}\| \tag{6.12}$$

függvényt felírva és elvégezve a w szerinti szélsőérték-keresést, a megoldás a következő (6.13):

$$\underline{W} = \left(\underline{G}^T\underline{G}\right)^{-1}\underline{G}^T\underline{d} \tag{6.13}$$

$\left(\underline{G}^T\underline{G}\right)$ kvadratikus, tehát amennyiben nem szinguláris, invertálható és így a megoldás létezik. Ha mégis a $\left(\underline{G}^T\underline{G}\right)$ szinguláris, akkor a következő egyenlettel helyettesítjük: $\left(\underline{G}^T\underline{G} + \lambda\underline{I}\right)$.

A λ megfelelő megválasztása biztosítja a nem szingularitást, $\underline{I}$ – egységmátrix.

Az RBF hálózat egyik változata a normalizált bázisfüggvények alkalmazása. A normalizálás azt jelenti, hogy bármely bemeneti vektorra az összes processzáló elem kimenetének összege konstans 1. A normalizált bázisfüggvények alkalmazását az approximációs tulajdonságok javulását mutató

eredmények indokolják.

$$g_i\left(\underline{u}, \underline{c}_i, \sigma\right) = \frac{e^{-\frac{\|\underline{u}-\underline{c}_i\|^2}{2\sigma^2}}}{\sum_{k=1}^{N}\left(e^{-\frac{\|\underline{u}-\underline{c}_k\|^2}{2\sigma^2}}\right)} \tag{6.14}$$

A normalizált bázisfüggvények (6.14) lényegében megegyeznek a softmax aktivációs függvénnyel.

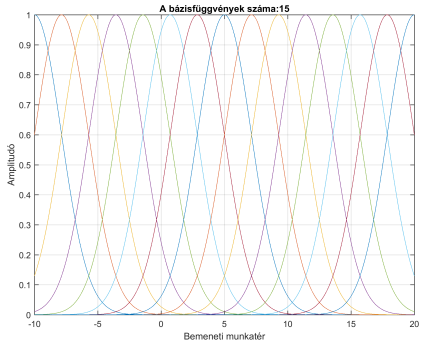
6.4. RBF neuronhálók alkalmazása

Az RBF neuronhálók fontosabb alkalmazási területei:
- többváltozós függvények approximációja,
- minták osztályba sorolása,
- adattömörítés,
- nemlineáris irányítási feladatok,
- rendszermodellezés,
- interpoláció.

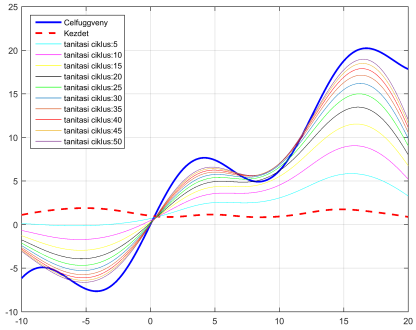
Az RBF neuronháló szerkezetéből adódóan, hardveres megvalósítás során magasfokú párhuzamosítást tesz lehetővé [31], [32], [33].

Az alábbi rajzokon egy RBF neuronhálót alkalmaztunk egy függvény megközelítésére. A tanító halmaz egy előre adott függvény alapján van generálva.

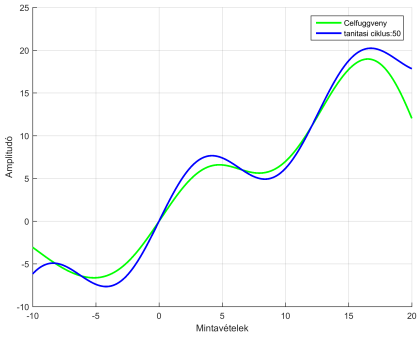
Egy kisebb $\mu = [0{,}0005]$ (6.8. ábra) és egy nagyobb $\mu = [0{,}01]$ (6.10. ábra) értékű tanítási együttható van alkalmazva. Az első esetben ($\mu = [0{,}0005]$) látszik a tanulási ciklusok során a tanulás eredménye, ahogy a neuronháló egyre jobban megközelíti a célfüggvényt (6.9. ábra).



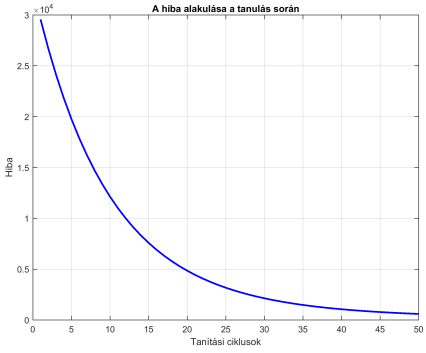
6.6. ábra. Bázisfüggvények elhelyezése



6.7. ábra. Tanítás alakulása a tanítási ciklusok során

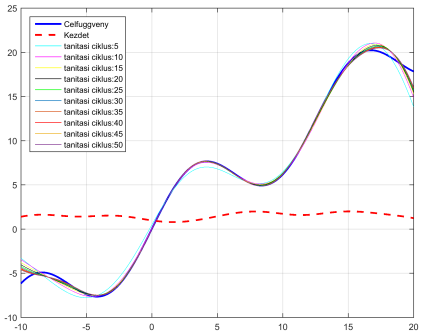


6.8. ábra. Függvény megközelítése RBF hálóval, $\mu = [0{,}0005]$

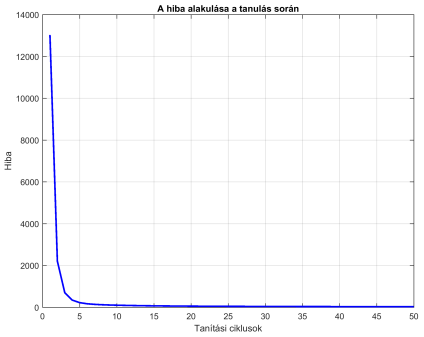


6.9. ábra. Tanítási hiba alakulása

A második esetben ($\mu = [0{,}0005]$) egy nagyobb értékű tanítási együtthatóra (6.10. ábra) már az első ciklust követően a neuronháló eredménye rásimul a célfüggvényre. A két mérés esetében a hiba alakulásából is jól érzékelhető a különbség (6.11. ábra). Utóbbi esetben a 8. tanítási ciklus körül már elfogadható az eredmény, a hiba közel zérus.



6.10. ábra. Függvény megközelítése RBF hálóval, $\mu = [0{,}01]$



6.11. ábra. Tanítási hiba alakulása

A 6.1. példaprogram egy radiális bázisfüggvényekből álló hálózat megvalósítását szemlélteti, amely alapján a szimulációs mérések is készültek. A bázisfüggvények egyenletesen vannak elosztva a bemeneti tartományon.

6.1. példa. *RBF hálózat*

```
clear all;
close all;
clc;
min_x=-10; max_x=20;
sigma=2.12; %a bázisfüggvény szélességparamétere
bsz=15;  %bázisfüggvények száma
d_x=0.1  %lépés a bemeneti tér mintavételezésére
mu =[0.0005];
a=4;
b=0.5;
szin = ['g','r','c','m','y','k','g--','r--','c--','m--
    ','y--','k--'];
be=[min_x:d_x:max_x];
ki = cel_fuggveny(be,a,b); %tanítóhalmaz előkészítése
n = length(be);
kozeppont = min_x : (max_x - min_x) / (bsz - 1) :
    max_x; %bázisfüggvények középpont paramétereinek
    meghatározása

%bázisfüggvények ábrázolása
figure(1)
```

```
for j=1:bsz
    b=(bazis(be , kozeppont(j) , sigma));
    plot(be,b); grid on;
    hold on;
end
 title(strcat('A bázisfüggvények száma:',num2str(bsz))
    );
  xlabel('x');
  ylabel('g(x)');

w = rand(bsz , 1);
  xlabel('Bemeneti munkatér');
  ylabel('Amplitudó');
%a tanulás időbeli alakulása
print -dpng -r300 bazisfuggvenyek_elhelyezese

figure(2);
plot(be,ki,'b','LineWidth',2);grid on;
mlegend{1}=strcat('Celfuggveny');

hold all;
for i = 1 : n,
    yy(i) = w' * (bazis(be(i) , kozeppont , sigma))';
end
plot(be,yy,'r--','LineWidth',2);
mlegend{2}=strcat('Kezdet');
legend(mlegend);
display('A továbblépéshez nyomd le az ENTER billentyüt
   ')
pause
l_index=2;
for t = 1 : 50, %tanítási ciklusok ismétlése
    for i = 1 : n, %tanitómináták indexelése
        y = w' * (bazis(be(i) , kozeppont , sigma))';
            %-a háló kiemenetének a kiszámolása
        hiba(i) = ki(i) - y;    % hiba számolás
        w = w + mu(1) * hiba(i) * (bazis(be(i) ,
            kozeppont , sigma))'; %sulyzótanítás
    end;
```

```
    negyzetes_hiba(t) = hiba * hiba'; %hibaösszegzés

   % Tanítás alakulása időben ábrázolva
    if (mod(t,5)==0) %minden ötödik tanítási ciklus
       után az eredmény ábrázolása
        figure(2);
        hold all;
        for i = 1 : n,
            yy(i) = w' * (bazis(be(i), kozeppont,
               sigma))';
        end
        l_index=l_index+1;
        plot(be,yy,szin(mod(l_index,14)));
        mlegend{l_index}=strcat('tanítási ciklus: ',
           num2str(t));
        pause(1)
    end;

end;
legend(mlegend,'Location','northwest');

print -dpng -r300 tanitas_kovetese


% a tanult függvény ábrázolása
figure(3);
  xlabel('Mintavételek');
  ylabel('Amplitudó');
hold all;
for i = 1 : n,
    yy(i) = w' * (bazis(be(i), kozeppont, sigma))';
end
plot(be,yy,szin(1),'LineWidth',2);


% a megközelitendő függveny ábrázolása
figure(3);
plot(be,ki,'b','LineWidth',2 );grid on;
hold off;
```

```
legend(mlegend{1},mlegend{l_index});

print -dpng -r300 tanitas_eredmenye
% a négyzetes hiba ábrázolása
figure(4);
plot(negyzetes_hiba,'b','LineWidth',2);grid on;
title('A hiba alakulása a tanulás során')
xlabel('Tanítási ciklusok')
ylabel('Hiba')
print -dpng -r300 tanitasi_hiba
```

7. fejezet

Nem ellenőrzött tanítású hálózatok

Jelen fejezet keretében szemléltetjük a nem felügyelt tanítású hálózatok struktúráját és tanítási algoritmusaikat. A nem felügyelt hálózatok közül a Kohonen-háló, valamint a háló tanítására alkalmazott Hebb-szabály és a versengő tanulás kerül bemutatásra. Említést teszünk a háló tanulásának alakulására alkalmazott Kohonen-térképről is. Az utazó ügynök típusú feladat Kohonen-hálóval való megvalósítását is bemutatjuk.

7.1. Bevezető

A nem felügyelt tanítású hálózatok esetében nem áll rendelkezésre a bemenetnek megfelelő elvárt kimenet. A tanításra használt tanító halmaz csak a háló bemeneteit tartalmazza. A háló feladata a bemeneti minták alapján osztályok létrehozása és a bemeneteknek az osztályokba sorolása.

A nem felügyelt tanítású hálózatok legfőbb alkalmazási területei:

- minták közötti hasonlóság megállapítása (formafelismerés);
- minták csoportosítása (klaszterezés);
- adattömörítés főkomponens-analízis módszerével (PCA Principle Component Analysis).

A főkomponens-analízis egy többváltozós statisztikai eljárás. Egy többdimenziós nagy adathalmaz dimenzióját csökkenti úgy, hogy a legnagyobb mértékben megtartja a jelben levő információ változatosságát.

Az önszerveződő térkép (SOM) típusú mesterséges neurális hálózatok nem felügyelt tanulással tanítanak. Az önszervező neuronháló a bemeneti többdimenziós teret kis dimenziós (tipikusan kétdimenziós) térre képezi le, amelyet térképnek neveznek. A háló dimenziócsökkentést tesz lehetővé. Az önszerveződő hálók eltérnek a többi mesterséges neurális hálózattól, mivel versengő tanulást alkalmaznak, szemben a hibajavító tanulással (mint például a gradiens módszerre épülő hiba-visszaáramoltatás). A bemeneti tér topológiai tulajdonságainak a megőrzésére szomszédsági függvényt használnak. Az önszervező neuronhálókat Teuvo Kohonen vezette be az 1980-as években, a nevéhez kapcsolódik a Kohonen-háló elnevezése [34], [35].

7.2. Nem ellenőrzött tanítási módszerek

7.2.1. Hebb-szabály alapú tanítás

A nem felügyelt tanítású hálózatok esetében az egyik leggyakrabban alkalmazott módszer a Hebb-szabályra épül (7.1). A Hebb-szabályon alapuló nem felügyelt tanító módszer egyszerűen a következőképpen foglalható össze: két processzáló elem közötti kapcsolat erőssége a processzáló elemek aktivitásának szorzatával arányosan változik [36], [28].

$$w_{i,j}\left[k+1\right] = w_{i,j}\left[k\right] + \mu y_i y_j \tag{7.1}$$

ahol $w_{i,j}\left[k\right]$ az i-dik és j-dik processzáló elem közötti súly y_i, y_j a két processzáló elem kimenetének az értékei, μ a tanítási együttható.

Ha a súly egy bemenet és egy processzáló elem között található, a súlymódosítás a (7.2) képlet szerint alakul:

$$w_{i,j}\left[k+1\right] = w_{i,j}\left[k\right] + \mu x_j y_i. \tag{7.2}$$

Egyes hálózatoknál az úgynevezett anti-Hebb (7.3) tanulási szabályt alkalmazzák:

$$w_{i,j}\left[k+1\right] = w_{i,j}\left[k\right] - \mu y_i y_j. \tag{7.3}$$

A súlyadaptációra a Hebb-szabály különböző normalizált változatait alkalmazzák (7.4), mivel az eredeti Hebb-szabály esetében a súlyok minden határon túlnövekedhetnek.

$$w_{i,j}\left[k+1\right] = w_{i,j}\left[k\right] + \mu\left(x_i - w_{i,j}\right) \tag{7.4}$$

7.2.2. A versengő tanulás

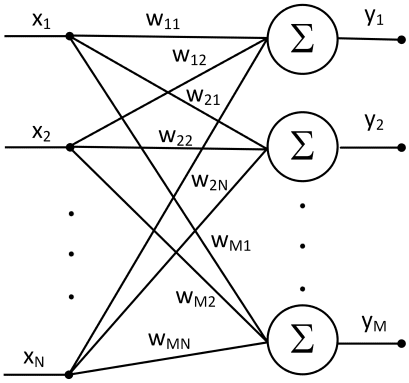
A versengő tanulás célja egy győztes neuron kiválasztása. A győztes neuron kimenete aktív lesz, míg az összes többi neuron passzív marad. A versengő tanulás célja a bemeneti mintatér szegmentálása, hogy egy adott tartományba tartozó bemenet hatására egy és csakis egy processzáló elem aktivizálódjon. A versengő tanulás egy tanító mintára a következő lépésekből áll:

1. A háló minden egyes processzáló elem kimenetének a meghatározása.
2. A győztes kiválasztása a győztes mindent visz elv alapján.
3. A győztes neuron kiválasztását követően csak a győztes neuron súlytényezőinek a tanítása. A súlymódosítást általában a Hebb-szabállyal végezzük.

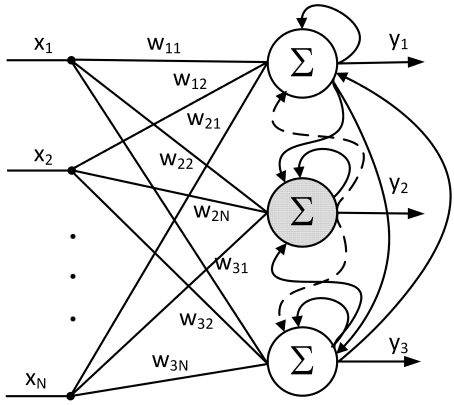
A győztes kiválasztása több módszerrel is történhet:
- Kiszámítjuk a kimenetet, és a legnagyobb kimenettel rendelkező neuron kimenetét egyesre, míg a többit 0-ra állítjuk.
- A neuronhálóban oldalirányú kapcsolatoknak az alkalmazása, amely lehetővé teszi a győztes automatikus kiválasztását. Az oldalirányú kapcsolatokat megvalósító súlyok megfelelő kiválasztásával megoldható, hogy a győztes kimenete egy legyen, és az összes többi kimenet pedig nullázódik. A közeli processzáló elemek gerjesztik, míg a távoli processzáló elemek gátolják egymást. A saját magára való visszacsatolásnak is gerjesztő hatása van, segíti, hogy a neuron győztes legyen.

7.2.3. A Kohonen-háló szerkezete

A nem ellenőrzött tanítású hálók többsége lineáris háló, tehát az alkalmazott processzáló elemek egyszerű súlyozott összegzést végeznek (7.1. ábra).

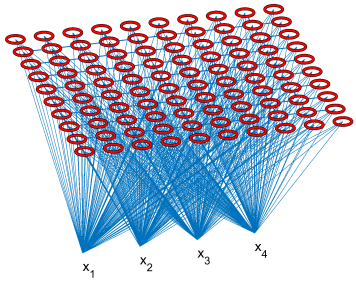


7.1. ábra. Kohonen-neuronháló szerkezete

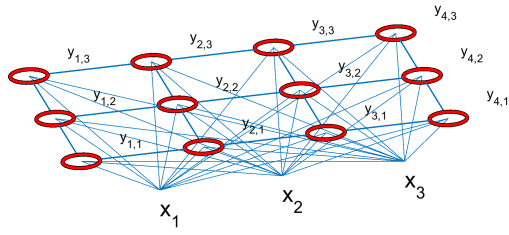


7.2. ábra. Kohonen-neuronháló szerkezete oldalirányú kapcsolatokkal

A háló alapeleme a lineáris összegző funkciót megvalósító processzáló elem. A processzáló elemek rácspontokban helyezkednek el. Minden bemenet a hálózat összes csomópontjához kapcsolódik. A hálózatban nemcsak előrecsatolást megvalósító súlytényezők léteznek, hanem amint említettük, közvetlen szomszédos csomópontokkal való oldalsó kapcsolatok is (7.2. ábra). A rács minden csomópontja egyben kimeneti csomópont is. Az oldalirányú kapcsolatokat megvalósító súlytényezők értéke rögzített. A Kohonen-háló nagymértékben párhuzamosítható, amint megfigyelhető a hardveres megvalósítás során [37].



7.3. ábra. Kohonen-neuronháló, neuronok térbeli elhelyezése



7.4. ábra. Kohonen-háló, neuronok térbeli elhelyezése

A Kohonen-háló fontosabb paraméterei:

– bemenetek száma (P);

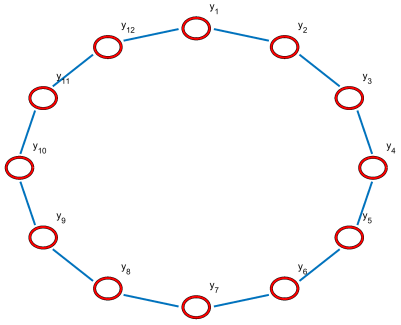
– az összneuronok (processzáló elemek, csomópontok) száma (N);
– súlymátrix ($W - N \times P$ méretű);
– a neuronok topológiai elhelyezkedése (alkalmazott topológia) $l = 1$, 1D elhelyezkedés, $l = 2$, 2D elhelyezkedés, $l = 3$, 3D elhelyezkedés;
– V mátrix a csomópontok pozíciójának az eloszlásáról ($V - m \times l$ méretű), $V = 12 \times 2$ mátrix megadja a topológiáját a csomópontoknak.

7.2.4. Alkalmazott topológiák

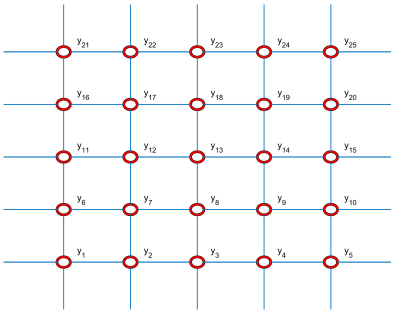
A rácspontok lehetnek egy egyenes mentén (egydimenziós), egy síkban (kétdimenziós), három- vagy többdimenziós térben elhelyezkedő rácspontok. Például egy kétdimenziós térben alkalmazhatunk négyzetes vagy háromszög csúcsain elhelyezkedő rácspontokat. Az alábbi rajzokon különböző módon elhelyezett rácspontok vannak szemléltetve:

7.5. ábra. Lineáris rácstopológia

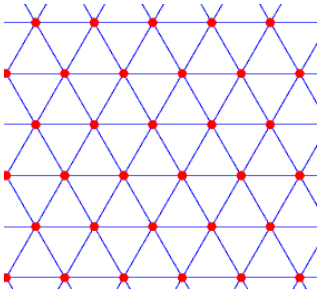
A Kohonen-hálók fontos eleme a Kohonen-térkép, amely egy hálóban található csomópontoknak a súlyvektortérben való ábrázolása. A súlyvektor térben minden neuront ábrázolunk egy pontként, ahol a pont koordinátáit az illető neuron súlyai határozzák meg. A szomszédos neuronok egymással össze vannak kötve az alkalmazott rácstopológia szerint.



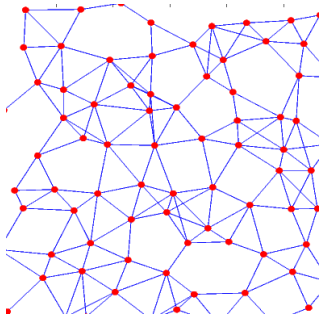
7.6. ábra. Gyűrűs topológia



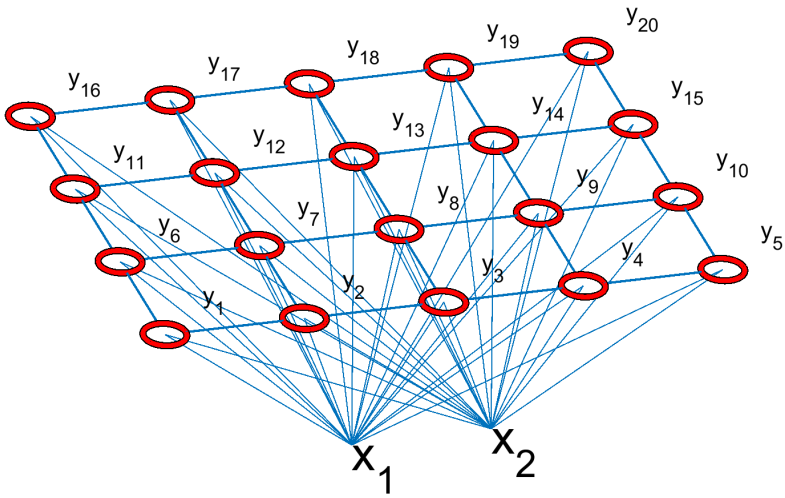
7.7. ábra. Négyzetes rácstopológia



7.8. ábra. Hexagonális elrendezésű topológia



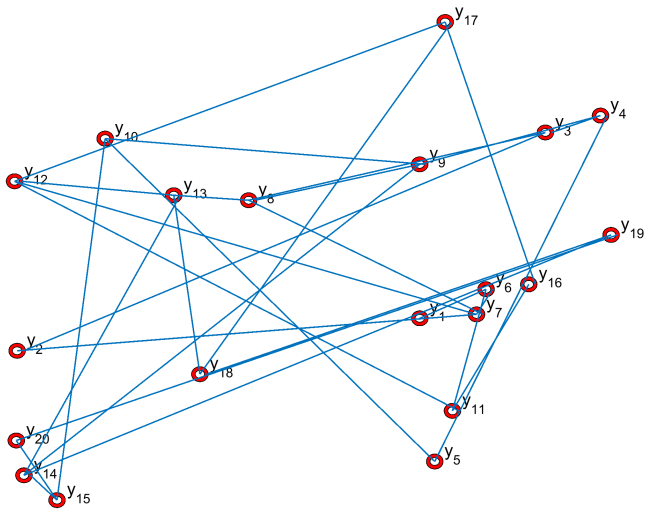
7.9. ábra. Véletlenszerűen generált csomópontok



7.10. ábra. Kohonen-háló négyzetes rácstopológia

A Kohonen-térképen az egyes pontok koordinátái a bemeneteknek az egyes neuronokhoz tartozó súlytényezői alapján vannak felrajzolva (7.11. ábra).

A példa szerint a neuronok egy négyzetrács-topológia szerint vannak elhelyezve (7.10. ábra). A KOHONEN térképen aszerint kell a csomópontokat összekapcsolni, hogy a rácstopológia szerint szomszédok-e a neuronok.



7.11. ábra. Kohonen-térkép

Ha a rácson két neuron szomszéd, akkor a Kohonen-térképen is meg kell jelölni (össze kell kötni) a két csomópontot. Például a 7-es neuron közvetlen szomszéd a 2, 6, 12 és 8-as sorszámú neuronokkal. Vagy például a 20-as neuron szomszéd a 15-ös és 19-es sorszámú neuronokkal. Egy csomópont mozgása a Kohonen-térképen, a tanulás során mutatja a súlytényezők változásának dinamikáját. Kezdetben, mivel a súlytényezők véletlenszerűen vannak inicializálva, a Kohonen-térkép egy véletlenszerű, rendezetlen háló. A súlytényezők folyamatos adaptálása során a csomópontok dinamikusan változnak, és ha sikeres a tanítás, a Kohonen-térképen megjelenített háló kifeszül a rácstopológiának megfelelően.

7.3. A Kohonen-háló tanítása

A tanulás után a Kohonen-térkép tükrözi a hálózat bemeneti adatainak az eloszlását.

A háló kimenetei a komponensek súlyozott összegei (7.5):

$$y_i = \sum_{j=1}^{N} w_{i,j} x_j, j = 1, \ldots, M. \tag{7.5}$$

Általános esetben az oldalirányú kapcsolatok rögzítettek: ha a processzáló elem önmagára való visszacsatolás, akkor gerjesztő, míg a többi oldalsó kapcsolat gátló.

A súlymódosítás a versengési elv felhasználásával a Hebb-szabályon alapul. A súlymódosítás a Hebb-szabály normalizált változata szerint történik. Csak a „győztes" processzáló elem súlyait módosítja. Az a „győztes" kimenet, amely a legnagyobb (vagy legkisebb) kimenő értéket adja:

$y_{i^*} \geq y_i$ ahol az i bármely neuron indexe, i^* a győztes neuron indexe.

A Hebb-szabály alapján a súlytényezők módosítása (7.6):

$$\underline{w}_{i^*} = \underline{w}_{i^*} + \mu y_i \underline{X}. \tag{7.6}$$

A győztes visz mindent (winner takes all) módszer szerint a győztes kimenet 1 lesz, a tanító algoritmus a (7.7) egyenlet szerint alakul:

$$\underline{w}_{i^*} = \underline{w}_{i^*} + \mu \underline{X}. \tag{7.7}$$

A módosítás során a súlyvektort azon bemeneti vektorok irányába toljuk, amely mellett az adott i* index a „győztes". A súlyvektorok – konvergencia esetén – olyan értékre állnak be, amely azon bemenő vektorok átlagaként, középpontjaként áll elő, melyek hatására az adott elem győztessé válik.

A korrekt versengéshez az kell, hogy ne a súlyok abszolút értéke, hanem az iránya határozza meg a győztest. Ezért a súlyokat normalizálni kell. Leggyakrabban az úgynevezett standard versengő tanulást alkalmazzák (7.8):

$$\underline{w}_{i^*} = \underline{w}_{i^*} + \mu \left(\underline{X} - \underline{w}_{i^*} \right), \tag{7.8}$$

de ebben az esetben a bemeneti vektorok is normalizált állapotban kell legyenek.

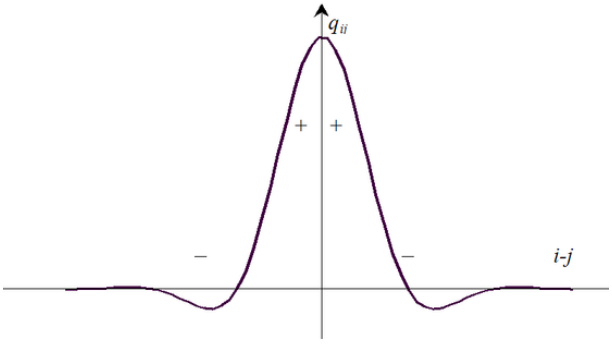
Ha csak a győztes neuron súlytényezőit hangoljuk, egyes neuronok súlytényezői sosem fognak tanulni, tehát ezen neuronok nem fognak részt venni a feladat megoldásában, viszont növelik a számításigényt, ugyanis a számításokat ezen neuronokra is el kell végezni. Javasolt, hogy ne csak a győztes neuron súlytényezőit, hanem a győztes környezetében levő neuronok súlytényezőit is tanítsuk.

Jobb minőségű tanítás érhető el, ha egy úgynevezett szomszédsági függvényt alkalmazunk. A szomszédsági függvény alkalmazásának a lényege, hogy a győztes neuronokhoz közelebb eső neuronok (közelebbi szomszédok) súlyait nagyobb tanítási együtthatóval, míg a távolabb eső neuronok súlyait kisebb tanítási együtthatóval tanítjuk. A szomszédsági függvény alkalmazásával a tanítási képlet a következőképpen alakul (7.9):

$$\underline{w}_i = \underline{w}_i + \mu \Phi \left(\underline{r}_i, \underline{r}_{i^*} \right) \left(\underline{X} - \underline{w}_{i^*} \right) \tag{7.9}$$

ahol r_i és r_j a neuronok topológiai helye, az r_i e neuronok egyike, míg r_j^* a győztes neuron topológiai helye. Az r_i vektornak annyi eleme van, ahány dimenziós a neuron topológiai elhelyezkedését meghatározó tér.

Szomszédsági függvényként alkalmazhatunk például Gauss [38] függvényt, vagy alkalmazhatjuk az úgynevezett mexikói kalapot (7.12. ábra).



7.12. ábra. Mexikói kalap

A mexikói kalap egyenlete (7.10):

$$\Phi(r) = \frac{2}{\sqrt{3\sigma} pi^{\frac{1}{4}}} \left(1 - \left(\frac{r}{\sigma} \right)^2 \right) e^{\frac{r^2}{2\sigma^2}}. \tag{7.10}$$

A súlyvektorok kezdeti értékeit megadhatjuk véletlenszerűen generált súlyeloszlásként, de akkor előfordulhat, hogy a tanulás nem lesz konvergens.

- A súlyvektorokat (kezdeti értéket) a tanító pontokból választjuk.
- Minden processzáló elem (a bemenetek számával egyenlő dimenziójú) súlyvektorait módosítjuk. Így a veszteseknek van esélye, hogy győztessé váljanak (szivárgó tanulás – ilyenkor kisebb μ-t alkalmazunk).

- A győztes környezetében levő elemeket is módosítjuk.
- A gyakran győzteseknél küszöbszintemelést alkalmazhatunk, míg a ritkán nyerteseknél csökkentjük a küszöbszintet, hogy ezáltal növeljük a győzelmi esélyt.
- A bemeneti vektorhoz zajt adunk, amelynek elég nagy a szórása, hogy mindenfajta bemenő érték előforduljon – újabb győztesek legyenek.
- A topológiai struktúrákat megőrző hálózat esetében nagyon fontos a szomszédsági függvény meghatározása.

A tanítás során kezdetben a súlytényezőket nagy lépéssel kell tanítani, és a szomszédsági függvénnyel a szomszédsági fokot is nagyra kell állítani. Ezáltal lehetővé tesszük a mintáknak egy adott halmazból egy másik halmazba való vándorlását. A tanítás során fokozatosan kell csökkenteni a tanítási együtthatót és a szomszédsági fokot. A tanítási ciklus végén a szomszédsági fokot le kell csökkenteni, hogy csak a győztes neuron súlytényezőit hangoljuk, ellehetetlenítve a mintáknak az osztályok közötti vándorlását. A legvégén finomhangolással (kicsi tanítási együtthatóval), a minták alapján, amelyek egy osztályba tartoznak, vagyis ugyanazt a neuront hozzák ki győztesnek, meghatározzák az osztály középpontját.

A Kohonen-háló tanulmányozására rendelkezésre áll egy Matlab környezetben elkészített példaprogram, vagy használni lehet a Matlab Neural Network Toolboxban megtalálható Kohonen-hálós függvényeket. A Matlab függvények közül a következőket javasolt átnézni: newsom, train, plotsom.

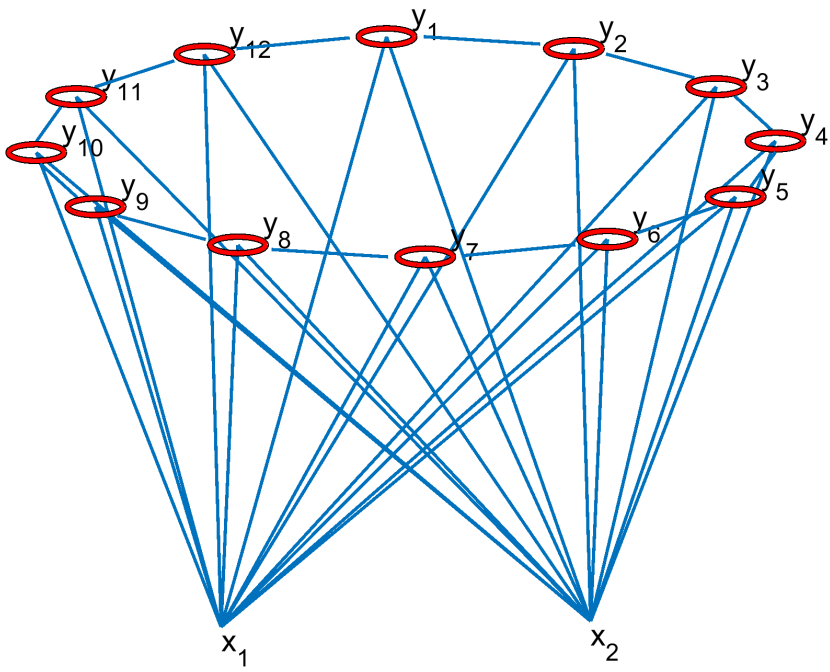
7.4. Travelling salesman probléma megoldása Kohonen-hálóval

A Kohonen-neuronháló-kimenet számításának a módja változhat a megoldandó feladat függvényében. Optimalizálási feladat esetében általában a kimenet számítását az optimalizálásra alkalmazható költségfüggvény határozza meg. Az utazó ügynök feladatot a következőképpen fogalmazhatjuk meg: adott N város, melyeknek ismerjük a koordinátáját. Elvileg bármelyik városból el lehet jutni az összes többi városba. A feladat az, hogy határozzuk meg a legrövidebb útvonalat, amelyen végig lehet járni a városokat. Minden egyes várost kötelező egyszer meglátogatni, de minden egyes város pontosan egyszer látogatható [4], [39], [40], [41].

Az utazó ügynök feladatnak a Kohonen-hálóval való megoldásához tekintsük át a fontosabb részfeladatokat és felmerülő kérdéseket, amelyek a feladat egyszerűbb áttekintéséhez vezetnek:

1. Milyen topológiát alkalmazzunk?

   Az utazó ügynök feladat megoldására a gyűrűs topológiát alkalmazzuk (7.13. ábra), ugyanis a megoldás szempontjából az utazó érkezik egy városból és továbbmegy egy célvárosba. A megoldásokat egy-egy neuron fogja jelenteni, tehát a neuronokat egy gyűrűs topológián kell elhelyezni. Ha a kezdőváros és célváros ismert, abban az esetben a gyűrűs topológia helyett egy lineáris topológiát alkalmazunk. Ugyanis ebben az esetben az utolsó meglátogatott városból nem kell visszajönni a kezdeti városba.



7.13. ábra. Neuronháló szerkezete utazó ügynök feladatra

2. Mi a neuronháló bemenete? A neuronháló bemenetei a meglátogatandó városnak a koordinátái vagy x_1, illetve x_2.

3. Miben fogjuk megkapni a megoldást? Minden egyes neuron egy megoldást szolgáltat, és a megoldás, vagyis a meglátogatandó városok koordinátái a neuronháló súlytényezőiben lesznek kódolva a tanítást követően.

4. Hogyan számoljuk ki a neuronháló kimenetét? A neuronháló kimenetét nem a hagyományos módszerrel, hanem lényegében a költségfüggvényt alkalmazva számoljuk ki. Ha logikusan átgondoljuk, ahhoz, hogy a legrövidebb utat tegye meg az ügynök, minden egyes városból a lehető legközelebbi városba kell ellátogatni, tehát a neuronháló kimenetének a számolása egy neuronra a (7.11) képlet szerint alakul

$$y_i = \sqrt{\sum_{j=1}^{N} (w_{i,j} - x_j)}. \tag{7.11}$$

5. Hogyan válasszuk ki a győztes neuront? Az lesz a győztes neuron, amelyre a pillanatnyi bemenet a legközelebb lesz egy neuron súlytényezőihez, vagyis a legkisebb kimenetet produkáló neuron.

6. A súlytényezők adaptálása a normalizált Hebb-szabállyal történik, alkalmazva a szomszédsági függvényt.

7. Szomszédsági függvényként Gauss-függvény (7.12) van alkalmazva:

$$\Phi\left(\underline{r}^*, \underline{r}_i, \sigma\right) = e^{-\frac{\|\underline{r}^* - \underline{r}_i\|^2}{2\sigma^2}} \tag{7.12}$$

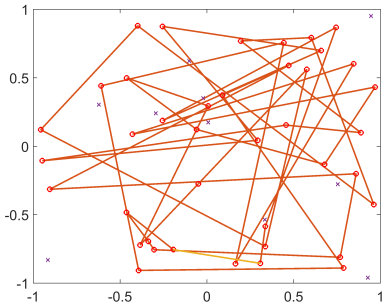
   ahol az i, j a neuronok indexe, amely között ki kell számolni a szomszédsági fokot, i a győztes neuron, j pedig az összes többi neuron egyike.

Az utazó ügynök feladatra a Kohonen-térképen a tanulás alakulása van szemléltetve több lépésben (7.14. ábra).

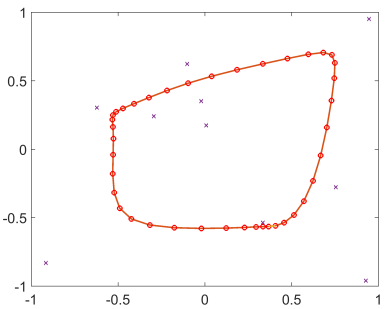
Az x-szel jelölt pontok a meglátogatandó városok koordinátái. A karikával jelölt pontok pedig a súlytényezőknek a súlyvektortérben való ábrázolása. A tanítás során a kezdeti súlytényezők véletlenszerűen vannak inicializálva. A kezdeti állapot (7.14. ábra) a tanítás előtti fázist jellemzi. Megfigyelhető, hogy a neuronok véletlenszerű inicializálásából adodóan a térkép teljes mértékben rendezetlen.

A 7.14. ábrákon a tanulási ciklusok során a Kohonen-térkép kezd kifeszülni a gyűrűs topológiának megfelelően. Az ábrákon ciklusonként az

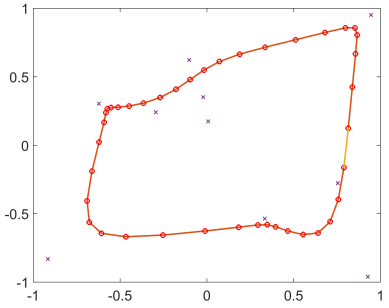
50-edik tanítási lépés eredménye van megjelenítve. A tanítás ciklusok során a térkép kifeszül a tanító pontokra, vagyis a városkoordinátákra, a neuronok elmozdulnak a városkoordináták irányába. A tanítás előrehaladtával a súlytényezők többsége felveszi a városkoordináták értékeit, a többi neuron pedig beáll a két várospontot összekötő egyenes szakaszra. Majd a tanítási ciklus előrehaladtával konvergálnak a városközéppontokra. A konvergencia sebessége és az eredmény alakulása a szomszédsági fok, illetve a tanítási együttható dinamikus hangolásával alakítható.



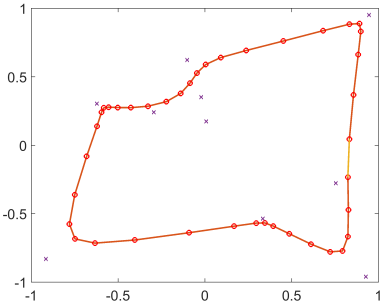
Kezdeti állapot



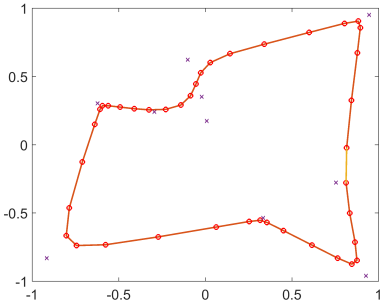
50. tanítási ciklus



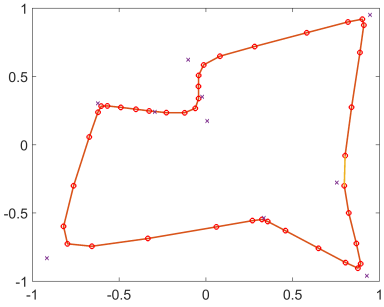
100. tanítási ciklus



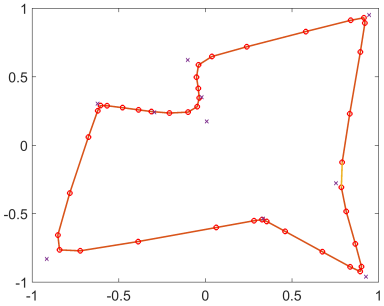
150. tanítási ciklus



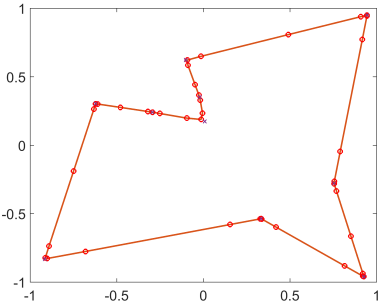
200. tanítási ciklus



250. tanítási ciklus



300. tanítási ciklus



350. tanítási ciklus

7.14. ábra. A Kohonen-térkép alakulása a tanítási ciklusok során

```
%Utazó ügynök feladat megoldása önrendező Kohonen
    hálóval
%A Kohonen rétegben levő neuronok koordinátáinak
    meghatározása
close all
clear all
%Gauss függvény deklaráció a szomszédsági fok
    számításához
tav=@(x,i,j,sigma) exp(-(norm(x(i,:)-x(j,:)).^2)/(2*
    sigma))
%                                    2
%                    ( - dist(i,j)   ) / ( 2 sigma)
%         r[i,j] = e
%tav = @(x,i,j,sigma) exp(-)
```

```
N=40 % -a neuronok száma
d_phi=2*pi/N;
phi=0;
R=1;
x=[]
meres=1
f_name='SOM_'
for i=1:N
    x=[x; R*sin(phi) R*cos(phi)];
    phi=phi+d_phi;
end

figure(1)
plot(x(:,1),x(:,2),'ro');
M=2% bemenetek száma
P =10 %városok száma
%X- városok koordinátái
figure(2)
X=2*(rand(P,2)-0.5); % városkoordináták inicializálása
plot(X(:,1),X(:,2),'x')
%súlyzómátrix
W=2*(rand(M,N)-0.5)

Tanitasi_ciklusok=1000
figure(3)
sigma0=3
for t=0:Tanitasi_ciklusok
u=0.8;
sigma=sigma0*1/(1+0.1*t) %szomszédsági fok
    ciklusonkénti csökkentése
plot(X(:,1),X(:,2),'x', 'Linewidth',2)
hold on
    if mod(t,50)==0
        f1=strcat(f_name,num2str(meres));
        plot(W(1,:),W(2,:),'Linewidth',2)
        plot(W(1,:),W(2,:),'ro','Linewidth',2)
        plot([W(1,1) W(1,N)],[W(2,1) W(2,N)],'
            Linewidth',2)
        plot(X(:,1),X(:,2),'x','Linewidth',2)
```

```
        set(gca,'FontSize',18)
        pause(0.001)
        print('-dpng','-r300',f1);
        hold off
        meres=meres+1;
    end

    for i=1:length(X)
        vx=X(i,1);
        vy=X(i,2);
      %Győztes neuron kiválasztása
      [min_ertek, gyoztes_index]=min(((W(1,:)-vx).^2)+
          ((W(2,:)-vy).^2))
      for j=1:length(W),
      %súlyzók tanítása
        W(1,j)=W(1,j)+u*tav(x,j,gyoztes_index,sigma)*(
            vx-W(1,j));
        W(2,j)=W(2,j)+u*tav(x,j,gyoztes_index,sigma)*(
            vy-W(2,j));
      end
    end
end
```

8. fejezet

Asszociatív és autoasszociatív hálók

Ebben a részben asszociatív és autoasszociatív neuronhálók kerülnek bemutatásra, többek között a Hopfield-háló. Szintén részletezve van az asszociatív hálók súlytényezőinek a meghatározása, amely Hebb-szabály szerint vagy egy előre meghatározott energiafüggvény szerint történik.

8.1. Asszociatív neuronhálók

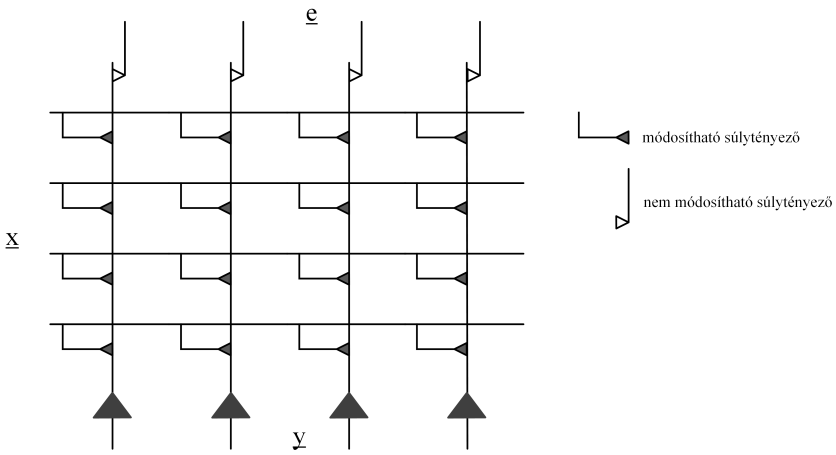
A neurális hálók egyik könnyen érthető művelete az úgynevezett asszociatív tanulás, ami azt feltételezi, hogy kapcsolatot teremt két különböző bemenő jel között úgy, hogy ha egy adott pillanatban csak az egyik van jelen, akkor a másik asszociatívan tanult jel is felidéződik. Az asszociatív háló struktúrája a 8.1. ábrán látható.

A cél az, hogy a kondicionált bemeneten levő jelet ($\underline{e}$) asszociáljuk a kondicionáló bemeneti jellel ($\underline{x}$) úgy, hogy a Hebb-szabály szerint módosítjuk a módosítható súlytényezőket, aminek az eredménye az, hogy ha a bemeneten csak a kondicionáló jelet alkalmazzuk, akkor a kimeneten visszanyerjük a tanuláskor a kondicionált (külső) bemeneten levő jelet.

A tanulás Hebb-szabály szerint történik:

$$w_{i,j} = w_{i,j} + \mu y_i x_j \tag{8.1}$$

ahol $\underline{e}$ – a kondicionált bemenet, w_{ij} – a súlytényező a j-edik bemenet és i-edik kimenet között, $\underline{x}$ – a kondicionáló bemenet, $\underline{y}$ – a kimenet, de tanítás során a kondicionált bemenetet alkalmazzuk.



8.1. ábra. Asszociatív neuronháló

Információ visszahívásához csak a kondicionáló ($\underline{x}$) bemenetet használjuk, az $s_i = \sum_{i=1}^{N} w_{i,j} x_j$ alapján megkapjuk minden neuronnak a megfelelő ingert, majd a neuronháló kimenetét mint $\underline{y} = f(\underline{s})$, ahol f a tanulás során is használt aktiváló küszöb függvény. Ekkor a kimeneten a tanuláskor a kondicionált bemeneti jelnek megfelelő és $\underline{x}$-szel kondicionáló $\underline{e}$-t kapjuk vissza.

Általános kérdések, amelyek felmerülnek:

- Mekkora az asszociációkapacitása egy ilyen neurális hálózatnak (NH) és mennyire stabil?
- Mekkora a hibatűrő képessége egy ilyen NH-nak (vagyis a zaj szerepe)?
- Mekkora az átfedés a kondicionáló jelek között, amikor egy adott kondicionált jelet akarunk visszahívni?
- Szükséges a teljes konnektivitás a bemenetek és neuronok között?

Példa:

1. lépés

Kezdetkor a módosítható súlytényezők zérus értékre vannak állítva. A kondicionáló bemenet $\underline{x} = (1, 0, 1, 0, 1, 0)$ míg a kondicionált bemenet $\underline{e} = (1, 1, 0, 0)$.

$$\underline{e} = (1 \quad 1 \quad 0 \quad 0)$$

$$\underline{x} = \begin{pmatrix}1\\0\\1\\0\\1\\0\end{pmatrix} \begin{pmatrix}0 & 0 & 0 & 0\\0 & 0 & 0 & 0\\0 & 0 & 0 & 0\\0 & 0 & 0 & 0\\0 & 0 & 0 & 0\\0 & 0 & 0 & 0\end{pmatrix}$$

2. lépés

Tanulást követően a módosítható súlytényezők az (8.1) egyenlet szerint a következőképpen alakulnak:

$$\underline{e} = (1 \quad 1 \quad 0 \quad 0)$$

$$\underline{x} = \begin{pmatrix}1\\0\\1\\0\\1\\0\end{pmatrix} \begin{pmatrix}1 & 1 & 0 & 0\\0 & 0 & 0 & 0\\1 & 1 & 0 & 0\\0 & 0 & 0 & 0\\1 & 1 & 0 & 0\\0 & 0 & 0 & 0\end{pmatrix}$$

3. lépés, lekérdezés

$$\underline{e} = (1 \quad 1 \quad 0 \quad 0)$$

$$x = \begin{pmatrix}1\\0\\1\\0\\1\\0\end{pmatrix} \begin{pmatrix}1 & 1 & 0 & 0\\0 & 0 & 0 & 0\\1 & 1 & 0 & 0\\0 & 0 & 0 & 0\\1 & 1 & 0 & 0\\0 & 0 & 0 & 0\end{pmatrix}$$

$$\underline{s} = (3 \quad 3 \quad 0 \quad 0)$$

$$f(\underline{s}) = (1 \quad 1 \quad 0 \quad 0)$$

A lekérdezési fázis során csak a kondicionáló bemenetre mindegyik neuron kimenetén megkapjuk az inger értékeket $\underline{s} = (3, 3, 0, 0)$, majd a küszöbfüggvényt alkalmazva kiszámoljuk a neuronok kimenetét. Eredményül visszakapjuk a kondicionált bemenetet.

4. lépés

Legyen a kondicionált jel: $\underline{e} = (0 \quad 1 \quad 0 \quad 1)$, és a kondicionáló $\underline{x} = (1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 1)$, és megtartjuk az előzőleg megtanult asszociáció generálta súlytényezőket, amihez hozzáadódnak az új asszociáció súlytényezői:

$$\underline{e} = (0 \quad 1 \quad 0 \quad 1)$$

$$x = \begin{pmatrix}1\\1\\0\\0\\0\\1\end{pmatrix} \begin{pmatrix}1 & 2 & 0 & 1\\0 & 1 & 0 & 1\\1 & 1 & 0 & 0\\0 & 0 & 0 & 0\\1 & 1 & 0 & 0\\0 & 1 & 0 & 1\end{pmatrix}$$

5. lépés

Használjuk most csak a második lépésbeli kondicionáló bemeneteket:

$$\underline{x} = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 1 & 2 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

$$\underline{s} = (1 \quad 4 \quad 0 \quad 3)$$

$$f(\underline{s}) = (0 \quad 1 \quad 0 \quad 1)$$

8.2. Autoasszociatív neurális hálók

Az autoasszociatív hálókat még attraktoroknak is nevezzük. Az autoasszociatív hálózatok lényege (8.2. ábra), hogy miután megtanítjuk a bemenő $\underline{e}$ jelet (vektort), akkor, ha ennek a vektornak egy részét visszük a bemenetre, akkor a kimenet egy adott pontossággal reprodukálja a megtanított $\underline{e}$ jelet.

Egy adott $[k]$ momentumban a hálózathoz rendeljük az $\underline{e}$ bemenetet, a nem módosítható súlytényezőkön keresztül. Így megkapjuk az $\underline{y}$ kimeneti jelet. Minden kimenő jel vissza van kapcsolva (backward) a dendritekre, tehát ténylegesen a kimenő jel a k momentumban képezi a $[k+1]$ momentumban a kondicionáló jelet, meghatározva a módosítható súlytényezők $w_{i,j}$ új értékeit ebben a lépésben. A tanulási folyamat $w_{i,j} = w_{i,j} + \mu y_i x_j$ Hebb-szabály szerint történik [24], [4]. Egy teljesen visszacsatolt rendszerben (egy rekurrens axon minden neuronnal kapcsolatban van), tanulás után a súlyozó mátrix szimmetrikus. Fontos, hogy tanulás alatt a külső $\underline{e}$ jelet minél jobban megközelítse a kimenő y, mert ha nem, az előzőleg megtanult bemenő jelek fogják dominálni a folyamatot, ami hamis asszociációhoz vezethet [42].

Előhíváskor, ha a bemenő $\underline{e}$ vektort vagy annak egy részét használjuk bemenetnek, a neuronháló vissza kell hívja a kimeneten a tanult jelet. A kimenet kiszámítása a következő lépések (8.2), (8.3) szerint történik:

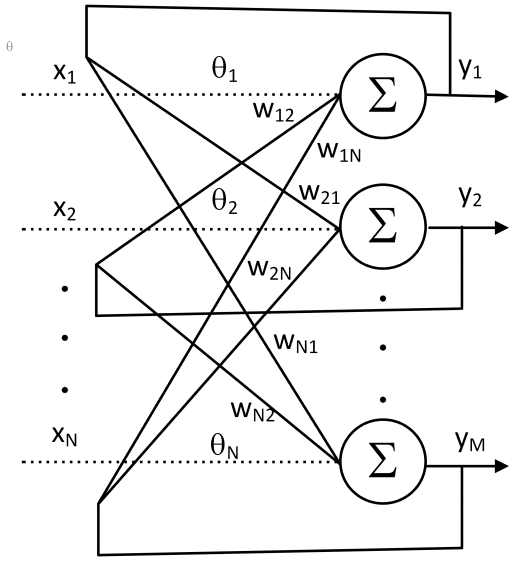
$$s_i = \sum_{i=1}^{N} w_{i,j} x_j \tag{8.2}$$

$$y_i = f(s_i + e_i) \tag{8.3}$$

s_i – az autoasszociatív neuronháló válasza, e_i – külső bemenet.

8.3. Hopfield-háló

Az 1982-ben publikált Hopfield-háló felépítése a 8.2. ábrán látható. A Hopfield-hálóra visszacsatolt perceptronként is hivatkozik a szakirodalom.



8.2. ábra. Asszociatív neuronháló

Bemenetek: $(x_1, x_2, \ldots, x_n)$ a neuronok kezdeti állapota.
Kimenetek: állapotok a működés során.
A Hopfield-háló neuronjai kétértékűek.

$$x_i = \begin{cases} -1, & \text{ha} \quad \textit{inaktív} \\ 1, & \text{ha} \quad \textit{aktív} \end{cases}$$

A súlytényezők dinamikája, ha

$$sign\,(x) = \begin{cases} -1, & \text{ha} \quad x < 0 \\ 1, & \text{ha} \quad x \geq 0 \end{cases}$$

$$x_i = sign\left(\sum_{j=1}^{N} w_{i,j} - \theta_i\right)$$

A neuronok egy állapota a konfiguráció, míg a neuronok összes állapota konfigurációs tér. Vonzási pontoknak nevezzük a konfigurációs térben a stabil állapotokat adó pontokat. A megfelelően beállított súlyokkal rendelkező hálózattól azt várjuk, hogy az adott kezdeti konfigurációból kiindulva az ehhez legközelebbi vonzási pontban stabilizálódjon. A hálózat működtetésének kétféle módja van:

– Szinkron ütemezés – minden időlépésnél minden neuron egyszerre változtatja a kimenetet.
– Aszinkron ütemezés – egy pillanatban csak egy egység válthat értéket.

A rendszer válasza a stabil állapotokhoz tartozó konfiguráció. A hálózat tanítása a megegyezendő minták tárolása (a súlyok megfelelő beállításával történik). Ezt a beállítást nevezzük tanításnak. A hálózat bemenetére mintát egyszer alkalmazunk. A tanulás módosított Hebb-szabállyal történik, ezzel határozzuk meg a súlyokat.

8.3.1. Egy minta tárolása

Legyen az $\underline{a}$ vektor a tanítandó minta. A tanítandó mintához tartozó konfigurációnak stabilnak kell lennie, és minél nagyobb vonzási körzetnek kell hozzá tartoznia. A minta stabil, ha:

$\forall\, i - re\ sign\left(\sum_{i=1}^{N} w_{i,j}a_j\right) = a_i\ a_i \in \{-1. + 1\}$

A súly megválasztása:

$w_{i,j} = a_i a_j$

$w_{i,j} = \frac{1}{N} a_i a_j$, ahol $\frac{1}{N}$ arányossági tényező.

Egy neuron akkor vált a kívánt értékre, ha a bemenetén keletkező összeg előjele megegyezik a minta megfelelő bitjének előjelével, vagyis:

$$s_i = \sum_{i=1}^{N} w_{i,j}a_j = \sum_{i=1}^{N} a_i a_j x_j = a_i \sum_{i=1}^{N} a_j x_j = s_i$$

8.3.2. Több minta tárolása

Több minta tárolása egy minta tárolásának szuperpozíciójával történik. Itt is igazolható, hogy az $a^{(p)}$ minták stabil állapotai lesznek a rendszernek.

$w_{i,j} = \sum_{i=1}^{p} a_i^{(l)} a_j^{(l)}$, ahol p a tanítandó mintáknak a száma.

A tárolt mintáktól valamilyen kis távolságban levő konfigurációból kiindulva a tárolt mintához tartozó állapotban stabilizálódik a rendszer, azaz képes hibás, zajos, hiányos minták helyreállítására, felismerésére. A minta vonzási körzete függ a tanított minták számától, egymáshoz viszonyított elhelyezkedésétől.

8.3.3. Az energiafüggvény

A rendszer minden konfigurációjához egy energiaértéket lehet rendelni, így a konfigurációs térben értelmezünk egy energiafelületet. Az energiafüggvény (Lyapunov-függvény) tulajdonsága, hogy a processzálás során az értéke csökken vagy állandó. Egy tetszőleges konfigurációból indított hálózat az energiafüggvény valamelyik lokális minimumpontjában fog stabilizálódni (ezek a vonzáspontok). Ha a súlytényezőket úgy választjuk, hogy az energiafelület minimumpontjai a tanulandó mintáknak feleljenek meg, akkor a háló vizsgálható az energiafüggvény alapján [2].

A hálózat energiafüggvénye a (8.4) egyenlet alapján számolható:

$$E[k] = \frac{1}{2} \sum_i \sum_j w_{i,j} x_i[k] x_j[k] - \sum_i \theta_i[k] x_i[k]. \tag{8.4}$$

A hálózat energiaváltozása (8.5):

$$\Delta E = E[k+1] - E[k]. \tag{8.5}$$

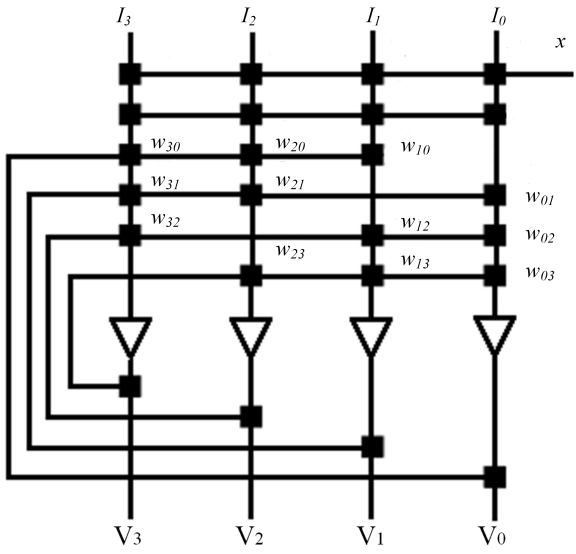
Az energiafüggvényből kiindulva úgy próbáljuk meghatározni az energiafüggvény minimumpontjait, hogy azok a tárolandó minták legyenek.

Algoritmus:

- Keresünk egy olyan függvényt, amely energiafüggvénye a rendszernek.
- Az energiafüggvény minimumpontjai a tanított mintákhoz tartozó konfigurációk.
- Ha sikerül ilyent találni, akkor a tagok szétválasztásával a hálózat súlyai meghatározhatóak. Tehát létrejött egy adott problémát megoldó hálózat.

8.3.4. Analóg-digitális átalakító megvalósítása Hopfield-típusú hálózattal

Egy Hopfield-hálózatot alkalmazva egy x értékű analóg bemeneti jelre szeretnénk meghatározni a bináris kimenetet (8.3. ábra). Az átalakító akkor működik helyesen, ha a kimeneten meghatározott bináris szó megegyezik a bemenő analóg értékkel [43], [44]. Az energiafüggvény a következőképpen írható fel, amely alapján azonosíthatók a bemeneti I_i, valamint a kimeneteket visszacsatoló súlytényezők $w_{i,j}$ értékei.



8.3. ábra. Analóg-digitális átalakítóhoz alkalmazott Hopfield-háló szerkezete

Az x bemeneti feszültségnek megfelelő kimeneti kód $V_3V_2V_1V_0$ közötti összefüggés a (8.6) egyenlet alapján írható fel:

$$x = \sum_{i=0}^{3} V_i 2^i. \tag{8.6}$$

Az energia változása a (8.7) képlet szerint modellezhető.

$$E = \frac{1}{2}\left(x - \sum_{i=0}^{3} V_i 2^i\right)^2 - \sum_{i=0}^{3}\left(2^i\right)^2 \left[V_i\left(V_i - 1\right)\right]. \tag{8.7}$$

A (8.7) egyenletet kiterjesztve, majd átrendezve az energia változására, a (8.8) egyenletet kapjuk

$$E = -\frac{1}{2}\sum_{i=0}^{3}\sum_{i\neq j=0}^{3} -2^{i+j} V_i V_j - \sum_{i=0}^{3}\left(-2^{2i-1} + 2^i x\right) V_i. \tag{8.8}$$

A (8.8) egyenletből azonosíthatók a $w_{i,j}$, illetve I_i paraméterek:

$$w_{i,j} = -2^{i+j}, I_i = -2^{2i-1} + 2^i x$$

9. fejezet

CMAC neuronháló

A fejezetben a CMAC (Cerebellar Model Articulation Controller) típusú neuronhálók struktúráját, paramétereit és tanítását ismertetjük. A fejezet végén egy egyváltozós függvény megközelítését oldjuk meg, CMAC típusú neuronhálót alkalmazva.

9.1. Bevezető

Az utóbbi években széles körű érdeklődés mutatkozott a rendszermodellezés és adaptív szabályozás iránt, melyeknek az alapját a biológiai struktúrák, rendszerek és különböző tanulási algoritmusok képezik. Intelligens szabályozási alkalmazásoknak a megvalósításához olyan algoritmusokra van szükség, amelyek biztosítják a következőket:

- Időben változó, hiányosan definiált területen is helyesen működjenek.
- Alkalmazkodjanak a rendszer dinamikájának a változásaihoz és a környezet kölcsönhatásaihoz.
- Az érdemi információkat stabilan sajátítsák el.
- Vegyék figyelembe a rendszer dinamikájára vonatkozó korlátozásokat.
- Kaotikus környezetben is képesek legyenek önállóan, minimális közbelépéssel működni.

Az emberi tanulási képesség magában foglalja azokat az elemeket, amelyek a felsorolt tulajdonságokat tartalmazzák. Vonzó cél olyan algoritmus keresése, amely egy egyetemes eljárást biztosítson számunkra az összes intelligens szabályozási feladatra, de ez egyelőre irreális elvárás. A mesterséges

neurális hálózatok hasznosak, mert a modellezési és a tanulási képességük az emberi viselkedésmóddal összehasonlítva tanulmányozható.

Az adaptációs algoritmusok általában lineáris rendszerekre és modellekre épülnek, a hangolható (megválasztható) paraméterek száma meghatározza az adaptációs módszer rugalmasságát. A neurális hálózatok alkalmazhatóak nemlineáris rendszerekre is, annak ellenére, hogy ez nem egyedüli megközelítése a megoldásnak, viszont egyes tanulási algoritmusok egyszerűek, összehasonlítva azokkal a módszerekkel, amelyeket a jelfeldolgozásban és szabályozástechnikában kifejlesztettek. A felügyelt tanulási algoritmusok többsége a gradiens módszeren alapszik, és csak az utóbbi időben jelentek meg a stabilitási elképzeléseken alapuló felhasználási módszerek. A CMAC neuronhálót széles körben alkalmazták különböző rendszerek adaptív modellezésére és szabályozására, mint például biotechnikai folyamatok irányítására, több szabadságfokú robotkarok, járművek irányítására, változatos szabályozási architektúrákat alkalmazva. A CMAC típusú neuronháló az egyik legnagyobb mértékben párhuzamosítható algoritmus [45], [46].

9.2. A CMAC hálók

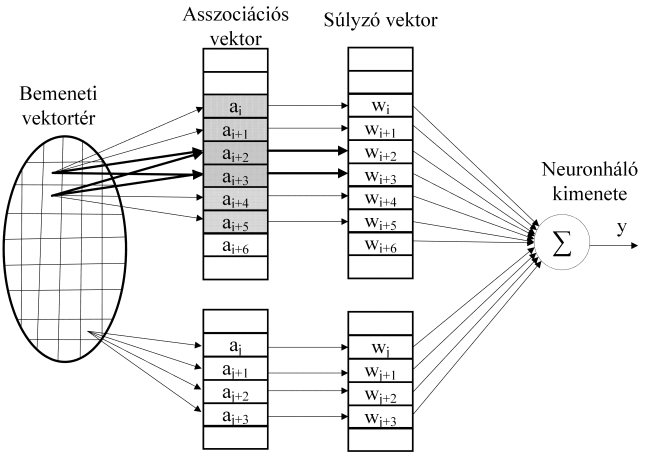
A CMAC mesterséges neurális hálót 1975-ben vezette be G. S. Albus. A CMAC egy asszociatív memória típusú háló (AMN Associative Memory Network), és nemlineráris, többdimenziós tanulási és modellezési algoritmusként használják.

9.2.1. A CMAC háló szerkezete

E rendszer alapszerkezete nagyon hasonlít Rosenblattnak az érzékelőjéhez, és mindkét rendszernek az eredeti leírása azonos [47]. A CMAC algoritmus a Rosenblatt-féle érzékelőből fejlődött ki.

A legegyszerűbb formájában a CMAC egy táblázat, amelynek a bázisfüggvényei helyben alakulnak ki. A legegyszerűbb változatban bináris kimenetű bázisfüggvényeket alkalmaztak. Ez azt eredményezi, hogy a CMAC kimenete szakaszonként konstans értéket szolgáltat.

Egy fix nemlineáris leképzést és egy adaptív, lineáris leképzést megvalósító részből áll. A hálózat kimenete a nemlineáris átalakítást végző bázisfüggvények lineáris kombinációjából áll (9.1. ábra). Az információ lokálisan, a háló egy kis részében memorizálódik.



9.1. ábra. A CMAC szerkezete

A hálózat n dimenziójú bemeneti vektora az x és a hálózat belső megjelenítése a ρ dimenziójú vektor által történik, ezt a vektort nevezzük a bemenet asszociációs vektorának vagy a bázisfüggvény kimenő vektorának. A CMAC háló y kimenete a bázisfüggvények lineáris kombinációja [48]:

$y = \sum_{i=1}^{\rho} w_{i^*} a_{i^*}$ – a képlet szerint minden átlapolódó rétegben csak az aktív bázisfüggvénnyel számoltunk.

Minden bázisfüggvényhez hozzárendelünk egy tartót vagy receptív mezőt, amely a bemenő pontok halmazából áll, amelyre a bázisfüggvény kimenete nem lesz zérus. A tervező meg kell jelöljön egy n dimenziójú struktúrát, amely normalizálja a bemenő teret. Minden dimenzióban a bemenetnek meg kell adni a minimális és maximális értékeket, amellyel az a mező generálódik, amelyen a CMAC értelmezve van. A bemeneti halmazon az x_i^{min} és x_i^{max} közötti belső pontok halmaza, r szintén meg kell legyen jelölve, mert ezek helye határozza meg a CMAC háló érzékenységét a bemeneti mezőben. A belső pontok pozíciója nagyon fontos ahhoz, hogy a CMAC a kívánt célfüggvényt minimális hibával képes legyen megtanulni. Ha a kívánt függvény értéke gyorsan változik, akkor több belső pontot kell elhelyezni ebben a mezőben. Hasonlóan, ha a függvény értéke lassan változik egy mezőben, akkor kevés számú belső pontot kell meghatározni.

Amint már említettük, a CMAC fix, nemlineáris leképzést valósít meg. A CMAC háló nagymértékben hasonlít az RBF hálóhoz, a különbség az, hogy míg az RBF háló esetében a bázisfüggvények helye nincs pontosan

meghatározva, addig a CMAC háló esetében a bázisfüggvények helye rögzített.

A CMAC algoritmus két különböző részre bontható. Az első egy nemlineáris transzformáció, amely levetíti a háló bemeneteit egy nagyobb méretű mezőre, ahol csak egynéhány változónak van nem zérus kimenete. A tervezőnek meg kell jelölnie az általánosító ρ paramétert, amely meghatározza a rejtett rétegbeli nem zérus változók számát, és ugyancsak meg kell határoznia a háló belső mezőjének a dimenzióját, amely befolyásolja a kimenetet.

A második rész az első rész által aktivált változókat lineárisan kombinálja a súlytényezőkkel.

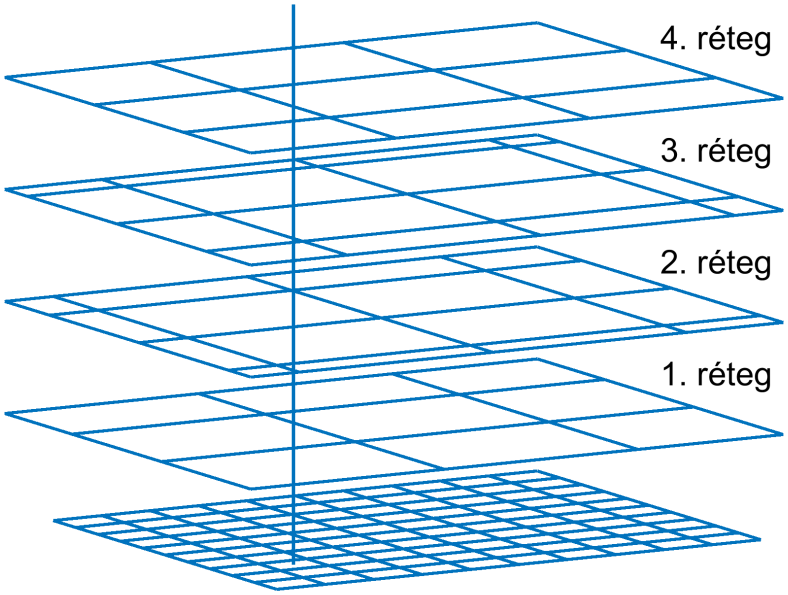
Az általánosító paraméter meghatározza a bázisfüggvény-tartó méretét is. Minden bázisfüggvénynek van egy ρ egységnyi tartója mindegyik dimenzióban, és a tartó ρ^n térfogatú n dimenziós hiperkocka. A ρ szinteken a tartók oly módon osztódnak el, hogy kielégítsék az egyenletes vetületek elvét.

A bázisfüggvény-tartók rácsszerűen vannak elosztva (9.2. ábra), oly módon, hogy ezen alapok vetülete mindegyik tengelyre egyenletes, érvényesítve az egyenletes vetületek elvét. Amilyen mértékben egy bemenet végighalad a rácson párhuzamosan a bemeneti tengellyel, átlépve az egyik cellából a másikba, a kimenetszámításnál alkalmazott bázisfüggvényeknek a száma konstans kell legyen és független a bemenet helyzetétől.

Albusnak az eredeti felosztása minden felső i-edik rétegben a receptív mező első sarkát a rács átlójának mentén helyezi, oly módon, hogy az első tartó az i-edik felső rétegen egybeesik az $(i, i, i, \ldots, i)$ belső rácsponttal, ahol $i = 1, \ldots, \rho$. A 9.2. ábrán egy kétdimenziós CMAC van bemutatva, ahol $\rho = 3$.

9.2.2. A bázisfüggvények formái

A bázisfüggvény méretét és eloszlását a tervező az általánosító ρ paraméter értékének megjelölésével határozza meg. Léteznek alternatív stratégiák is a felső rétegek felosztására, az eredeti CMAC bináris bázisfüggvényekkel rendelkezik, és a kimenet zérus vagy egy lehet. Minden n méretű cellában a hálózat kimenete konstans, úgy, hogy az alapalgoritmus egy táblázatban rögzíthető. A bináris kimenetű bázisfüggvények esetében csak egész típusú számítások felhasználására van igény, ebből adódóan az alapalgoritmus nagyon hatékony. Egyes feladatok megoldása a hálózat folytonos kimenetét igényli, és így az egyedüli mód ennek az elérésére a magasabb rendű bázisfüggvények használata.



9.2. ábra. CMAC rétegek

A ρ általánosító paraméter nemcsak a bázisfüggvények számát jelöli meg, hanem meghatározza a bázisfüggvény-tartó méretét is. Ahogy nő a ρ, úgy nő a bázisfüggvény-tartó mérete is, és a tanulás kevésbé válik helyi jellegűvé. Meghatározás szerint a bázisfüggvény kimenete akkor és csakis akkor válik nem zérussá, ha a bemeneti pont egy tartó hyperkocka belsejében van. A háló kimenete a ρ súlyok összegéből számítható ki. Mindegyik bemenet mindegyik átlapolódó rétegen belül egyetlen bázisfüggvény tartóján található. Egy kétdimenziós példára, ahol a $\rho = 3$ és a bázisfüggvény-tartók mérete: 3,3, az egymásra lapuló rétegek elhelyezése a 9.2. ábrán van szemléltetve.

Az átlapolódó rétegen (overlayek) meghatározunk egy elhelyezési vektort, d-t, amely n dimenziós, és mindegyik átlapolódó réteg a d vektorok értékétől függően az előző overlay-hez képest van elhelyezve. Az elhelyezési vektor elemei kielégítik a következő feltételt: $1 < d_i < \rho$, ahol d_i co-prim ρ-val.

Egy bemenetbe bevont bázisfüggvények száma nem függ a bemenő tér dimenziójától. Az átlapolódó rétegek relatívan vannak elhelyezve egymáshoz képest, úgy, hogy mindig egyenletes legyen a vetületük minden bemenő

tengelyre. Egy CMAC-ban minden bázisfüggvénynek egy jól meghatározott, releváns szerepe van a kimenetben.

9.2.3. A CMAC háló paramétereinek hangolása

A CMAC hálózatok tanításánál LMS algoritmust alkalmazhatunk (9.1):

$$w_i\left[k+1\right] = w_i\left[k\right] + \mu\delta g_i \tag{9.1}$$

ahol g_i az x bemenetre előállított asszociációs vektor eleme az i-edik átlapolódó rétegből, a módosítás mértékét a $\mu\delta g_i$ adja.

Az asszociatív vektor aktív elemei azt mondják meg, hogy milyen címen levő súlytényezőt kell módosítani. A tanító pontok hatása csak korlátos bemeneti tartományra terjed ki, a hálózat lokális általánosító képességgel rendelkezik, tehát egymástól távoli tartományban a válaszok egymástól függetlenül alakulnak ki, a hálózat gyorsan tanul.

A modellező képesség javítható, ha magasabb rendű bázisfüggvényt alkalmazunk (0 rendű – bináris bázisfüggvény, amelynek értéke „1”, ha a bemenet az érzékelési mezőbe esik, míg az összes többire „0”). Magasabb rendű CMAC hálóknál a bináris helyett lineáris, kvadratikus bázisfüggvényeket használhatunk. Hasznosak még a lépcsős bázisfüggvények, így az általánosító képesség nem csak additív függvényosztályokra lesz igaz. Magasabb rendű, ún. Spline bázisfüggvényekkel megvalósítható a folytonos leképzés, de elveszítjük a feldolgozási, tanulási, lekérdezési sebességet.

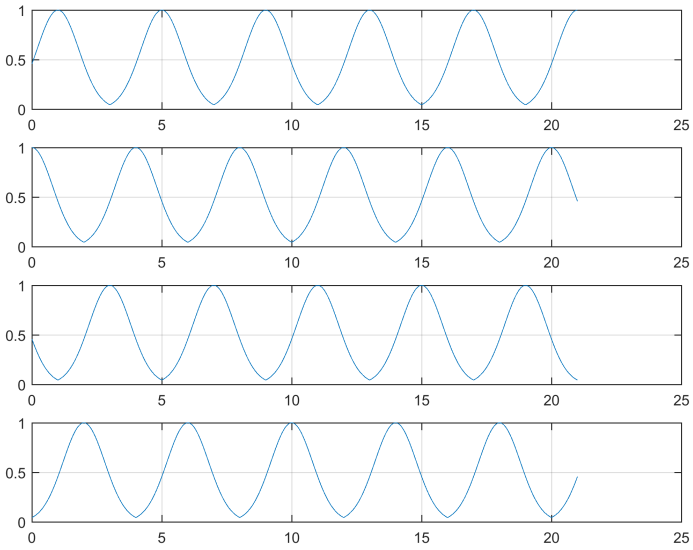
Az alábbi rajzokon egybemenetű rendszerre van szemléltetve a bázisfüggvények elhelyezése (9.3), valamint az, hogy a tanítási ciklusok során (9.4.a – 9.4.h ábra) a neuronháló hogyan közelíti meg a célfüggvényt.

$\rho = 4$ általánosító paraméter értéke $min_x = 0$, $max_x = 21$, $r = 20$ – belső pontok száma

A négy átlapolási rétegen a $d = (0, 1, 2, 3)$ elhelyezési paramétereknek megfelelően a bázisfüggvények a következőképpen vannak elhelyezve (9.3. ábra).

9.3. Feladat

Tervezzünk előbb egy egybemenetű és egy kétbemenetű, egykimenetű CMAC hálót, és alkalmazzuk egy függvényapproximációs feladat megoldására. Különböző tanítási értékekre végezzük el a háló tanítását.



9.3. ábra. Bázisfüggvények elhelyezése

Az egydimenziós bemenet esetében a feladat megvalósításához a következő főbb lépéseket kell elvégezni:

- A bemenetek számának – n, valamint minden bemenetre a minimum és maximum értékeknek a meghatározása.
- A belső pontok számának a meghatározása, majd belső pontok alapján a minimum és maximum pontok közötti szakaszok számának meghatározása.
- A ρ általánosító paraméter megválasztása (ez meghatározza, hogy hány neuron aktív minden bemenetre).
- Az általánosító paraméter és a belső pontok száma meghatározza a bázisfüggvény-tartó méretét.
- A bázisfüggvények elhelyezési vektorának a meghatározása. Annyi szintre osztjuk a bázisfüggvényeket, amennyi a ρ általánosító paraméter értéke. A szakaszok hosszát a (9.2) képlet adja meg:

$$s = \frac{max - min}{r_i + 1}. \tag{9.2}$$

- Válasszuk meg a μ tanítási együtthatót.

– Inicializáljuk a súlymátrixot. A súlytényezőket előnyös egy kétdimenziós tömbben tárolni. Az egyik index szerint szelektáljuk a szintet, a második index szerint pedig meghatározzuk, hogy az adott szinten hányadik súlytényezőről van szó.

El kell készíteni a tanító halmazt, amely alapján el lehet végezni a neuronháló tanítását. A tanítási ciklus lépései:

– Veszünk egy elemet a tanító halmazból.
– Minden szinten kiszámítjuk a bázisfüggvények értékét. Mivel minden egyes bázisfüggvény paramétere megegyezik, az x-et, amelyre ki kell számolni a bázisfüggvény értékét, visszavezetjük egyetlen bázisfüggvényre. Ez a művelet rétegenként megvalósítható a ρ, s és d paraméterek függvényében egy egyszerű balra tolással. Az eltolt x' értékkel (9.3) kiszámítjuk a bázisfüggvény értékét:

$$x' = x_i - \rho s \left[\frac{x_i - s d_j}{\rho s} \right] + s \left(\rho - d_j \right). \tag{9.3}$$

  Az x vagy x' értékkel számított bázisfüggvények értékei megegyeznek.
– Minden szinten kiszámítjuk az aktív bázisfüggvényekhez tartozó súlytényezők indexét (9.4):

$$index_j = \left[\frac{x_i - s d_j}{\rho s} \right] + 1. \tag{9.4}$$

– Kiszámítjuk a háló kimenetét (9.5):

$$y = \sum_{i=1}^{\rho} w_{i,index[i]} g_i \left(x \right). \tag{9.5}$$

– Kiszámoljuk a hibát (9.6):

$$\delta = d_i - y_i. \tag{9.6}$$

– Minden átlapolódó szinten tanítjuk a súlytényezőket (9.7):

$$w_{i,j} \left[k+1 \right] = w_{i,j} \left[k \right] + \mu \delta g_i \left(x \right) \tag{9.7}$$

  ahol $j = index[i]$ megadja az i-edik átlapolódó szinten az aktív súlytényezők indexét.

A tanítási ciklust addig ismételjük, amíg a hiba egy adott érték alá csökken, végül pedig ábrázoljuk a háló kimenetét és a hiba alakulását, amelyeket elmentünk minden lépésben.

A CMAC háló megvalósítását a 9.1. példa szemlélteti.

9.1. példa. *CMAC háló*

```
clc;
clear all;
close all;

min_x = 0;                          %minimum
max_x = 21;                         %maximum
ri = 20;                            %belső pontok száma
c = 4;                              %átlapoló szintek száma és a
    bázisfüggvény hossza
s = (max_x-min_x)/(ri+1);     %szakaszok hossza
n = 1;                              %bemenetek száma
%n = 3^n;                           %bázisfüggvények száma
d = [0 1 2 3 4];
mu = 0.01;
szigma=0.7
%tanító halmaz meghatározása
dx=(max_x-min_x)/100;
meres=1
f_name='CMAC_'
for i=0:100
    %x(i) = (i*dx-min_x)/(max_x-min_x)+min_x;          %
        bemenet
    x(i+1) = i*dx
    f(i+1) = fugv(x(i+1)+min_x);                       %
        függvényérték
end
%súlyzók és középértékek meghatározása
W = zeros(c, fix(ri/c)+2);
yy = [];
xx=[];
figure(1);
plot(x,f);
hold on;
```

```
delta_v=[];

for p=1:100 % tanítási ciklusok ismétlése
    yy = [];
    delta_v(p)=0;
    for i = 1:length(x)
        %aktív súlyzók kiszámolása
        index = [];
        y = 0;
        for j=1:c
            xx(j) = x(i)-(c)*s*floor((x(i)-s*d(j))/(c*
                s)) -s*d(j); % x eltolás
            index(j) = floor( ( x(i)- s*d(j) ) / c*s )
                + 2;% aktív súlyzók indexének
                eltárolása
            g(j)= gauss( c*s/2, xx(j),szigma ); %j-
                edik rétegen bázisfüggvény értékének
                kiszámítása
            y = y + W(j,index(j))*g(j);  %neuronháló
                kimenetének kiszámítása
        end
       % pause
         yy = [yy y];
       if p>1
        delta=fugv( x(i)+min_x )-y; %hibaszámítás
        for j=1:c
            W(j,index(j)) = W(j,index(j)) + mu*delta*g
                (j); % súlyzók tanítása
        end
        delta_v(p-1)= delta_v(p-1)+delta*delta;
       end;
    end
% a célfüggvény és neuronháló kimenetének ábrázolása a
    tanítási ciklusok során
 if mod(p-1,5)==0
    f1=strcat(f_name,num2str(meres));
    figure(2)
    plot(x+min_x,f,'k','Linewidth',2)
    hold on
```

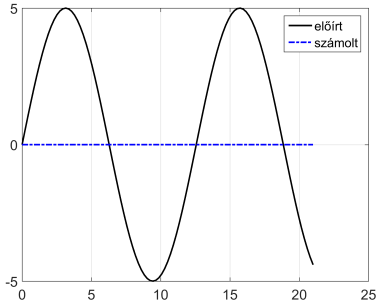
```
      plot(x+min_x, yy,'b-.','Linewidth' ,2);
      grid
      legend('előírt','számolt')
      set(gca,'FontSize',18)
      print('-dpng','-r300',f1);
      pause(0.1)
      meres=meres+1;
      hold off
  end
end
%eredmény ábrázolása
figure
plot(delta_v,'Linewidth' ,2)
set(gca,'FontSize',18)
grid
ylabel('Hiba')
xlabel('Tanítási ciklusok')
print('-dpng','-r300','hiba_alakulasa_b');
```

A példában megvalósított CMAC háló négy réteget tartalmaz. Célfüggvényként egy szinuszosan változó jelet alkalmazunk. A tanulás alakulása a tanítási ciklusok során a 9.4. ábrákon van szemléltetve. A tanítás során a súlytényezők kezdetben zérussal vannak inicializálva, amint a 9.4. első ábráján is jól látható, hogy a tanítás elkezdése előtt a számolt kimenet zérus értéket vesz fel. A tanítási együttható $\mu = 0{,}01$.

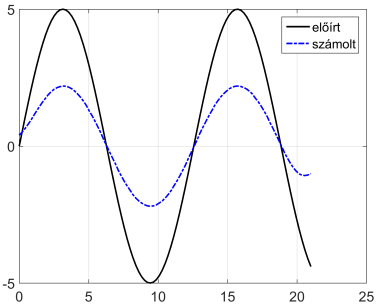
A receptív mezők Gauss-függvényekkel vannak lefedve ($\sigma = 0{,}7$). A tanítási együttható céltudatosan van egy nagyon kicsi értékre csökkentve azért, hogy ciklusonként értékelhető legyen a tanulás alakulása.

A számolt kimenet minden 5-dik ciklust követően van ábrázolva a 9.4. ábrákon.

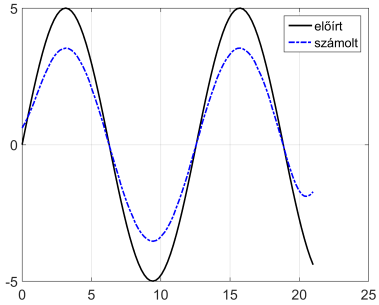
A 9.5. ábrán a négyzetes hiba van ábrázolva. A tanítás 100 ciklusban valósult meg. A 40-dik ciklust követően a négyzetes hiba 0,7 érték alá csökken. Összesen 101 mintát tartalmaz a tanító halmaz, tehát egy mintára a négyzetes hiba átlag értéke kisebb: 0,007.



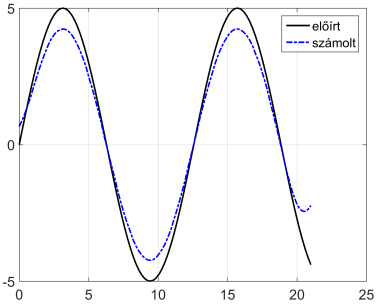
Kezdeti állapot



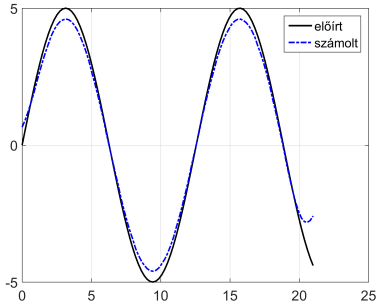
5. tanítási ciklus



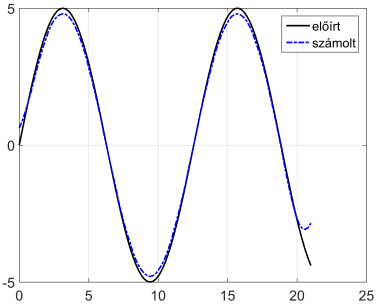
10. tanítási ciklus



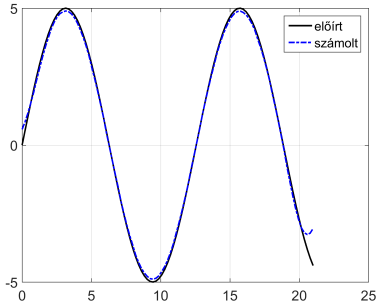
15. tanítási ciklus



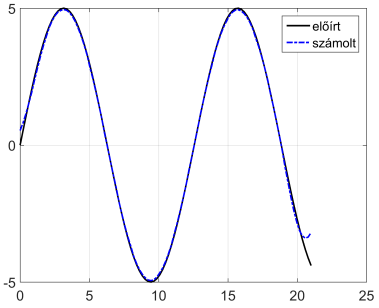
20. tanítási ciklus



25. tanítási ciklus

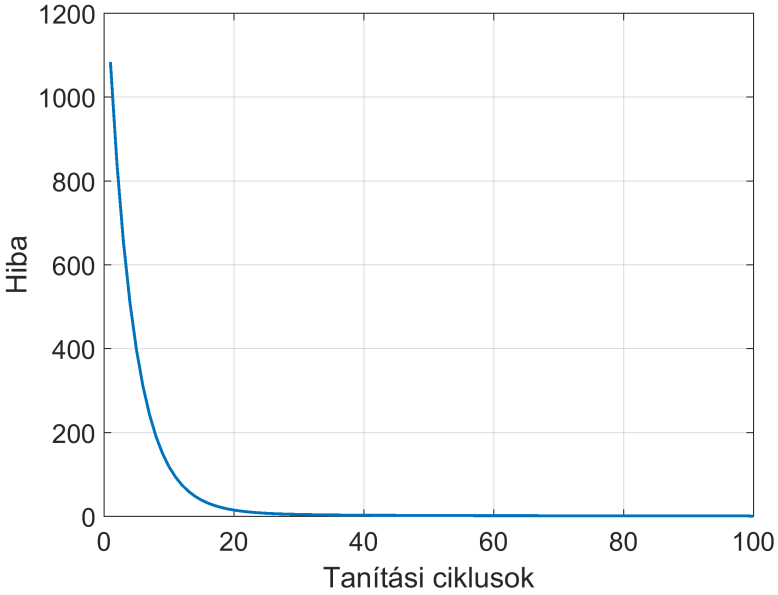


30. tanítási ciklus



35. tanítási ciklus

9.4. ábra. Tanítási ciklusok alakulása



9.5. ábra. A hiba alakulása

II. rész

Fuzzy logika. Fuzzy következtető rendszerek

10. fejezet

Fuzzy logika

Ebben a részben a fuzzy logikával kapcsolatos elemeket, fuzzy halmazok definícióját, fuzzy tagsági függvényeket, fuzzy halmazműveleteket, fuzzy relációkat, fuzzy relációk vetületét, a hengeres kiterjesztést tárgyaljuk. Továbbá bemutatjuk a fuzzy következtető rendszerek felépítését, a fuzzyfikációt, a defuzzyfikációs eljárásokat, a fuzzy szabálybázist, a fuzzy szabályok kitöltését. A MATLAB függvényeket használva ismertetjük a fuzzy következtető rendszerek tervezésével kapcsolatos fontosabb tudnivalókat.

10.1. Elméleti alapfogalmak

A fuzzy logika nem más, mint a bináris logika általánosított formája. A fuzzy logika egy multivalens, többértékű logikai rendszer. Míg a bivalens, kétértékű logika lehetséges értékei 0 vagy 1, addig egy fuzzy rendszer esetében a lehetséges érték elvileg végtelen is lehet. A klasszikus logika szerint egy elem halmazba tartozása egyértelműen megállapítható: $a \in U \rightarrow \{0, 1\}$, egy tetszőleges elemről el tudjuk dönteni, hogy eleme-e az adott halmaznak vagy sem [49], [50]. Ha hozzá tartozik, akkor egy logikai igaz, ha nem, egy logikai hamis értékkel jellemezzük. Az egyszerűség kedvéért jelöljük a logikai igaz értéket 1-gyel, a hamis értéket 0-val. Vagyis az, hogy egy elem beletartozik-e az U halmazba, vagy egy 0-val, vagy egy 1-gyel jellemezhető. A többértékű rendszerben az elemnek egy halmazhoz való tartozását a következőképpen írhatjuk fel: $\{a \in U\} \rightarrow [0, 1]$. A fuzzy logika abban hoz újat, hogy a halmazba tartozás nem egyszerűen 0, illetve 1 binárisan értékelhető, hanem köztes értékek is léteznek, amelyek megmutatják, hogy egy

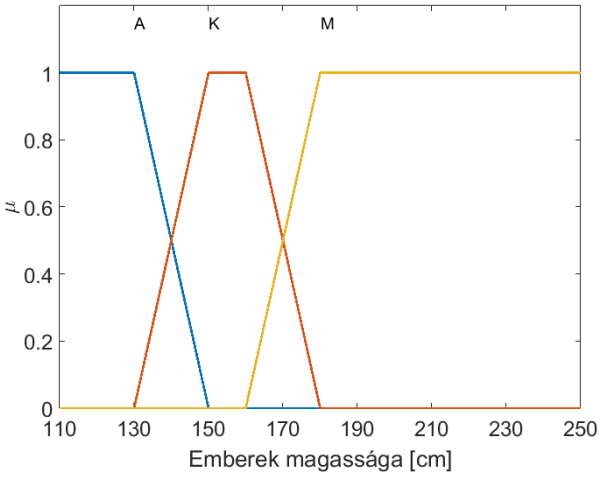
adott elem mennyire tartozik a halmazba: nagyon, kissé, kevésbé vagy egyáltalán nem. Így minden U halmazbeli elemhez hozzárendelünk egy számot, általában 0 és 1 közötti értéket, mely meghatározza az adott elem halmazba tartozásának mértékét. Tehát a különbség a kétértékű és többértékű rendszer között a logikai kifejezések értékkészletében van. A kétértékű rendszer esetében a {0, 1} halmaz két elemet tartalmaz, míg a többértékű rendszer a [0, 1] intervallumban végtelen sokat. Szükség van egy rendszerre, amely megadja a logikai értéket a [0, 1] intervallumból.

Egy klasszikus példa a bináris és fuzzy halmazok szemléltetésére az emberek magasság szerinti halmazba sorolása. Legyen egy E halmazunk, amely az emberek cm-ben kifejezett testmagassága, és vegyük csak az egész értékeket: E = 130, 131,. . . , 183,. . . , 250. Az E halmazbeli elemeket próbáljuk alacsony (A), közepes (K), magas (M) növésű emberek részhalmazokba sorolni.

A klasszikus halmazelmélet szerint, ha az E halmazbeli elemet valamely A, K vagy M részhalmazba szeretnénk sorolni, akkor élesen kell találjunk egy elemet (például 130 és 170 cm), amelynél magasabb emberek az M = 170, 171, . . . , 250 halmazba tartoznak, a két intervallum között a közepesek a K = 130, 131, . . . , 169 halmazba, míg a 130 cm-nél is alacsonyabbak az A halmazba.

Az életben viszont ilyen éles határokat gyakran nem szabhatunk, azt mondjuk valakire, hogy „a körülbelül 155 cm magas illető nagyjából közepesnek mondható”. Az állításban van egyfajta bizonytalansági tényező, „körülbelül”, illetve „nagyjából”. A fuzzy halmazelméletben az egyes elemekhez rendelt értékpárok éppen ezt a bizonytalanságot próbálják feloldani. A fuzzy halmazelméletben az egyes részhalmazok elemeihez hozzárendelünk egy-egy számot, amely megadja, hogy az adott elem milyen mértékben, milyen lehetőséggel tartozik az adott halmazhoz. Például: A=110(1), 120(1), 130(1), 135(0,75), 140(0,5), 144(0,3), 149(0,05), 150(0), 160(0), 170(0), 180(0), 190(0); K=130(0), 135(0,25), 140(0,5), 144(0,7), 149(0,95), 150(1), 160(1), 170(0,5), 180(0), 190(0); M=130(0), 140(0), 150(0), 160(0), 162(0,1), 170(0,5), 174(0,7), 180(1), 210(1).

A két halmazban lehetnek azonos vagy teljesen különböző elemek is. Az elemek több halmazhoz is tartozhatnak, és mindegyik halmazhoz való tartozást leírja az elemhez rendelt tagsági érték (10.1. ábra). Az elemekhez rendelt számok között sem halmazon belül, sem két halmaz között semmilyen összefüggés nincs előírva, leszámítva azt, hogy szemantikai jelentéssel bíró adatoknak kell lenniük. Ha a halmazhoz való hozzárendelésre függvényt alkalmazunk, akkor a 10.1. ábrán szemléltetett tagsági függvényt kapjuk.



10.1. ábra. Emberek osztályozása magasságuk szerint

A tradicionális halmazokkal szemben, amelyekben egy elem vagy a halmazhoz tartozik, vagy nem, egy fuzzy halmaz értelmében az elem részben is tartozhat a halmazhoz.

A halmazhoz tartozás mértékét a μ fuzzy tagsági függvény határozza meg. A tagsági függvény a fuzzy halmaz elemeihez egy nulla és egy közötti valós számot rendel hozzá.

Lofti Zadeh a Kaliforniai Berkeley Egyetem professzoraként 1965-ben közölte a *Fuzzy Sets* című munkáját, amelyben letette a fuzzy halmazelmélet és ennek kiterjesztéseként a fuzzy logika matematikai alapjait [51].

Zadeh megfigyelte, hogy a hagyományos számítógépes logika nem képes kezelni a szubjektív, hiányos vagy bizonytalan információt, ezért létrehozta a fuzzy logikát, amely lehetővé teszi a számítógépek számára, hogy az emberi gondolkodáshoz hasonlóan kezelje az információt.

Legyen A az U univerzum fuzzy részhalmaza ($A \subset U$) és legyen $\mu : A \rightarrow [01]$ tagsági függvény, akkor az A fuzzy halmazt az U univerzumból a következőképpen (10.1) írjuk le [52]:

$$A = \{(x, \mu(x) \,|x \in U)\} \tag{10.1}$$

ahol x – egy tulajdonságérték az U univerzumból, $\mu(x)$ tagsági függvény: a tulajdonság-értékhez hozzárendeli a halmazhoz tartozás mértékét [0, 1].

Diszkrét esetben a fuzzy halmazt a (10.2) szerint írjuk le:

$$A = \frac{\mu_A(x_1)}{x_1} + \frac{\mu_A(x_2)}{x_2} + \cdots + \frac{\mu_A(x_n)}{x_n} = \sum_{i=1}^{n} \frac{\mu_A(x_i)}{x_i}. \tag{10.2}$$

Az $X = \{x_1, x_2, \cdots, x_n\}$ esetben a fuzzy halmazok tagsági értékét táblázatos formában is célszerű tárolni (10.1. táblázat)

10.1. táblázat. Fuzzy halmaz táblázatos leírása

x_1	x_2	$\cdots$	x_n
$\mu_A(x_1)$	$\mu_A(x_2)$	$\cdots$	$\mu_A(x_n)$

Folytonos esetben pedig a (10.3) képlet alapján határozzuk meg:

$$A = \int_{x \in U} \frac{\mu_A(x)}{x}. \tag{10.3}$$

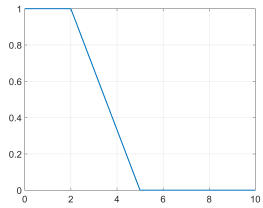
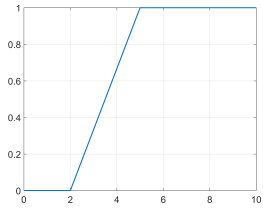
A fuzzy halmaz tartója (10.4) azon x elemekből áll, melyekre az A halmazban pozitív tagsági értékkel rendelkeznek:

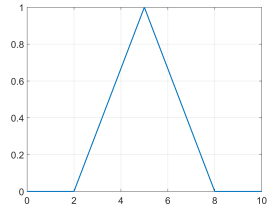
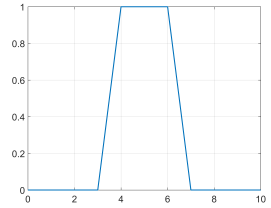
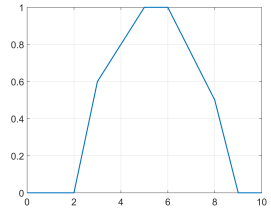
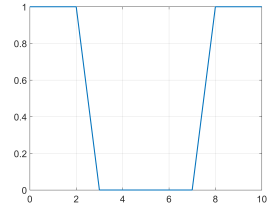
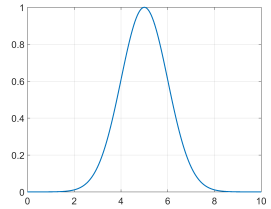
$$supp(A) = \{x \in X | \mu_A(x) > 0\}. \tag{10.4}$$

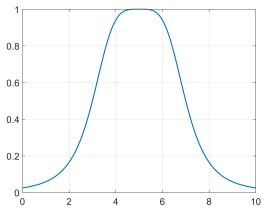
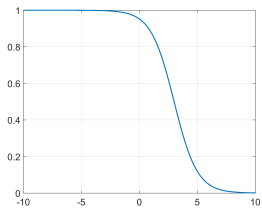
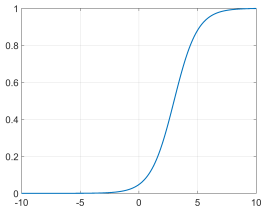
Egy X alaphalmazon értelmezett két A és B fuzzy halmaz azonos, ha a tagsági függvényeik megegyeznek: $A = B$, ha $\mu_A(x) = \mu_B(x) \, \forall x \in X$.

A tagsági függvények formája döntő a fuzzy halmazok leírásában. Az alábbi ábrán láthatjuk a fontosabb tagsági függvényformákat:

10.2. táblázat. Tagsági függvények

Függvénytípus	Egyenlet
Z tagsági függvény 	$\mu(x) = \begin{cases} 1 & \text{ha} \quad x \le a_1 \\ \frac{a_2 - x}{a_2 - a_1} & \text{ha} \quad a_1 < x < a_2 \\ 0 & \text{ha} \quad x \ge a_2 \end{cases}$
S tagsági függvény 	$\mu(x) = \begin{cases} 0 & \text{ha} \quad x \le a_1 \\ \frac{x - a_1}{a_2 - a_1} & \text{ha} \quad a_1 < x < a_2 \\ 1 & \text{ha} \quad x \ge a_2 \end{cases}$

Függvénytípus	Egyenlet
Háromszög 	$\mu(x)=\begin{cases} 0 & \text{ha} \quad x \le a_1 \\ \frac{x-a_1}{a_2-a_1} & \text{ha} \quad a_1 < x \le a_2 \\ \frac{a_3-x}{a_3-a_2} & \text{ha} \quad a_2 < x < a_3 \\ 0 & \text{ha} \quad x \ge a_3 \end{cases}$
Trapéz 	$\mu(x)=\begin{cases} 0 & \text{ha} \quad x \le a_1 \\ \frac{x-a_1}{a_2-a_1} & \text{ha} \quad a_1 < x < a_2 \\ 1 & \text{ha} \quad a_2 \le x \le a_3 \\ \frac{a_4-x}{a_4-a_3} & \text{ha} \quad a_3 < x < a_4 \\ 0 & \text{ha} \quad x \ge a_4 \end{cases}$
Általánosított trapéz 	$\mu(x)=\begin{cases} 0 & \text{ha} \quad x \le a_1 \\ \frac{b_2(x-a_1)}{a_2-a_1} & \text{ha} \quad a_1 < x < a_2 \\ \frac{(b_3-b_2)(x-a_2)}{a_3-a_2}+b_2 & \text{ha} \quad a_2 \le x \le a_3 \\ b_3=b_4=1 & \text{ha} \quad a_3 < x < a_4 \\ \frac{(b_4-b_5)(a_4-x)}{a_3-a_2}+b_5 & \text{ha} \quad a_4 \le x \le a_5 \\ \frac{b_5(a_6-x)}{a_6-a_5} & \text{ha} \quad a_5 < x < a_6 \\ 0 & \text{ha} \quad x \ge a_6 \end{cases}$
V típusú tagsági függvény 	$\mu(x)=\begin{cases} 1 & \text{ha} \quad x \le a_1 \\ \frac{a_2-x}{a_2-a_1} & \text{ha} \quad a_1 < x < a_2 \\ 0 & \text{ha} \quad a_2 \le x \le a_3 \\ \frac{x-a_3}{a_4-a_3} & \text{ha} \quad a_3 < x < a_4 \\ 1 & \text{ha} \quad x \ge a_4 \end{cases}$
Gauss tagsági függvény 	$\mu(x) = e^{-\frac{(x-c)^2}{2\sigma^2}}$

Függvénytípus	Egyenlet
Általánosított haranggörbe 	$\mu(x) = \frac{1}{1+\lvert\frac{x-c}{a}\rvert^{2b}}$
Z típusú tagsági függvény 	$\mu(x) = \frac{1}{1+e^{a(x-b)}}$
S típusú tagsági függvény 	$\mu(x) = \frac{1}{1+e^{-a(x-b)}}$

10.1.1. Fuzzy logikában alkalmazott módosító operátorok

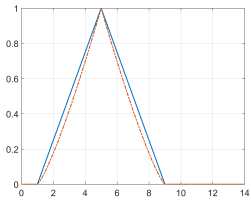
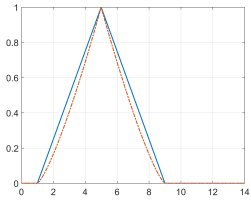
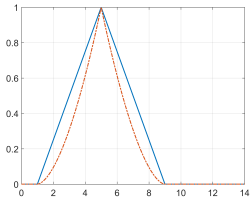
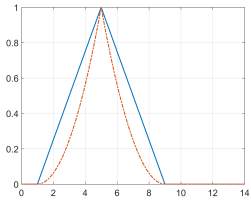
A lingvisztikai módosító szavakat [53], amely lényegében nem más, mint egy egyváltozós operátor, a tagsági függvények formájának módosítására alkalmazzák. A lingvisztikai módosítónak két fő hatása van az általuk modulált fuzzy halmazokra gyakorolt hatásuk tekintetében. Megkülönböztethetünk egy megerősítő hatást [nagyon, erős], illetve egy gyengítő hatást [többé-kevésbé, viszonylag]. A módosító szavakat három osztályba sorolhatjuk [54], [55]:

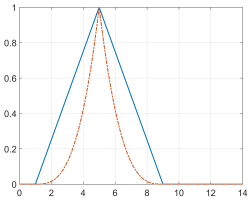
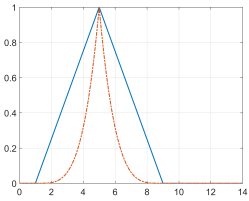
- korlátozó módosítók – koncentrátorok (kontrakció);
- kiterjesztő módosítók – dilatátorok (dilatáció);
- kontraszt erősítő módosító operátorok.

Korlátozó módosítók – koncentrátorok (koncentráció). Összehasonlítva a kiterjesztő, dilatáló módosító szavakkal, a dilatáció a kontrakciónak az

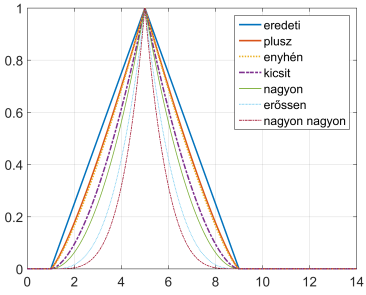
ellentéte. A korlátozó operátor hatására az eredményezett tagsági függvény értékei kisebbek az eredeti tagsági függvény értékeinél.

10.3. táblázat. Kontrakciós operátorok

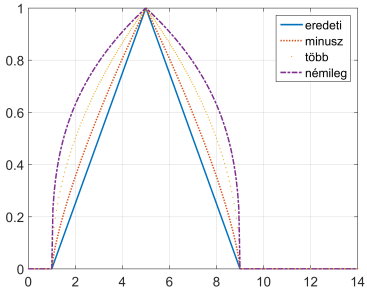
Kontrakciós operátor hatása	Kontrakciós operátor egyenlete
	$\mu_{plusz(x)} = \mu\left(x\right)^{1.25}$
	$\mu_{enyhén}(x) = \mu\left(x\right)^{1.3}$
	$\mu_{kicsit}(x) = \mu\left(x\right)^{1.7}$
	$\mu_{nagyon(x)} = \mu\left(x\right)^{2}$

Kontrakciós operátor hatása	Kontrakciós operátor egyenlete
	$\mu_{erőssen}(x) = \mu(x)^3$
	$\mu_{nagyonnagyon}(x) = \mu(x)^4$

Az alábbi rajzokon a korlátozó (*plusz, enyhén, kicsit, nagyon, erősen, nagyon-nagyon*) (10.2. ábra), valamint kiterjesztő (*mínusz, plusz, némileg*) (10.3. ábra) jellegű módosító operátorok vannak szemléltetve. Az ábrákon a folytonos vonal az eredeti tagsági függvény, míg a szaggatott vonallal jelölt a számított érték. A kiterjesztő operátorok alkalmazása esetében az eredményezett tagsági értékek nagyobbak, összehasonlítva az eredeti tagsági értékekkel.

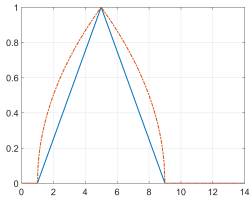
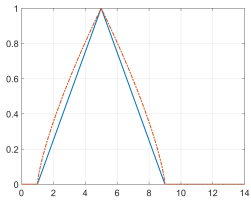
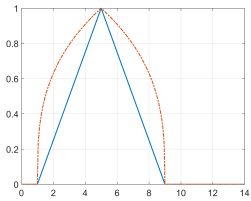


10.2. ábra. Korlátozó módosító operátorok összehasonlítása



10.3. ábra. Dilatáló módosító operátorok összehasonlítása

10.4. táblázat. Dilatációs operátorok

Dilatációs operátor hatása	Dilatációs operátor egyenlete
	$\mu_{more}(x) = \mu(x)^{\frac{1}{2}}$
	$\mu_{minus}(x) = \mu(x)^{\frac{3}{4}}$
	$\mu_{némileg}(x) = \mu(x)^{\frac{1}{3}}$

10.1.2. Fuzzy halmazműveletek

A fuzzy halmazok kombinálására, összekapcsolására az általános halmazelmélethez hasonlóan alkalmazhatók az alapvető műveletek. Több lehetőség is van az egyesítés, metszet és komplemens halmazműveletek fuzzy halmazokra való alkalmazására. Az egyesítés és metszet fogalmát a fuzzy halmazelméletben úgy általánosíthatjuk, hogy az $A \cup B$, valamint $A \cap B$ fuzzy halmazok tetszőleges $x \in X$ elemének tagsági értéke csak az A, illetve B halmazok $\mu_A(x)$ és $\mu_A(x)$ tagsági függvényektől függjön.

Az $s : [0,1] \times [0,1] \to [0,1]$ és $t : [0,1] \times [0,1] \to [0,1]$ függvények segítségével definiálhatjuk a két fuzzy halmaz egyesítését (10.5) és metszetét (10.6):

$$\mu_{A \cup B}(x) := s(\mu_A(x), \mu_B(x)) \tag{10.5}$$

$$\mu_{A\cap B}(x) := t\left(\mu_A(x), \mu_B(x)\right). \tag{10.6}$$

A t-norma kétváltozós, szimmetrikus, asszociatív, monoton növekvő operátor, egységeleme 1, nulleleme pedig 0 [56]. A t-normára teljesülnek a következő tulajdonságok $(x, a, b, c, d, e \in [0, 1])$:

- kommutatív: $t(a, b) = t(b, a)$
- asszociatív: $t(a, t(b, c)) = t(t(a, b), c)$
- peremfeltételek: $t(1, x) = t(x, 1) = x$, $t(0, x) = t(x, 0) = 0$
- monotonitás: Ha $a \leq c$ és $b \leq d$ akkor $t(a, b) \leq t(c, d)$

Az s norma a következő tulajdonságokkal rendelkezik:

- kommutatív: $s(a, b) = s(b, a)$
- asszociatív: $s(a, t(b, c)) = s(t(a, b), c)$
- peremfeltételek: $t(0, x) = t(x, 0) = x$, $t(1, x) = t(x, 1) = 1$
- monotonitás: Ha $a \leq c$ és $b \leq d$ akkor $s(a, b) \leq s(c, d)$

A leggyakrabban alkalmazott műveletek a fuzzy egyesítésre (10.8) és metszetre (10.7):

$$t\text{-norma} : \mu_{A\cap B}(x) = min\left(\mu_A(x), \mu_B(x)\right) \tag{10.7}$$

$$s\text{-norma} : \mu_{A\cup B}(x) = max\left(\mu_A(x), \mu_B(x)\right) \tag{10.8}$$

Az s-norma a fuzzy unió, míg a t-norma a fuzzy metszet.

A komplemens számítás a fuzzy halmazokra (10.9):

$$\mu_C(x) = 1 - \mu_A(x) \tag{10.9}$$

A max és min operátorokra alapuló normák a Zadeh-féle normák. Számos lehetőség van a t és s normák meghatározására, mint például Lukasiewicz-, Hamacher-, Einstein-, Frank-, Yager-, Scweiczer-, Dombi-, Weber-, Dubois-féle normák.

Elég gyakran alkalmazzák a Lukasiewicz-féle normákat, főleg az algebrai szorzatot (*t-norma*), illetve algebrai összeget (*s-norma*).

A 10.5. táblázatban a Lukasiewicz-féle normák vannak összefoglalva.

10.5. táblázat. Lukasiewicz-féle normák összefoglalása

t-norma	s-norma
korlátos különbség	korlátos összeg
$t_b(\mu(x),\mu(y)) = max\{0,\mu(x)+\mu(y)-1\}$	$s_b(\mu(x),\mu(y)) = min\{1,\mu(x)+\mu(y)\}$
algebrai szorzat	algebrai összeg
$t_a(\mu(x),\mu(y)) = \mu(x)\mu(y)$	$s_a(\mu(x),\mu(y)) = \mu(x)+\mu(y)-\mu(x)\mu(y)$
drasztikus metszet	drasztikus egyesítés
$t_d(\mu(x),\mu(y)) = \begin{cases} min(\mu(x),\mu(y)) & \text{ha } \mu(x)=1 \text{ vagy } \mu(y)=1 \\ 0 & \text{ha } \textit{különben} \end{cases}$	$s_d(\mu(x),\mu(y)) = \begin{cases} max(\mu(x),\mu(y)) & \text{ha } \mu(x)=0 \text{ vagy } \mu(y)=0 \\ 1 & \text{ha } \textit{különben} \end{cases}$

10.1.3. Aggregációs operátorok

Előfordulhat, hogy a t- és s-norma közé eső operátorokra van szükség, ezeket aggregációs vagy kompenzáló operátoroknak hívjuk. Az alábbiakban megemlítjük a lambda (10.10) és gamma (10.11) operátorokat [56].

$$\mu_{A\lambda B}(x) = \lambda\left[\mu_A(x)\,\mu_B(x)\right] + (1-\lambda)\left[\mu_A(x)+\mu_B(x)-\mu_A(x)\,\mu_B(x)\right] \tag{10.10}$$

$\lambda = 0$ esetében az algebrai összeg alapú s-normát, illetve $\lambda = 1$ az algebrai szorzat alapú t-normát kapjuk.

$$\mu_{A\gamma B}(x) = \left[\mu_A(x)\,\mu_B(x)\right]^{1-\gamma}\left[1-(1-\mu_A(x))\,(1-\mu_B(x))\right]^{\gamma} \tag{10.11}$$

$\gamma = 1$ az algebrai összeg alapú s-normát, illetve $\gamma = 0$ az algebrai szorzat alapú t-normát eredményezi.

10.1.4. Klasszikus és fuzzy relációk

10.1.4.1. Klasszikus relációk

Adottak az Y és V halmazok. Az Y és V halmazok Descartes-féle szorzatán, amit $Y \times V$-vel jelölünk, az összes (y, v) párból álló halmazt értjük (10.12), amelyre $y \in Y$ $v \in V$.

$$Y \times V = \{(y,v)\,|y \in Y \textit{és}\ v \in V\}. \tag{10.12}$$

Két halmaz Descartes-féle szorzata több halmazra $X_1, X_2, \cdots, X_k$ a következőképpen terjeszthető ki $X_1 \times X_2 \times \cdots \times X_k$, ahol $k \in N$. A Descartes-féle szorzat az összes rendezett $(x_1, x_2, \cdots, x_k)$ számból áll, ahol $x_1 \in X_1, x_2 \in X_2, \cdots, x_k \in X_k$

Bináris reláció:

Két halmaz, Y és V közötti bináris reláció R, az $Y \times V$ egy részhalmaza. Ha $y \in Y$ és $v \in V$ relációban állnak, a következőképpen jelöljük: $(y, v) \in R$.

Két halmaz közötti bináris reláció (10.13) leírható a karakterisztikus függvénnyel is:

$$\mu_R(y, v) = \begin{cases} 1, & \text{ha} \quad (y, v) \in R \\ 0, & \text{ha} \quad (y, v) \notin R \end{cases} \tag{10.13}$$

Adottak az Y és V halmazok a következő elemekkel $Y = \{2, 4, 6\}$ és $V = \{3, 6, 7\}$, akkor a két halmaz közötti $R = y < v$ a következő:

$$R = \begin{matrix} \\ 2 \\ 4 \\ 6 \end{matrix} \begin{matrix} \begin{bmatrix} 3 & 6 & 7 \end{bmatrix} \\ \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \end{matrix}$$

A következő példában két Y,V halmaz közötti R =y=v egyenlőségi relációt szemléltetjük, ahol $Y = \{3, 6, 7, 9\}$ és $V = \{3, 6, 7, 9\}$. Az eredmény a következő karakterisztikus mátrixszal írható le:

$$R = \begin{matrix} \\ 3 \\ 6 \\ 7 \\ 9 \end{matrix} \begin{matrix} \begin{bmatrix} 3 & 6 & 7 & 9 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{matrix}$$

Több halmaz $X_1, X_2, \cdots, X_k$, ahol $k \in N$ közötti reláció $X_1 \times X_2 \times \cdots \times X_k$ az R egy részhalmaza, és az R-et k-dimenziós relációnak nevezzük.

Amennyiben $x_1 \in X_1, x_2 \in X_2, \cdots, x_k \in X_k$ k-elemből álló számtöbbszörösök relációban állnak, a következőképpen jelöljük: $(x_1, x_2, \cdots, x_k) \in R$.

10.1.4.2. Fuzzy relációk

Adottak az Y és V halmazok.

Az $R = \{((y, v), \mu_R((y, v)) \,|((y, v) \in Y \times V\}$ az $X \times Y \subseteq R$-ben fuzzy relációnak nevezzük. A két halmaz közötti fuzzy reláció (10.14) a következőképpen is felírható [50]:

$$R(y, v) = \left\{ \frac{\mu_R(y, v)}{(y, v)} | ((y, v) \in Y \times V \right\}. \tag{10.14}$$

Két halmaz közötti fuzzy relációt gyakran kétdimenziós tömb formában is szokták szemléltetni [57].

$$R = \begin{matrix} \\ \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \end{matrix} \begin{matrix} \begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix} \\ \begin{bmatrix} \mu_R(y_1, v_1) & \mu_R(y_1, v_2) & \cdots & \mu_R(y_1, v_n) \\ \mu_R(y_2, v_1) & \mu_R(y_2, v_2) & \cdots & \mu_R(y_2, v_n) \\ \vdots & \vdots & \ddots & \vdots \\ \mu_R(y_m, v_1) & \mu_R(y_m, v_2) & \cdots & \mu_R(y_m, v_n) \end{bmatrix} \end{matrix}$$

Példa fuzzy relációra

Adottak az $Y = \{1, 2, 3, 4\}$, illetve $V = \{1, 2, 3, 4\}$ halmazok. Ha az egyes rendezett (y, v) elempárokhoz hozzárendelt tagsági függvényt a következő egyenlet írja le: $\mu_R(y, v) = e^{-(y-v)^2}$, akkor származtatható a két halmaz közötti fuzzy reláció.

$$R = \left\{ \frac{e^{-(1-1)^2}}{(1,1)}, \frac{e^{-(1-2)^2}}{(1,2)}, \frac{e^{-(1-3)^2}}{(1,3)}, \frac{e^{-(1-4)^2}}{(1,4)}, \ldots, \frac{e^{-(3-4)^2}}{(3,4)}, \frac{e^{-(4-4)^2}}{(4,4)} \right\}$$

$$R = \left\{ \frac{1}{(1,1)}, \frac{0{,}368}{(1,2)}, \frac{0{,}018}{(1,3)}, \frac{0{,}0001}{(1,4)}, \ldots, \frac{0{,}368}{(3,4)}, \frac{1}{(4,4)} \right\}$$

$$R = \begin{bmatrix} 1{,}0 & 0{,}368 & 0{,}018 & 0{,}0001 \\ 0{,}368 & 1{,}0 & 0{,}368 & 0{,}018 \\ 0{,}018 & 0{,}368 & 1{,}0 & 0{,}368 \\ 0{,}0001 & 0{,}018 & 0{,}368 & 1{,}0 \end{bmatrix}$$

10.1.4.3. Fuzzy szorzatreláció

Adottak az $X = \{x_1, x_2, \ldots, x_k\}$, $Y = \{y_1, y_2, \ldots, y_m\}$, valamint $Z = \{z_1, z_2, \ldots, z_n\}$ fuzzy halmazokon értelmezett R (10.15), Q (10.16) fuzzy relációk:

$$R = \{((x, y), \mu_R(x, y)) \,|\, x \in X, y \in Y, R \subset X \times Y\} \tag{10.15}$$

$$Q = \{((y, z), \mu_Q(y, z)) \,|\, y \in Y, z \in Z, Q \subset Y \times Z\}. \tag{10.16}$$

A P (10.17) egy reláció, amely az R által tartalmazott X elemek, valamint a Q által tartalmazott Z elemek közötti relációt írja le:

$$P = R \circ_{TS} Q. \tag{10.17}$$

Az R, Q fuzzy relációk $X \times Z$ szorzathalmazon értelmezett $R \circ_{TS} Q$ kompozícióját a következő módon írhatjuk le (10.18):

$$\mu_P(x,z) = \mathbf{S}_{y \in Y} \left\{ \mathbf{T}\left(\mu_R(x,y), \mu_Q(y,z)\right) \right\} \ \forall (x,z) \in X \times Z \tag{10.18}$$

ahol az $\mathbf{S}$, illetve $\mathbf{T}$ általánosan tetszőleges s-normát, illetve t-normát jelöl. Az R, Q, P mátrixalakja a (10.19) szerint alakul:

$$R = (r_{u,v}), Q = (q_{v,w}), P = (p_{u,w}) \tag{10.19}$$

ahol $w = 1 \cdots n$, $u = 1 \cdots k$, $v = 1 \cdots m$.

A P mátrixalakja a következőképpen származtatható (10.20):

$$P_{u,w} = S_v T(r_{u,v}, q_{v,w}) \tag{10.20}$$

Ha $P = R \circ Q$ kompozíciójában *s-normának* max, valamint *t-normának* min normát használunk, akkor $P = R \circ Q$ relációt max-min kompozíciónak nevezzük és a (10.21) szerint definiálhatjuk:

$$\mu_{R \circ Q}(x,z) = \max_{y \in Y} \left\{ min\left(\mu_R(x,y), \mu_Q(y,z)\right) \right\} \ \forall (x,z) \in X \times Z \tag{10.21}$$

Az alábbi példában a $Q(x,y)$és$R(y,z)$ relációk a relációs mátrixokkal vannak megadva.

$$Q = \begin{bmatrix} 0{,}6 & 0{,}5 \\ 1{,}0 & 0{,}4 \\ 0{,}8 & 0{,}7 \end{bmatrix} \text{ és } R = \begin{bmatrix} 0{,}7 & 0{,}9 & 0{,}4 \\ 0{,}5 & 0{,}1 & 0{,}6 \end{bmatrix}$$

$$Q \circ R = \begin{bmatrix} 0{,}6 & 0{,}5 \\ 1{,}0 & 0{,}4 \\ 0{,}8 & 0{,}7 \end{bmatrix} \circ \begin{bmatrix} 0{,}7 & 0{,}9 & 0{,}4 \\ 0{,}5 & 0{,}1 & 0{,}6 \end{bmatrix}$$

$$\mu_{Q \circ R}(x_1, z_1) = \max_{j=1\ldots2} \left\{ min\left(q_{1,j}, r_{j,1}\right) \right\} =$$

$$= \max\left\{\min(0{,}6; 0{,}7), \min(0{,}5; 0.5)\right\} = \max\left\{0{,}6; 0{,}5\right\} = 0{,}6$$

Az (x_1, z_1) elempárhoz hasonlóan a $Q \circ R$ reláció elemei is kiszámolhatóak

$$Q \circ R = \begin{bmatrix} 0{,}6 & 0{,}6 & 0{,}5 \\ 0{,}7 & 0{,}9 & 0{,}4 \\ 0{,}7 & 0{,}8 & 0{,}6 \end{bmatrix}$$

10.1.4.4. Fuzzy relációk vetülete

A projekció és a hengeres kiterjesztés fuzzy relációkon értelmezett műveletek. A fuzzy szabályalapú irányítási rendszerek esetében a projekció és hengeres kiterjesztés fontos szerepet játszanak.

Adott az $R = \{((y,v), \mu_R((y,v)) \,|((y,v) \in Y \times V\}$ az $X \times Y \subseteq R$ fuzzy reláció.

Az R relációnak a vetületét az X univerzumra jelöljük Q-val, míg az Y univerzumra való vetületét P-vel. A két vetületet a (10.22) illetve (10.23) egyenletek írják le:

$$Q = \left\{ \left(x, \max_y \mu_R(x,y) \right) \,|\, (x,y) \in X \times Y \right\} \tag{10.22}$$

$$P = \left\{ \left(\max_x \mu_R(x,y), y \right) \,|\, (x,y) \in X \times Y \right\} \tag{10.23}$$

$$R = \begin{matrix} \\ x_1 \\ x_2 \\ x_3 \end{matrix} \begin{matrix} \begin{bmatrix} y_1 & y_2 & y_3 & y_4 \end{bmatrix} \\ \begin{bmatrix} 0{,}7 & 0{,}6 & 0{,}5 & 0{,}2 \\ 0{,}2 & 0{,}8 & 0{,}1 & 0{,}8 \\ 0{,}5 & 0{,}9 & 0{,}1 & 0{,}2 \end{bmatrix} \end{matrix}$$

Az $R(x,y)$ az X szerint a következőképpen számolható ki:

$\mu_Q(x_1) = \max\{0{,}7; 0{,}6; 0{,}5; 0{,}2\} = 0{,}7$.

Hasonló módon kiszámolva minden elempárra: $\mu_Q(x_2) = \max\{0{,}2; 0{,}8; 0{,}1; 0{,}8\} = 0{,}8$

$\mu_Q(x_2) = \max\{0{,}5; 0{,}9; 0{,}1; 0{,}2\} = 0{,}9$ a következő eredményt kapjuk: $Q = \{(x_1, 0{,}7), (x_2, 0{,}8), (x_3, 0{,}9)\}$.

Hasonló módon kiszámolva az Y szerinti vetületet: $P = \{(0{,}7, y_1), (0{,}9, y_2), (0{,}5, y_3), (0{,}8, y_4)\}$.

10.1.4.5. Fuzzy relációk hengeres kiterjesztése

Egy A fuzzy halmaznak az $X \times Y$-ra való hengeres kiterjesztése az X (10.24), illetve Y (10.24) szerint a következő relációkkal írható le:

$$cyl_x A(x,y) = A(x) \tag{10.24}$$

$$cyl_y A(x,y) = A(y). \tag{10.25}$$

Az előbbi példa alapján a Q, illetve P hengeres kiterjesztése a következő eredményt szolgáltatja:

$$\begin{bmatrix} 0{,}7 & 0{,}7 & 0{,}7 \\ 0{,}8 & 0{,}8 & 0{,}8 \\ 0{,}9 & 0{,}9 & 0{,}9 \end{bmatrix} \text{ illetve } \begin{bmatrix} 0{,}7 & 0{,}9 & 0{,}5 & 0{,}8 \\ 0{,}7 & 0{,}9 & 0{,}5 & 0{,}8 \\ 0{,}7 & 0{,}9 & 0{,}5 & 0{,}8 \end{bmatrix}$$

10.1.4.6. Fuzzy szabályok értelmezése fuzzy relációként

Egy fuzzy szabálybázis természetes nyelvi szabályokkal a (10.26) szerint írható le:

$$HA\ x = A_1\ AKKOR\ y = B_1 \tag{10.26}$$

ahol $x \in X$ bemeneti változó, $y \in Y$ kimeneti változó, X, Y bemeneti, illetve kimeneti változók alaphalmaza vagy univerzuma, Az $A = \{A_1, A_2, \ldots, A_n\}$, illetve $B = \{B_1, B_2, \ldots, B_m\}$ a bemeneti és kimeneti univezumokon meghatározott nyelvi változók, fuzzy halmazok. A fuzzy halmazok a nyelvi változókhoz rendelt tagsági függvényekkel írhatóak le.

A bemeneti $\mu_{A_i}(x)$, illetve kimeneti $\mu_{B_j}(y)$ nyelvi változókhoz rendelt tagsági függvények. Egy rendszerre a fuzzy szabálybázist a rendszer működését leíró összes nyelvi szabály alkotja.

Minden egyes szabálynak van egy precedens része, $HAx = A_i$ és egy konzekvens része. Vagyis egy fuzzy szabályban az előfeltételekhez hozzárendeljük az antecedenshez tartozó kimeneti fuzzy értékeket.

A fuzzy *HA AKKOR* szabályok értelmezhetőek konjunkció-, valamint implikációként is. A gyakorlati alkalmazásokban a konjukciós megoldásokat alkalmazzák. A konjukciós alapú modell a szabályokat adatpárként (A_i, B_j) kezeli és az A_i precedens és B_i konzekvens közötti relációt írja le.

Az R_i fuzzy szabály-reláció $X \times Y$ Descartes-szorzat téren értelmezett fuzzy halmaz (10.27).

$$R_i(x,y) : \mu_{R_i(x,y)}(x,y) = \{((x,y), t(\mu_{A_i}(x), \mu_{B_i}(y))) \mid (x,y) \in X \times Y\} \tag{10.27}$$

ahol t egy választott t-norma.

A gyakorlatban egyszerű alkalmazhatósága miatt általában a Zadeh-féle min normát használják. A következtetés eredményét a szabálybázisban található szabályok egyesítéseként számoljuk ki (10.28).

$$R = \bigcup_{i=1}^{r} R_i \tag{10.28}$$

Az egyesítésre valamelyik s-norma alkalmazható (10.29).

$$R(x,y) : \mu_{R(x,y)}(x,y) = \left\{ \left((x,y), \underset{i=1}{\overset{r}{s}} \left(\mu_{R_i(x,y)}(x,y) \right) \right) \mid (x,y) \in X \times Y \right\} \tag{10.29}$$

A következő szabálybázis esetén:

$R_1 : HA\ x = A_1\ AKKOR\ y = B_1$
$R_2 : HA\ x = A_2\ AKKOR\ y = B_2$
$R_3 : HA\ x = A_3\ AKKOR\ y = B_3$
$R_4 : HA\ x = A_4\ AKKOR\ y = B_4$

$$\mu_{R_1(x,y)}(x,y) = \{((x,y), \min(\mu_{A_1}(x), \mu_{B_1}(y))) \,|\, (x,y) \in X \times Y\} \quad (10.30)$$

$$\mu_{R_2(x,y)}(x,y) = \{((x,y), \min(\mu_{A_2}(x), \mu_{B_2}(y))) \,|\, (x,y) \in X \times Y\} \quad (10.31)$$

$$\mu_{R_3(x,y)}(x,y) = \{((x,y), \min(\mu_{A_3}(x), \mu_{B_3}(y))) \,|\, (x,y) \in X \times Y\} \quad (10.32)$$

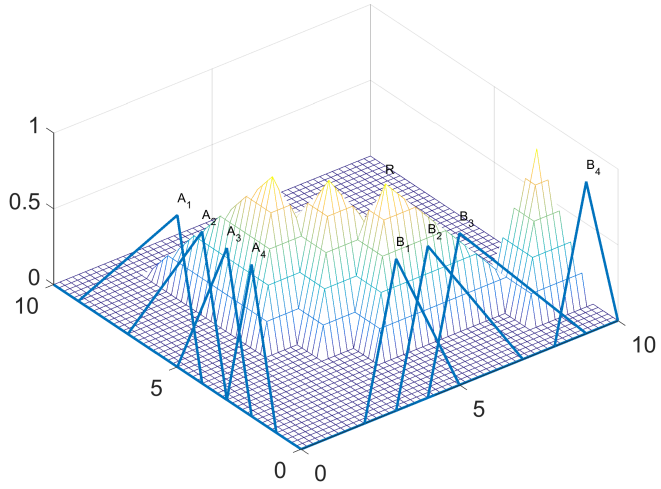
$$\mu_{R_4(x,y)}(x,y) = \{((x,y), \min(\mu_{A_4}(x), \mu_{B_4}(y))) \,|\, (x,y) \in X \times Y\} \quad (10.33)$$

Az egyes szabályoknak megfelelő fuzzy relációk egyesítéséből (10.30), (10.31), (10.32), (10.33) megkapjuk a következtetés eredményét (10.34).

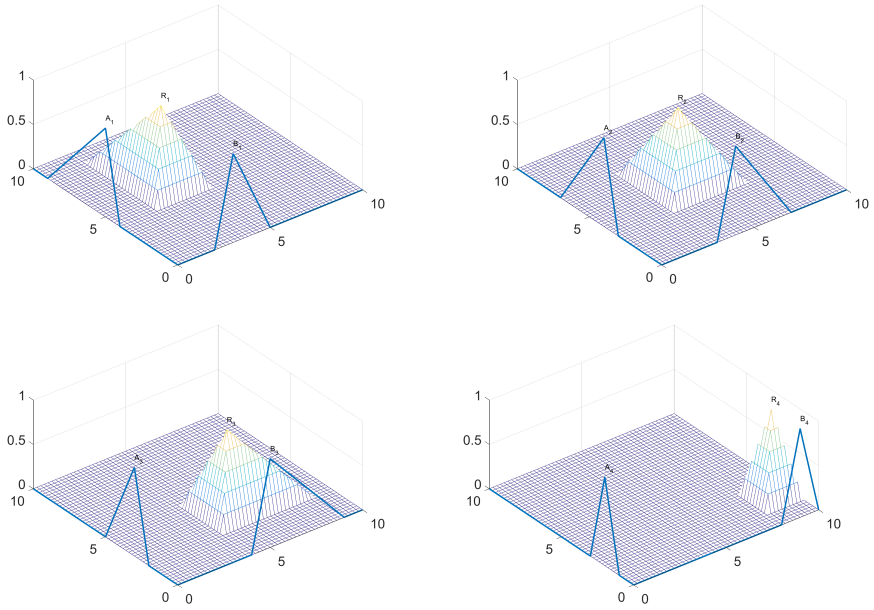
$$R(x,y) : \mu_{R(x,y)}(x,y) =$$

$$\{((x,y), \max\{\mu_{R_1}(x,y), \mu_{R_2}(x,y), \mu_{R_3}(x,y), \mu_{R_4}(x,y)\}) \,|\, (x,y) \in X \times Y\}. \quad (10.34)$$

Grafikusan az eredmény a 10.5., illetve 10.4. ábrán van szemléltetve.



10.4. ábra. Az R reláció, amely tartalmazza az R_1, R_2, R_3 és R_4 relációk egyesítését



10.5. ábra. Fuzzy szabályok relációként való értelmezése

11. fejezet

Fuzzy következtető rendszerek

A fuzzy logikának számos alkalmazási területe van. Az egyik az alkalmazások közül a fuzzy következtető rendszerek.

Az elmúlt néhány évben a mesterséges intelligenciára épülő technikákat alkalmazták arra, hogy az emberi tapasztalatot számítógépek számára érthető formává alakítsák. A fejlett mesterséges intelligenciára épülő irányítási módszereket intelligens irányításnak nevezik. Az intelligens rendszereket általában a biológiai rendszerek analógiájával írják le, megfigyelve, hogy az emberek hogyan végeznek el vezérlési feladatokat, hogyan hoznak döntéseket, hogyan ismernek fel különböző formákat.

Van egy lényeges eltérés az emberek és a gépek közötti gondolkodásmódban. Míg az emberek képesek hiányos, pontatlan vagy akár bizonytalan információk esetében is fuzzy módon gondolkodni, addig a számítógépek bináris logikára épülnek. A fuzzy logikának az alkalmazása az egyik lehetséges módja, hogy a gépeket intelligensebbé tegyük, lehetővé téve számukra, hogy az emberi gondolkodáshoz hasonlóan, fuzzy módon képesek legyenek döntéseket hozni.

A Lotfy Zadeh által 1965-ben javasolt fuzzy logika olyan eszközként jelent meg, amely bizonytalan, pontatlan döntéshozatali problémák kezelésére szolgáltat hatékony megoldást. Az intelligens és hagyományos technikákat ötvöző vezérlők általánosan használatosak az összetett dinamikus rendszer intelligens vezérlésében [58].

11.1. Fuzzy következtető rendszerek szerkezete

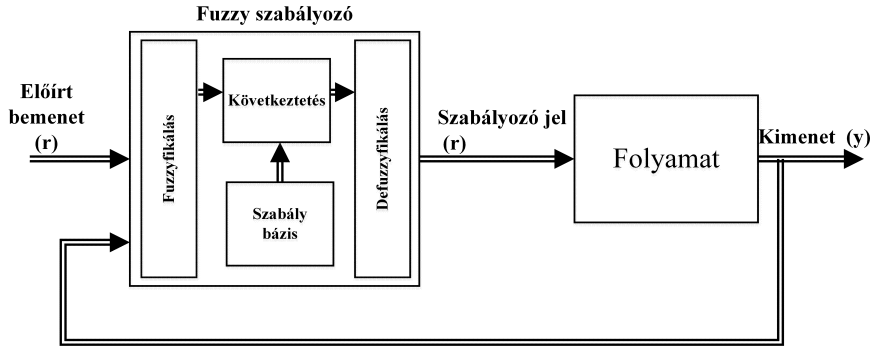
A fuzzy irányítás formális módszert kínál az emberi heurisztikus tudásalapú rendszerek irányításának az értelmezésére és megvalósítására [59].

A fuzzy következtetés alkalmazható olyan esetekben, amikor a folyamatirányítás túl bonyolult a hagyományos módszerek alkalmazása esetében, vagy a rendelkezésre álló információ számszerűen nem értelmezhető, pontatlan vagy bizonytalan. Összehasonlítva a hagyományos módszerekkel, a fuzzy következtető rendszerek számos előnyt szolgáltatnak a bizonytalanság kezelésére.

A fuzzy szabályozásnak az alkalmazása előnyös olyan irányítási feladatoknál, ahol a rendszer matematikai modellje bonyolult vagy nem ismert, valamint ahol a hagyományos irányítási rendszerek alkalmazása nehézkes, illetve drágának bizonyul.

A fuzzy következtető rendszerek számos változatát fejlesztették ki. A leggyakrabban elterjedt fuzzy következtető rendszerek a Mamdani [60], Takagi Sugeno [61] vagy Tsukamoto [62].

A fuzzy következtető rendszer alapját a fuzzy szabálybázis alkotja, amely HA AKKOR szabályokból áll. A szabály HA premissza (vagy antecedens) része az előfeltételt határozza meg, hogy az adott szabály érvényes legyen. Az AKKOR rész a következtetés (konzekvens), és meghatározza, hogy mi történjen abban az esetben, ha az előfeltétel teljesül.



11.1. ábra. Fuzzy következtető rendszer [64]

A fuzzy szabályozó egyszerű felépítésű (11.1. ábra). Egy bemeneti, egy kiértékelő és egy kimeneti szakaszból áll. A bemeneti rész letapogatja az érzékelőket vagy más bemeneteket, és leképezi őket a megfelelő tagsági

függvényekre és igazságértékekre (fuzzy halmazokra). A kiértékelő szakasz meghatározza a szabályok értékeit és azok kombinációit. Végül, a kimeneti szakasz visszaalakítja a fuzzy halmaz kombinációkat *crisp* értékekre. A hagyományos, nem fuzzy halmazokra a szakirodalomban a *crisp* halmaz kifejezést alkalmazzák [63].

Egy fuzzy szabályozó rendszernek négy fő komponensét különböztetjük meg:

- a fuzzyfikáció: ez nem más, mint a bemenet olyan módosítása, hogy a fuzzy rendszer használhassa, a bemeneteket leképezi fuzzy halmazokra;
- fuzzy szabályok adatbázisa: ez a fő komponens, és ha ..., akkor szabályok formájában tartalmazza azokat a fuzzy halmazokra épülő ismereteket, amelyekre a szabályozás logikája épül;
- következtető rendszer: ennek a komponensnek a teljesítményétől is függ a szabályozó rendszer hatékonysága;
- a fuzzy szinten kapott következtetés eredményének a rendszer számára használható alakra való átalakítását a defuzzyfikáló rendszer végzi.

Fuzzyfikáció. A fuzzy logikát a valós világ nyelvének használata teszi igazán hatékony eszközzé. A nyelvi változók alkalmazása kulcsfontosságú a fuzzy szakértői rendszerek és a fuzzy kontrollrendszerek területén. A nyelvi változók értékei szavak, mondatok, vagyis hétköznapi nyelvi elemek lehetnek. A legtöbb valós helyzetben egy precíz válasz nem feltétlenül jelent optimális megoldást a problémára. A fuzzy logika kiterjeszti a szigorú igaz/hamis osztályozást olyan értékekkel, mint például a nagyjából igaz és a teljesen rossz. Ezeket a kifejezéseket nyelvi változóknak, magát a folyamatot fuzzyfikálásnak hívjuk.

A következtető rész egy logikai szabálygyűjteményen alapszik, melynek elemei HA-AKKOR (IF-THEN) mondatok. A HA részt precedensnek és az AKKOR részt következtetésnek nevezik. A szabályozási felhasználásokban az antecedens általában a szabályozott rendszer hibája vagy a hiba változási sebessége. A következtetés egy szabályozó parancskimenet. A fuzzy szabályozási rendszerek tipikusan több tucat szabállyal rendelkeznek. A következtető lehet Mamdani, Takagi-Sugeno, Tsukamoto vagy egyéb típusú.

Defuzzyfikálás A fuzzy szabályok kiértékelésével kapott eredmény, szintén fuzzy halmazok, valós (crisp) értékekké való visszaalakítását nevezzük defuzzyfikálásnak. Különböző eljárások állnak rendelkezésünkre. Mivel egyidejűleg több kifejezés is érvényes lehet, ezért a defuzzyfikálási eljárás feladata, hogy ezek együttes figyelembevételével hozza meg a döntését.

A legáltalánosabban elterjedt defuzzyfikáló eljárások *a maximumok közepe* (Center-of-Maximum – COM), *súlypont módszer* (Center-of-Gravity – COG) a *terület középpont módszer* (Center-of-Area – COA) és a *maximumok átlaga* (Mean-of-Maximum – MOM). A COM módszer esetén a tagsági függvények maximumhelyének súlyozott átlagaként számolja ki a kimeneti értéket, ahol a súlyok a megfelelő szabályok tüzelési értékei. A legelterjedtebb tagsági függvény a háromszög, de a trapéz és haranggörbék is használatosak. A függvény formájánál fontosabb az elhelyezett függvények száma és helyzete. Háromtól hét függvényig általában elegendő a bemeneti vagy kimeneti tartományok lefedéséhez.

A rendszer felépítését, a ki- és bemenetek kiválasztását úgy valósítanánk meg, mint egy jó szakember. A szakember megmondja, hogy milyen bemeneteket használna a rendszer szabályzására. Az első lépésben figyelembe kell venni azt, hogy a rendszerről milyen információ áll a szakember rendelkezésére. Ugyanakkor jó, ha nemcsak a bemenő jel, hanem annak a deriváltja (változási sebessége) is elérhető, alkalmazható.

Szabályozási ismeretek szabályokba ágyazása során a szakember szabályozási ismereteinek lingvisztikus leírását keressük. Ismernünk kell a ki- és bemenetek univerzumát és ezen univerzumoknak megfelelő fuzzy halmazok pontos jelentését. Példa: egy bemenet lehet hőmérséklet, ennek megfelel egy hőmérséklet-tartomány, és ezt az univerzumot lefedjük fuzzy halmazokkal („nagyon alacsony”, „alacsony”, „közepes”, „magas”, „nagyon magas”). Hasonló módon járunk el minden ki- és bemeneti változóval. Nagyon fontos hogy helyesen megbecsüljük a bemenő jelek alsó és felső korlátjait.

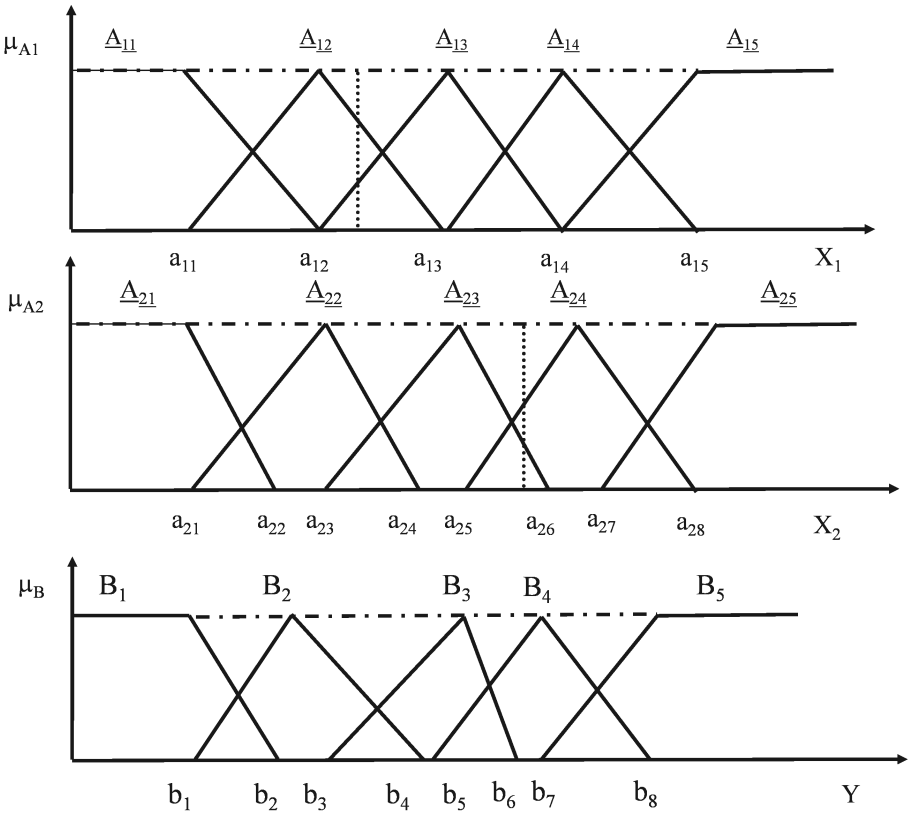
11.1.1. MAMDANI típusú következtető rendszer

A Zadeh-féle irányítási rendszerek fuzzy relációként értelmezik a szabálybázist. A nagy számítási igénye miatt a Mamdani által javasolt módosított változata terjedt el a gyakorlati alkalmazásokban [63].

11.1.2. Fuzzyfikálás

Legyen két bemenetet és egy kimenetet alkalmazó fuzzy szabályozó. Az x_1 és x_2 bemeneti változók az X_1 és X_2 univerzumokon vannak értelmezve $x_1 \in X_1$ és $x_2 \in X_2$, valamint y kimeneti változó az $y \in Y$ univerzumon. Az A_1 és A_2, az X_1 és X_2 univerzumokon értelmezett halmazok $A_1 =$

$\{A_{11}, A_{12}, \cdots, A_{15}\}$, $A_2 = \{A_{21}, A_{22}, \cdots, A_{25}\}$ és $B = \{B_1, B_2, \cdots, B_5\}$ a kimeneti univerzumon (11.2. ábra).



11.2. ábra. Bemeneti és kimeneti univerzumok lefedése tagsági függvényekkel

Az X_1 bemeneten értelmezett fuzzy halmazok:

$$\mu_{A_{11}}(x) = \begin{cases} 1 & \text{ha} \quad x \leq a_{11} \\ \frac{a_{12}-x}{a_{12}-a_{11}} & \text{ha} \quad a_{11} < x < a_{12} \\ 0 & \text{ha} \quad x \geq a_{12} \end{cases}$$

$$\mu_{A_{12}}(x) = \begin{cases} 0 & \text{ha} \quad x \leq a_{11} \\ \frac{x-a_{11}}{a_{12}-a_{11}} & \text{ha} \quad a_{11} < x \leq a_{12} \\ \frac{a_{13}-x}{a_{13}-a_{12}} & \text{ha} \quad a_{12} < x < a_{13} \\ 0 & \text{ha} \quad x \geq a_{13} \end{cases}$$

$$\mu_{A_{13}}(x) = \begin{cases} 0 & \text{ha} \quad x \leq a_{12} \\ \frac{x-a_{12}}{a_{13}-a_{12}} & \text{ha} \quad a_{12} < x \leq a_{13} \\ \frac{a_{14}-x}{a_{14}-a_{13}} & \text{ha} \quad a_{13} < x < a_{14} \\ 0 & \text{ha} \quad x \geq a_{14} \end{cases}$$

$$\mu_{A_{14}}(x) = \begin{cases} 0 & \text{ha} \quad x \leq a_{13} \\ \frac{x-a_{13}}{a_{14}-a_{13}} & \text{ha} \quad a_{13} < x \leq a_{14} \\ \frac{a_{15}-x}{a_{15}-a_{14}} & \text{ha} \quad a_{14} < x < a_{15} \\ 0 & \text{ha} \quad x \geq a_{15} \end{cases}$$

$$\mu_{A_{15}}(x) = \begin{cases} 0 & \text{ha} \quad x \leq a_{14} \\ \frac{x-a_{14}}{a_{15}-a_{14}} & \text{ha} \quad a_{14} < x < a_{15} \\ 1 & \text{ha} \quad x \geq a_{15} \end{cases}$$

Az X_2 bemeneten értelmezett fuzzy halmazok:

$$\mu_{A_{21}}(x) = \begin{cases} 1 & \text{ha} \quad x \leq a_{21} \\ \frac{a_{22}-x}{a_{22}-a_{21}} & \text{ha} \quad a_{21} < x < a_{22} \\ 0 & \text{ha} \quad x \geq a_{22} \end{cases}$$

$$\mu_{A_{22}}(x) = \begin{cases} 0 & \text{ha} \quad x \leq a_{21} \\ \frac{x-a_{21}}{a_{23}-a_{21}} & \text{ha} \quad a_{21} < x \leq a_{23} \\ \frac{a_{24}-x}{a_{24}-a_{23}} & \text{ha} \quad a_{23} < x < a_{24} \\ 0 & \text{ha} \quad x \geq a_{24} \end{cases}$$

$$\mu_{A_{23}}(x) = \begin{cases} 0 & \text{ha} \quad x \leq a_{23} \\ \frac{x-a_{23}}{a_{25}-a_{23}} & \text{ha} \quad a_{23} < x \leq a_{25} \\ \frac{a_{26}-x}{a_{26}-a_{25}} & \text{ha} \quad a_{25} < x < a_{26} \\ 0 & \text{ha} \quad x \geq a_{26} \end{cases}$$

$$\mu_{A_{24}}(x) = \begin{cases} 0 & \text{ha} \quad x \leq a_{25} \\ \frac{x-a_{25}}{a_{27}-a_{25}} & \text{ha} \quad a_{25} < x \leq a_{27} \\ \frac{a_{28}-x}{a_{28}-a_{27}} & \text{ha} \quad a_{27} < x < a_{28} \\ 0 & \text{ha} \quad x \geq a_{28} \end{cases}$$

$$\mu_{A_{25}}(x) = \begin{cases} 0 & \text{ha} \quad x \leq a_{27} \\ \frac{x-a_{27}}{a_{28}-a_{27}} & \text{ha} \quad a_{27} < x < a_{28} \\ 1 & \text{ha} \quad x \geq a_{28} \end{cases}$$

Az Y kimeneti univerzumon értelmezett fuzzy halmazok:

$$\mu_{B_1}(x) = \begin{cases} 1 & \text{ha} \quad x \leq b_1 \\ \frac{b_2-x}{b_2-b_1} & \text{ha} \quad b_1 < x < b_2 \\ 0 & \text{ha} \quad x \geq b_2 \end{cases}$$

$$\mu_{B_2}(x) = \begin{cases} 0 & \text{ha} \quad x \leq b_1 \\ \frac{x-b_1}{b_3-b_1} & \text{ha} \quad b_1 < x \leq b_3 \\ \frac{b_4-x}{b_4-b_3} & \text{ha} \quad b_3 < x < b_4 \\ 0 & \text{ha} \quad x \geq b_4 \end{cases}$$

$$\mu_{B_3}(x) = \begin{cases} 0 & \text{ha} \quad x \leq b_3 \\ \frac{x-b_3}{b_5-b_3} & \text{ha} \quad b_3 < x \leq b_5 \\ \frac{b_6-x}{b_6-b_5} & \text{ha} \quad b_5 < x < b_6 \\ 0 & \text{ha} \quad x \geq b_6 \end{cases}$$

$$\mu_{B_4}(x) = \begin{cases} 0 & \text{ha} \quad x \leq b_5 \\ \frac{x-b_5}{b_7-b_5} & \text{ha} \quad b_5 < x \leq b_7 \\ \frac{b_8-x}{b_8-b_7} & \text{ha} \quad b_{27} < x < b_8 \\ 0 & \text{ha} \quad x \geq b_8 \end{cases}$$

$$\mu_{B_5}(x) = \begin{cases} 0 & \text{ha} \quad x \leq b_7 \\ \frac{x-b_7}{b_8-b_7} & \text{ha} \quad b_7 < x < b_8 \\ 1 & \text{ha} \quad x \geq b_8 \end{cases}$$

11.1.3. Szabálybázis

A szabálybázisból a pillanatnyi bemenetekre az aktív szabályok a 11.1. táblázatban van összefoglalva. Nem kötelező minden kombinációra épülő szabály használata, mert nagyon korlátozza a feldolgozási sebességet. A szabályokat a teljes dinamikájukban kell átgondolni. Nagyon fontos, hogy olyan szabályaink is legyenek, amelyek „beindítják" és „leállítják" a szabályozó rendszert, ugyanakkor a be- és kimeneti jelek időbeni változatosságát is figyelembe kell venni. A rendszer dinamikájának követése benne kell legyen a tudásbázisban. Ez azt jelenti, hogy nem tölthetjük ki sablonosan a szabálytáblázatot.

11.1. táblázat. Szabálytáblázat

A_2 \ A_1	A_{11}	A_{12}	A_{13}	A_{14}	A_{15}
A_{21}					
A_{22}					
A_{23}		B_2	B_4		
A_{24}		B_3	B_3		
A_{25}					

Számoljuk ki a rendszer kimenetelét, ha a két bemenő érték x_1 és x_2. A táblázatnak megfelelően a következő tagsági függvények aktívak a bemeneti (a,b) értékpárra:

- $x_1 \in [a_{11}, a_{13}]$, tehát az A_{12} tagsági függvény adja az $[a_{11}, a_{13}]$ leképezési értéket,
- $x_1 \in [a_{12}, a_{14}]$, tehát az A_{13} tagsági függvény adja az $[a_{12}, a_{14}]$ leképezési értéket,
- $x_2 \in [a_{23}, a_{26}]$, tehát az A_{23} tagsági függvény adja a $[a_{23}, a_{26}]$ leképezési értéket,
- $x_2 \in [a_{25}, a_{28}]$, tehát az A_{24} tagsági függvény adja a $[a_{25}, a_{28}]$ leképezési értéket.

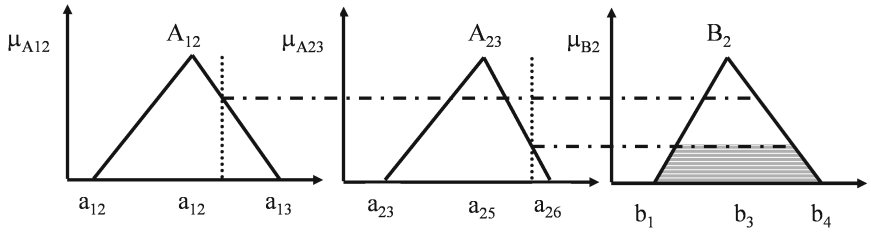
Megfigyelhető, hogy mindkét bemenetre két tagsági függvény aktiválódik, x_1-re az A_{12} és A_{13}, az x_2-re pedig az A_{23} és A_{24}, ami összesen négy szabályt jelent, ezek a szabályok a következőek:
HA $x_1 = A_{12}$ **és** $x_2 = A_{23}$ **AKKOR** $y = B_2$
HA $x_1 = A_{12}$ **és** $x_2 = A_{24}$ **AKKOR** $y = B_3$

HA $x_1 = A_{13}$ **és** $x_2 = A_{23}$ **AKKOR** $y = B_4$
HA $x_1 = A_{13}$ **és** $x_2 = A_{24}$ **AKKOR** $y = B_3$

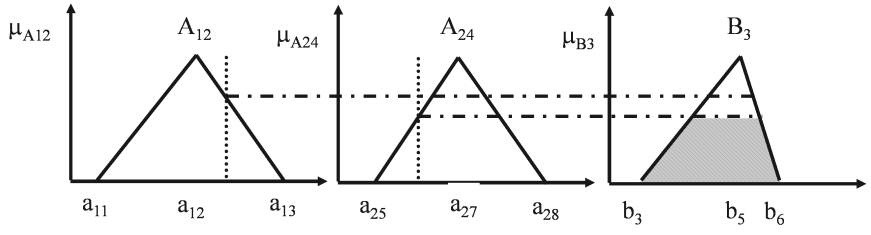
11.1.4. Következtetés

Az alábbi négy rajzon a négy szabály azonosítható. Minden rajzon látható az x_1 és x_2 bemenet, valamint az y kimenet, de minden univerzumról csak egy, a szabálynak megfelelő tagsági függvény van feltüntetve. Amint látható, a szabályokban a két bemenet között és műveletet kell elvégezni. A kapott érték határozza meg, hogy az adott szabály mennyire tüzel, milyen mértékben játszik szerepet a kimeneti értékben. A bemeneti érték alapján mindkét tagsági függvényre kiszámoljuk a tagsági értékeket, $\mu_{A_{12}}$ és $\mu_{A_{23}}$, és a kimeneti univerzumban a szabálynak megfelelő tagsági függvényen elvégezzük a $min(\mu_{A_{12}}, \mu_{A_{23}})$ értékkel egy α-vágást (11.3., 11.4., 11.5., 11.6. ábra). Ezeket a műveleteket megismételjük mind a négy szabályra.

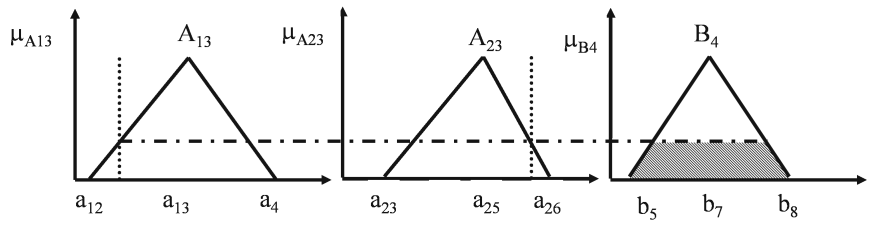
Az α-vágást egy A halmazra a következőképpen definiáljuk:
$A_\alpha = \{x \in X | \mu_A(x) \geq \alpha; \alpha \in (0, 1]\}$



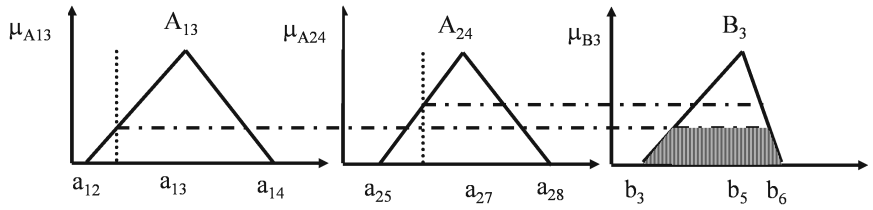
11.3. ábra. Első szabály grafikus értelmezése



11.4. ábra. Második szabály grafikus értelmezése

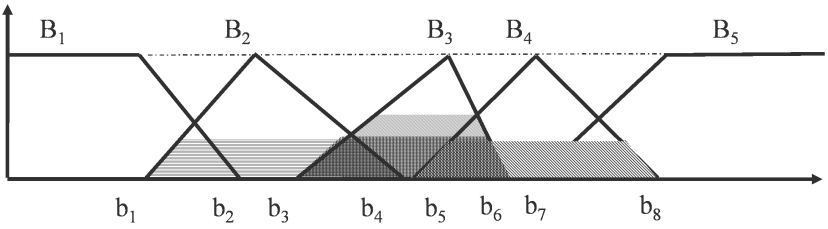


11.5. ábra. Harmadik szabály grafikus értelmezése



11.6. ábra. Negyedik szabály grafikus értelmezése

A következtetés eredményét a kimeneti univerzum tagsági függvényein elvégezett α-vágásból származó részhalmazok egyesítéséből kapjuk, amint az alábbi ábrán látható (11.7. ábra), majd valamelyik típusú defuzzyfikálási algoritmust alkalmazva kiszámoljuk a kimenetet.



11.7. ábra. A kimeneti univerzumból az egyes halmazok egyesítése

11.1.5. Defuzzyfikálás. Defuzzyfikálási módszerek

A fuzzy következtető rendszerek kimenete irányítási gyakorlati alkalmazásoknál crisp numerikus értékek kell legyenek. A MAMDANI-típusú következtető rendszerek eredménye egy fuzzy halmaz. A fuzzy halmazhoz

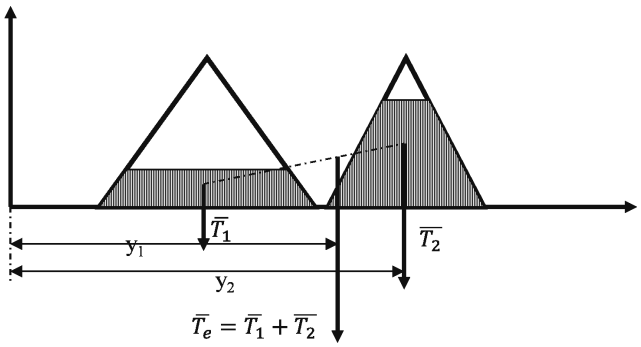
hozzá kell rendelni egy konkrét crisp értéket, amely a halmazt a legjobban jellemzi. Egy konkrét kimeneti érték kiszámításához a kapott fuzzy halmaz kiértékelésére, defuzzyfikálására van szükség. Több deffuzzyfikálási módszer alkalmazható [65], ezek közül ismertetünk egy párat a következő részben.

11.1.5.1. Súlypontmódszer

$$y_i^* = \frac{\int\limits_{y \in supp(B_i)} \mu_{B_i^*}(y)\, y dy}{\int\limits_{y \in supp(B_i)} \mu_{B_i^*}(y)\, dy} \tag{11.1}$$

$$T_i = \int\limits_{y \in supp(B_i)} \mu_{B_i^*}(y)\, dy \tag{11.2}$$

$$y_{COG} = \frac{\sum\limits_{i=1}^{r} T_i y_i}{\sum\limits_{i=1}^{r} T_i} \tag{11.3}$$



11.8. ábra. COG defuzzyfikálás

Általánosan a szabálybázisból egy szabályt a következőképpen írunk fel:
R_i : **HA** $x_1 = A_{1i}$ **és** $x_2 = A_{2i}$ **AKKOR** $y = B_i$
$i = 1 \ldots r$ ahol r a szabálybázisban található szabályoknak a száma.

T normát számolva, a bemeneteknek megfelelő tagsági értékekkel meghatároztuk a szabály erősségét, vagyis azt, hogy milyen mértékben tüzel az

adott szabály, w_i.
$w_i = min\left(\mu_{A_{1i}}, \mu_{A_{2i}}\right)$.

A kimeneti univerzumból, a szabály által megjelölt halmazon, elvégeztük az α-vágást a szabály tüzelési értékével, w_i. Az így kapott fuzzy részhalmazokra (B_i^*) kiszámoljuk a részhalmazt leíró egyenletekre a T_i területet (11.2), illetve az y tengely szerint a súlypontot (11.1). A változókban alkalmazott i indexek a szabálybázisból az i-edik szabálynak megfelelő értékek. A $\mu_{B_i^*}$, vagyis az alfa vágott halmaz a kimeneti univerzumból, a (11.4) egyenlettel írható le:

$$\mu_{B_i^*} = min(w_i, \mu_{B_i}) \tag{11.4}$$

ahol μ_{B_i} az i-edik szabály következtetési részében szereplő halmaz a kimeneti univerzumból (11.8. ábra).

11.1.5.2. Területközéppont-módszer

Egyes szerzők a *geometriai középpont módszer* elnevezést is alkalmazzák [63]. A lényeges különbség a területközéppont-módszer (COA), valamint a súlypontmódszer között (COG), hogy míg a COG esetében a kimenetszámításban az egymásra átfedő halmazok többször megjelennek, addig a COA esetében csak egyszer (11.9. ábra).

$$\mu_B(y) = \max_{i=1}^{r}\left(\mu_{B_i^*}(y)\right) \tag{11.5}$$

ahol $y \in Y$ a $\mu_{B_i^*}(y)$ az i-edik szabály szerint származtatott kimeneti halmaz (B_i^*) tagsági értéke az y pontban.

A területközéppont a (11.6) egyenlet szerint számolható.

$$y_{COA} = \frac{\int\limits_{y\in Y} \mu_{B_i}(y)\, y dy}{\int\limits_{y\in Y} \mu_{B_i}(y)\, dy} \tag{11.6}$$

Diszkrét kimenet esetén $\{y_1, y_2, \cdots, y_k, \cdots\} \in Y$ a területközéppont a (11.7) és (11.7) egyenletek szerint számítható.

$$\mu_B(y_k) = \max_{i=1}^{r}\left(\mu_{B_i^*}(y_k)\right) \tag{11.7}$$

$$y_{COA} = \frac{\sum\limits_{y_i\in Y} \mu_{B_i}(y_i)\, y_i}{\sum\limits_{y_i\in Y} \mu_{B_i}(y_i)} \tag{11.8}$$

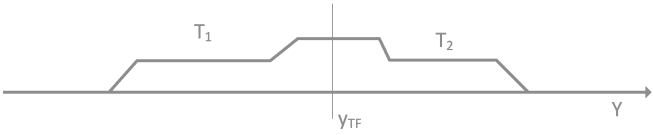
11.9. ábra. COA defuzzyfikálás szemléltetése

11.1.5.3. Területfelezéses módszer

A kimeneti univerzumon a COA defuzzyfikálás során kapott fuzzy halmazt valahol az y tengely mentén kétfelé osztjuk, úgy, hogy a két halmazt határoló tagsági egyenleteket integrálva, a két terület egyforma legyen [56].

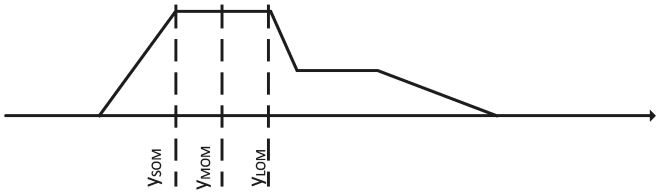
$$\int_{y<y_{TF}}^{y_{TF}} \mu_B(y)\,dy = \int_{y_{TF}}^{y>y_{TF}} \mu_B(y)\,dy \tag{11.9}$$

Az ordináta tengellyel párhuzamos egyenes helyzetét úgy határozzuk meg, hogy az egyenestől balra (T_1) és jobbra (T_2) eső tagsági függvények alatti területek egyformák legyenek (11.10. ábra). Elvileg egy iteratív megoldásra van szükség. Diszkrét kimenet esetén az integrálást el lehet végezni párhuzamosan haladva az Y univerzum alsó és felső határától indulva úgy, hogy az integrálás során a kapott két terület megegyezzen. Mindig azon az oldalon lépünk egy diszkrét értéket, ahol a terület kisebb.



11.10. ábra. Területfelező defuzzyfikálás szemléltetése

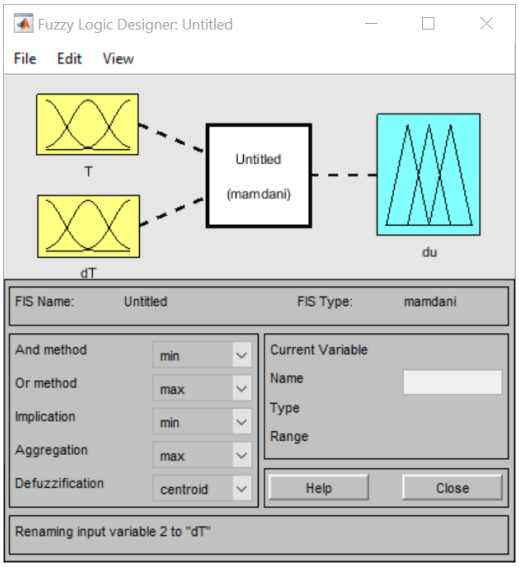
A bemutatott defuzzyfikálási módszereken kívül vannak egyszerűbb módszerek is, mint például legkisebb maximum, középső maximum vagy legnagyobb maximum (11.11. ábra). Mindhárom esetben a kimeneti univerzum halmazain csak a legnagyobb tagsági értékeket vesszük figyelembe. A legkisebb maximum (SOM) esetében a legbaloldalibb maximum pontot, legnagyobb maximum (LOM) esetében a legjobboldalibb, míg a maximumok közepe (MOM) módszer esetében a maximumoknak vesszük a közepét.



11.11. ábra. Maximum defuzzyfikálási eljárások: SOM, MOM és LOM

11.1.6. A Matlab környezetbe integrált vizuális fuzzy tervező

A Fuzzy Logic Toolbox számos függvényt és Simulink tömböt szolgáltat fuzzy logikára épülő rendszerek tervezésére, analizálására és szimulációjára [66], [67]. A Matlab parancsablakában a fuzzy utasítás beírása után megjelenik egy grafikus felület, amely a FIS (Fuzzy Inference System) változóban tárolt fuzzy szabályozó tulajdonságait mutatja:



11.12. ábra. Fuzzy következtető rendszer szerkesztő felülete Matlab környezetben

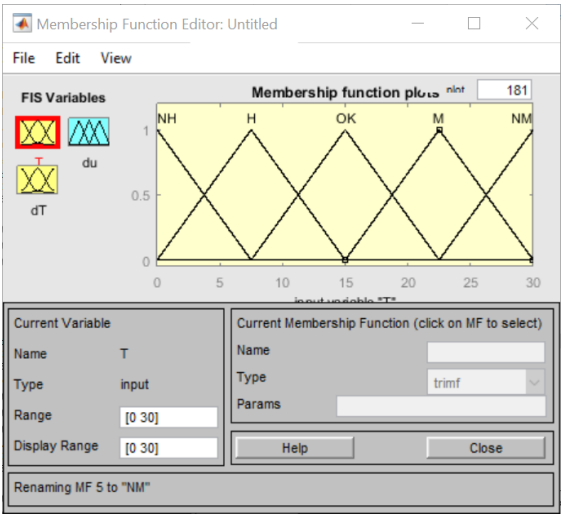
Mint látható, a szabályozónak két bemenete és egy kimenete van (11.12. ábra), az ÉS operátort, valamint az implikációt a *min*, a VAGY operátort és részkonklúziók egyesítését *max* műveletekkel végezzük, a defuzzyfikálást

pedig a területközéppont (*centroid*) eljárással. A File menüben elérhető parancsok segítségével lehet betölteni, illetve elmenteni egy fuzzy szabályozót, mind lemezre, mind a Matlab környezetbe elérhető változóként. Innen kezdhető el egy teljesen új szabályozó szerkesztése is. Az Edit menü alatti utasításokkal adhatunk a szabályozónak további bemeneteket, illetve kimeneteket, és a fenti grafikus felületen kijelölt bemeneti vagy kimeneti változókat eltávolíthatjuk. A View menüpont alatti parancsok lehetőséget nyújtanak:

1. az egyes be/kimeneti változókhoz rendelt tagsági függvények szerkesztésére,
2. a szabálybázis szerkesztésére,
3. a szabályozó viselkedésének grafikus megjelenítésére különböző bemeneti értékekre,
4. a bemenet-kimenet függvény hiperfelületének megjelenítésére.

Az alábbiakban ezek rövid részletezése következik:

11.1.6.1. Tagsági függvények szerkesztése

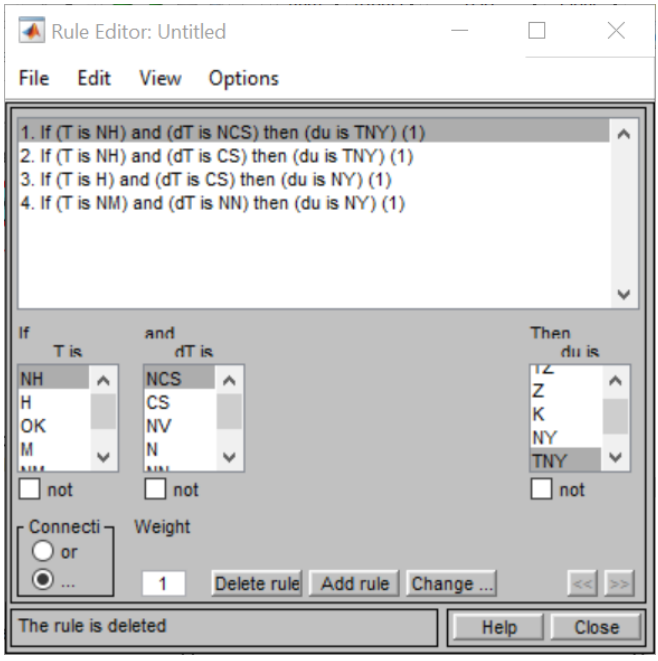


11.13. ábra. Tagsági függvények szerkesztése

Az Edit menüpont parancsai szolgálnak új tagsági függvények hozzárendelésére, illetve tagsági függvények eltávolítására (11.13. ábra). A „FIS Variables” felirat alatt a megfelelő változóra kattintva megjelennek az ahhoz rendelt tagsági függvények, illetve (a „Current variable” ablakrészben) a változó értelmezési tartománya. Egy tagsági függvény grafikonjára kattintva a „Current membership function” ablakrész mutatja annak tulajdonságait. (Pl. „trimf”: háromszög alakú tagsági függvény).

11.1.6.2. Szabálybázis szerkesztése

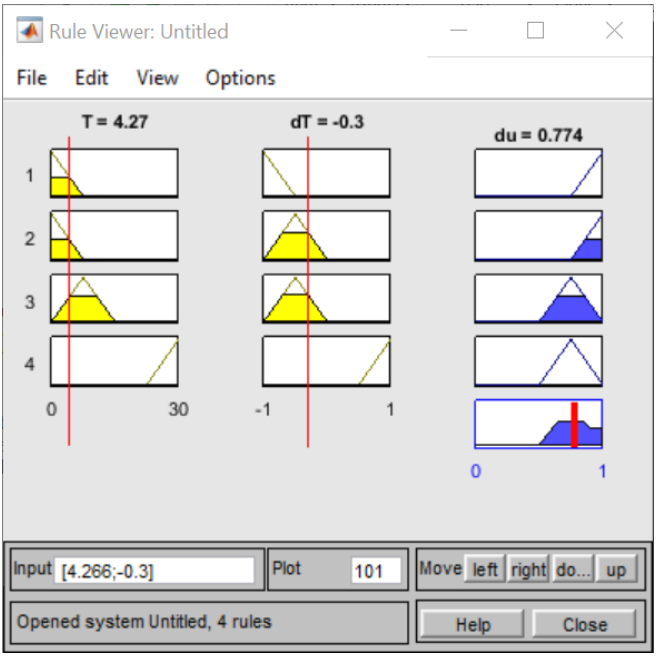
A szabálybázis szerkesztése a 11.14. ábrán van szemléltetve. Egy-egy listából kiválasztható a bemeneti univerzumokból egy-egy halmazpár, majd a kimeneti univerzumból mellérendelhető a megfelelő kimeneti lingvisztikus változó.



11.14. ábra. Szabálybázis szerkesztése

A szerkesztés során, a szabályok egyszerűbb áttekintése végett, javasolt előbb egy táblázatban rögzíteni a szabályokat, majd annak alapján megszerkeszteni a vizuális környezetben a szabálybázist.

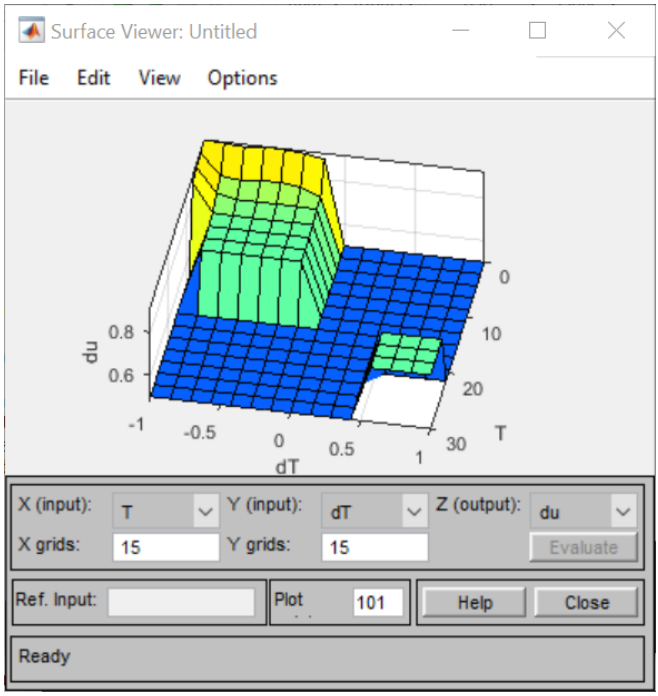
11.1.6.3. Szabályok áttekintése



11.15. ábra. Következtetés áttekintése és értelmezése

Ebben az ablakban az „Input” mezőben megadható bemenetre látható a négy aktív szabály, az egyes szabályok premisszáinak tüzelési értékei, a részkonklúziók, azok egyesítése, végül pedig a kiszámolt kimeneti érték (11.15. ábra). A grafikus felületen az egyes bemenetek csökkenthetők vagy növelhetők, értékelhető, hogy a pillanatnyi bemeneti értékekre milyen mértékben aktiválódnak az egyes halmazok a kimeneti univerzumból. Ez a felület is lehetőséget biztosít a következtető rendszer működésének megértésére és tanulmányozására.

11.1.6.4. Szabályozási felület megjelenítése



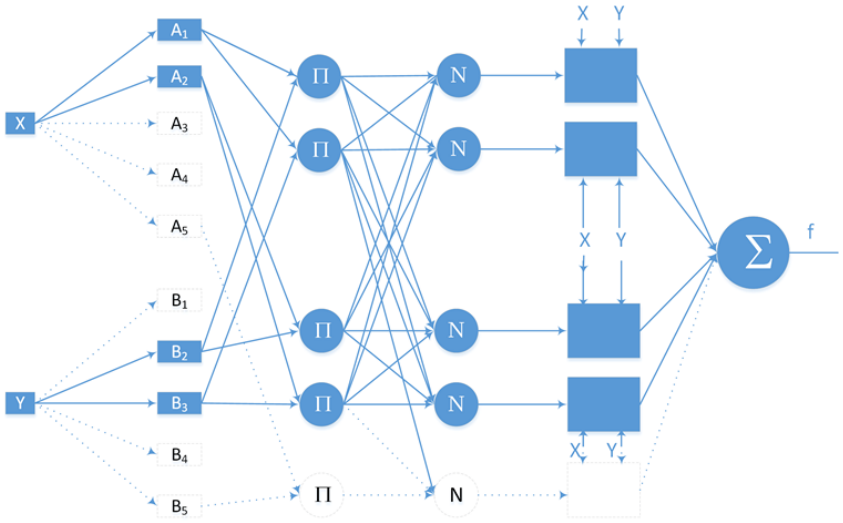
11.16. ábra. Szabályozási felület tanulmányozása

Itt látható a szabályozó kimenete a bemeneteinek függvényében. A felület formájából következtetni lehet arra, hogy a megírt szabályok helyesek-e vagy nem (11.16. ábra). A hangolás során módosítva a szabálybázist, a bemeneti vagy a kimeneti univerzumokon értelmezett tagsági függvények paramétereit, értékelni lehet a beavatkozás hatását a szabályozási felületre.

11.2. Sugeno fuzzy modellre épülő ANFIS szerkezete

A következőkben egy Takagi-Sugeno fuzzy következtető rendszer kerül bemutatásra, amely az alapja az adaptív neuro fuzzy következtető rendszernek (ANFIS). A Sugeno-modell struktúrája két előfeltételből és két

következtető rétegből áll, melyeket egy szabályhalmaz köt össze (11.17. ábra). A Sugeno-modell esetében öt réteget különböztetünk meg, amelyek egy többrétegű hálót alkotnak [68], [69]. A két bemenetet és egy kimenetet tartalmazó Sugeno típusú fuzzy következtető rendszer az alábbi 11.17. ábrán van szemléltetve. Az alábbi rajzon mindkét bemeneti univerzum öt-öt tagsági függvénnyel van lefedve. A lehetséges aktív szabályok száma abban az esetben, ha csak az egymással szomszédos tagsági függvények között van átfedés, elvileg maximálisan négy.



11.17. ábra. Sugeno fuzzy modell

A szemléltetett rajz alapján az aktív szabályokat a következő formában írhatjuk fel:

$Ha\ x_1 = A_1\ és\ x_2 = B_2\ akkor\ f_{1,2}(x_1, x_2) = p_{1,2}x_1 + q_{1,2}x_2 + r_{1,2}$
$Ha\ x_1 = A_1\ és\ x_2 = B_3\ akkor\ f_{1,3}(x_1, x_2) = p_{1,3}x_1 + q_{1,3}x_2 + r_{1,3}$
$Ha\ x_1 = A_2\ és\ x_2 = B_2\ akkor\ f_{2,2}(x_1, x_2) = p_{2,2}x_1 + q_{2,2}x_2 + r_{2,2}$
$Ha\ x_1 = A_2\ és\ x_2 = B_3\ akkor\ f_{2,3}(x_1, x_2) = p_{2,3}x_1 + q_{2,3}x_2 + r_{2,3}$

Általánosan egy szabály a (11.10) formában írható fel.

$$Ha\ x_1 = A_i\ és\ x_2 = B_j\ akkor\ f_{i,j} = p_{i,j}x_1 + q_{i,j}x_2 + r_{i,j} \tag{11.10}$$

11.2. táblázat. Takagi–Sugeno-paraméterhalmaz

X_2 \ X_1	A_1	A_2	A_3	A_4	A_5
B_1	$p_{1,1}; q_{1,1}; r_{1,1}$	$p_{2,1}; q_{2,1}; r_{5,1}$	...	...	$p_{5,1}; q_{5,1}; r_{2,1}$
B_2	$p_{1,2}; q_{1,2}; r_{1,2}$	$p_{2,1}; q_{2,1}; r_{2,1}$	...	...	$p_{5,1}; q_{5,1}; r_{5,1}$
B_3	...	...	...	...	...
B_4	...	...	...	...	...
B_5	$p_{1,5}; q_{1,5}; r_{1,5}$	$p_{2,5}; q_{2,5}; r_{2,5}$	...	...	$p_{5,5}; q_{5,5}; r_{2,1}$

Az A_i és B_i fuzzy halmazok, míg az $p_{i,j}$, $q_{i,j}$ és $r_{i,j}$ paraméterek, amelyek értékeit a fuzzy következtető rendszer hangolása során, ANFIS esetében pedig a tanulási fázis során határozzuk meg. Az ANFIS lényegében a Takagi–Sugeno-modellből van származtatva, felépítésük megegyezik, csak az ANFIS esetében a paraméterek adaptívan vannak meghatározva.

A szakirodalomban az f függvényt, valamint a p, q, r paramétereket csak egy indexszel szokták jelölni. Az ötletet, amely arra késztetett, hogy két indexet alkalmazzak, a MAMDANI típusú fuzzy következtető rendszernél alkalmazott szabálytáblázat alkalmazásából merítettem. A szabályhalmaz egyszerűen felírható és áttekinthető egy szabálytáblázat formájában, és hasonlóan az f függvény paramétereit is egy táblázatba rendezhetjük.

Általánosan felírható $f_{i,j} = p_{i,j}x_1 + q_{i,j}x_2 + r_{i,j}$, ahol az i jelöli az X_1 bemeneti univerzumon értelmezett tagsági függvény indexét, míg a j jelöli az X_2 bemeneti univerzumon a tagsági függvény indexét. A javasolt jelölés, és a paramétereknek táblázatban való rendszerezése egyszerűen alkalmazható FPGA áramkörben való hardveres megvalósítás során, a paraméterek BRAM memóriában való tárolását alkalmazva.

Az egyes rétegeken belül található csomópontokat ugyanaz a függvény írja le. Az l-edik rétegben található k-adik csomópontot $O_{l,k}$-val jelöljük.
Első réteg: adaptív neuronokból áll, amelyek az előfeltétel, (precedens), tagsági függvényeket tartalmazzák, és kimeneteik a tagsági függvények tüzelési, aktivációs fokait térítik vissza (11.11), (11.12).

$$O_{1,k} = \mu_{A_k}(x_1), k = 1 \dots 5 \tag{11.11}$$

$$O_{1,k} = \mu_{B_j}(x_2), k = 5 + j, j = 1 \dots 5. \tag{11.12}$$

Tagsági függvényként a fuzzy következtető rendszerek esetében alkalmazott tagsági függvények közül bármelyik alkalmazható. Haranggörbe alkalmazása esetén a tagsági függvény a (11.13) (11.13) képletekkel írható le, melynek paraméterei a_i, b_i és c_i és előfeltétel paraméterekként említjük:

$$\mu_{A_i}(x_1) = \frac{1}{1 + |\frac{x_1 - c_i}{a_i}|^{2b_i}}, i = 1 \ldots 5 \tag{11.13}$$

$$\mu_{B_j}(x_2) = \frac{1}{1 + |\frac{x_2 - c_j}{a_j}|^{2b_j}}, j = 1 \ldots 5. \tag{11.14}$$

Második réteg: minden csomópont ebben a rétegben rögzített és t-normával képezi a bemenetek eredő aktivációs fokát. Bármilyen t-normát megvalósító függvény is alkalmazható (11.15).

$$O_{2,k} = w_k = t\left(\mu_{A_i}(x_1), \mu_{B_j}(x_2)\right), k = 5 * (j-1) + i, i = 1 \ldots 5, j = 1 \ldots 5 \tag{11.15}$$

Gyakran alkalmazzák normaként az algebrai szorzatot (11.16).

$$O_{2,k} = w_k = \mu_{A_i}(x_1)\, \mu_{B_j}(x_2), k = 5 * (j-1) + i, i = 1..5, j = 1..5 \tag{11.16}$$

A harmadik réteg rögzített neuronjai képezik a normalizált aktivizációs fokokat (11.17):

$$O_{3,k} = \overline{w}_k = \frac{w_k}{\sum_{i=1}^{25} w_i} \tag{11.17}$$

A normalizálás során az aktivációs fokot minden csomópontra elosztjuk a rétegbeli csomópontok aktivációs fokának összegével.

A 11.17. ábra szerint az x_1 bemenetre az A_1 ésA_2, az x_2 bemenetre pedig a B_2 és B_3 tagsági függvények aktívak. A második rétegben tehát a k=6, k=7, k=11 és k=12 csomópontokra az aktivációs fok különbözik nullától. $O_{3,k}$, ha csak a nullától különböző aktivációs fokot eredményező csomópontokra végezzük el a számítást (11.18):

$$O_{3,k} = \overline{w}_k = \frac{w_k}{w_6 + w_7 + w_{10} + w_{11}} \tag{11.18}$$

Negyedik réteg. Minden csomópont ebben a rétegben adaptív a következő csomópont-függvénnyel (11.19):

$$O_{4,k} = \overline{w}_k f_k(x_1, x_2) = \overline{w}_k (p_k x_1 + q_k x_2 + r_k) = \overline{w}_k (p_{i,j} x + q_{i,j} y + r_{i,j}), \tag{11.19}$$

ahol $k = 5 * (j-1) + i$, $i = 1 \dots 5$, $j = 1 \dots 5$ $\overline{w}_k$ a k-adik csomópontnak a normalizált aktivációs foka a harmadik rétegből, míg a p_k, q_k r_k a negyedik réteg paraméterei.

Ez a réteg a rendszer adaptív csomópontjai (p, q, r hangolható paraméterek), a függvényeket szorozzák a szabály előfeltételében származó normalizált aktivációs fokokkal.

Ötödik réteg. Egyetlen fix csomópontot tartalmaz, az eredő kimenetet képezi, összegezve a negyedik réteg kimeneteit (11.20)

$$O_{5,k} = \sum_k \overline{w}_k f_k = \frac{\sum\limits_k w_k f_k}{\sum\limits_k w_k}. \tag{11.20}$$

A 11.17. ábra szerint a harmadik rétegből csak a nullától különböző aktivációs fokkal rendelkező csomópontokat vesszük figyelembe, a rendszer-eredő kimenete a (11.21) formában írható fel [63]:

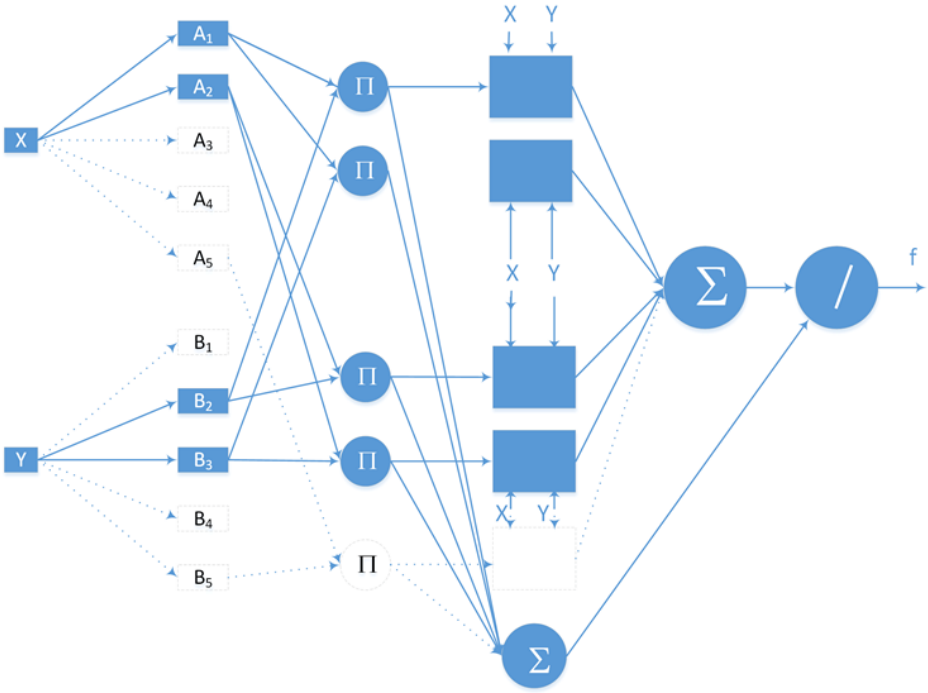
$$O_{5,k} = \sum k\overline{w}_k f_k = \frac{w_6 f_6 + w_7 f_7 + w_{10} f_{10} + w_{11} f_{11}}{w_6 + w_7 + w_{10} + w_{11}}. \tag{11.21}$$

A Sugeno-modell hardveres megvalósítása a 14. fejezetben van részletezve.

11.3. Módosított szerkezetű Sugeno fuzzy modell

Hardveres megvalósításnál előnyösebb, ha az aktivációs fokok normalizálását, tehát a harmadik rétegben elvégzett számításokat csak az összegzés után végezzük el. Ebben az esetben akár egész számon való ábrázolással is kódolhatjuk a rendszerben a különböző jeleket, illetve paramétereket. Többek közt a pipeline architektúra kialakításában is szinkronizációs szempontból egyszerűsödnek a dolgok, mivel az osztást nem tudjuk egy órajel alatt elvégezni, egyszerűbb, ha az osztási műveletet a pipeline rendszer legvégére helyezzük [70].

Ha kivágjuk a harmadik réteget és a súlytényezők normalizálását a legutolsó rétegbe tesszük a 11.18. ábrán szemléltetett Sugeno-modell alapú fuzzy következtető rendszert kapjuk.



11.18. ábra. A Takagi–Sugeno-modell, normalizálás az utolsó rétegben

Ebben az esetben a különböző rétegekben a csomópontokat leíró függvények a (11.22) alapján alakulnak:

$$O_{1,k} = \mu_{A_k}(x_1), k = 1 \dots 5 \tag{11.22}$$

$$O_{1,k} = \mu_{B_j}(x_2), k = 5 + j, j = 1 \dots 5. \tag{11.23}$$

Második réteg kimenete (11.24):

$$O_{2,k} = w_k = \mu_{A_i}(x_1)\,\mu_{B_j}(x_2), k = 5 * (j-1) + i,\ i = 1 \dots 5,\ j = 1 \dots 5. \tag{11.24}$$

Harmadik réteg kimenete (11.25):

$$O_{3,k} = w_k f_k(x_1, x_2) = w_k(p_k x_1 + q_k x_2 + r_k) = w_k(p_{i,j} x + q_{i,j} y + r_{i,j}), \tag{11.25}$$

ahol $k = 5 * (j-1) + i$, $i = 1 \dots 5$, $j = 1 \dots 5$. Negyedik réteg kimenete (11.26):

$$O_{4,k} = \sum_k w_k f_k = \sum_k w_k f_k. \tag{11.26}$$

Ha csak az aktív csomópontokat vesszük figyelembe, a negyedik réteg kimenete a (11.27) szerint alakul:

$$O_{4,k} = w_6 f_6 + w_7 f_7 + w_{10} f_{10} + w_{11} f_{11}. \tag{11.27}$$

Az ötödik rétegben eredő kimenetként megkapjuk az első változat szerinti eredményt (11.28):

$$O_{5,k} = \frac{\sum\limits_k w_k f_k}{\sum\limits_k w_k}. \tag{11.28}$$

11.4. Mamdani következtető rendszer alkalmazása

Tervezzünk egy Mamdani típusú fuzzy szabályozót, amely egy adott teremben a hőmérsékletet az előírt értékre szabályozza. Fűtési szezonban alkalmazzuk a rendszert, amikor a kinti hőmérséklet jóval alacsonyabb. A rendszerbe a hőcserélőkre szerelt, elektronikusan vezérelhető termosztátszeleppel avatkozhatunk be. A termosztátszelepet egy impulzusszélességben modulált (PWM) jellel vezérelhetjük. Ha a PWM jel kitöltési tényezője 0%, akkor a szelep zárva, 100% kitöltési tényező mellett a szelep teljesen nyitva. A fuzzy szabályozónak két bemenetet alkalmazunk, a hőmérsékletet, valamint a hőmérséklet változását. A hőmérséklet 0 és 30 °C között változik, és az alkalmazott hőmérő is ebben a tartományban képes érzékelni. A rendszer mintavételezési periódusát a rendszer időállandójának függvényében választjuk meg, úgy, hogy időállandónyi idő alatt többször felülmintavételezhessük a rendszert. A feladatban percenként a hőmérséklet legfeljebb 1 °C fokot változhat. A két bemeneti és kimeneti univerzumok a következő intervallumban változhatnak:

$T \in [0 \dots 30]\,°\mathrm{C}$, $\Delta T \in [-1 \cdots 1]\,°\mathrm{C/perc}$, valamint a kimenet $u \in [0 \cdots 100]\,\%$.

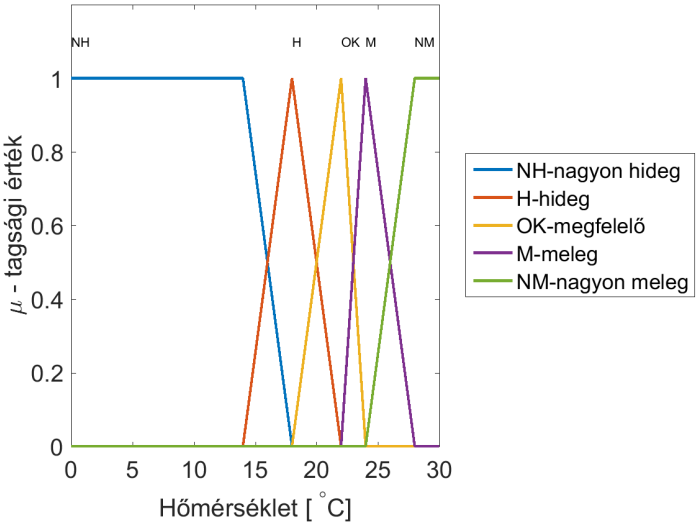
Tervezzünk egy Mamdani típusú fuzzy következtető rendszert, amely a hőmérsékletet az előírt 22 °C-ra szabályozza. A feladat megoldásánál kövessük a következő lépéseket:

– a bemeneti és kimeneti univerzumokat fedjük le fuzzy halmazokkal, és írjuk fel az egyes tagsági függvények egyenletét,

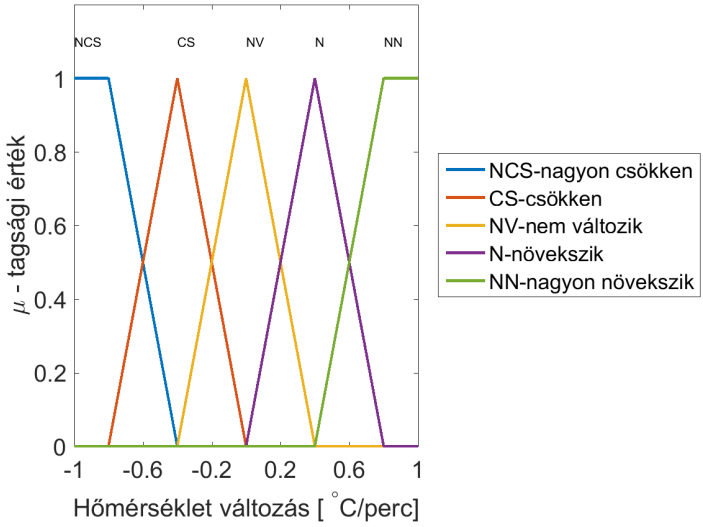
– két adott bemeneti értékre: $T = 17,5\,°\mathrm{C}$ és $\Delta T = 0,3\,°\mathrm{C}/$ perc, számoljuk ki az aktív tagsági függvények tagsági értékeit,
– írjuk fel a szabálybázist legalább az aktív tagsági értékekre,
– végezzük el a szabályoknak megfelelő alfa-vágást a kimeneti univerzum tagsági függvényein,
– különböző kiértékelő (defuzzyfikálási) módszereket (COG, COA) alkalmazva számoljuk ki a következtetés eredményét.

11.4.1. A bemeneti-kimeneti univerzumok lefedése tagsági függvényekkel

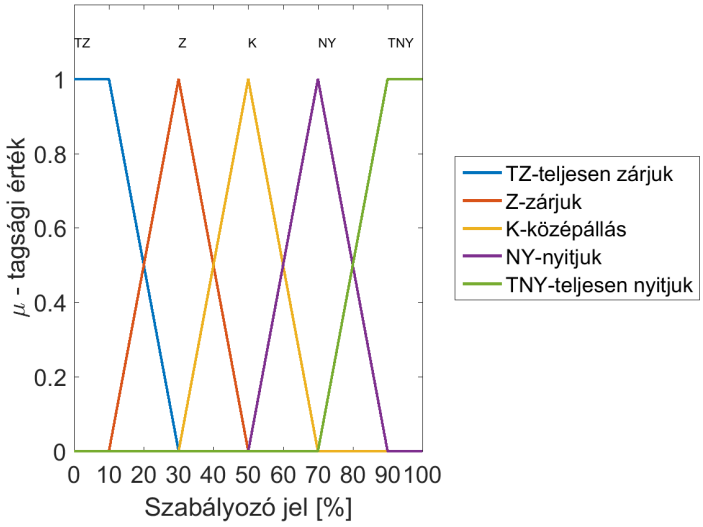
A bemeneti (11.19. ábra), (11.20. ábra) és kimeneti (11.21) univerzumok a következőképpen vannak lefedve tagsági függvényekkel:



11.19. ábra. Hőmérséklet univerzum lefedése tagsági függvényekkel



11.20. ábra. Hőmérséklet változása univerzum lefedése tagsági függvényekkel



11.21. ábra. Kimeneti univerzum lefedése tagsági függvényekkel

11.3. táblázat. Szabálybázis

ΔT \ T	NH	H	OK	M	NM
NCS	TNY	TNY	NY	K	Z
CS	TNY	NY	NY	K	Z
NV	TNY	NY	K	Z	TZ
N	NY	K	Z	Z	TZ
NN	NY	K	Z	TZ	TZ

Az egyes halmazok esetében alkalmazott nyelvi változók: Hőmérséklet univerzumon értelmezett tagsági függvények:

- NH – nagyon hidegn
- H – hideg
- OK – megfelelő hőmérséklet
- M – meleg
- NN – nagyon meleg

Hőmérséklet-változás univerzumon értelmezett tagsági függvények:

- NCS – nagyon csökken a hőmérséklet
- CS – csökken a hőmérséklet
- NV – nem változik a hőmérséklet
- N – növekszik a hőmérséklet
- NN – nagyon növekszik a hőmérséklet

A kimeneti univerzumon értelmezett nyelvi változók:

- TZ – teljesen zárjuk a szelepet
- Z – zárjuk a szelepet
- K – középállásra állítjuk a szelepet
- NY – nyitjuk a szelepet
- TNY – teljesen nyitjuk a szelepet

11.4.2. Szabálybázis

Hőmérséklet bemeneti univerzumra a tagsági függvények egyenletei

$$\mu_{NH}(x) = \begin{cases} 1 & \text{ha} \quad x \geq 14 \\ \frac{18-x}{18-14} & \text{ha} \quad 14 < x < 18 \\ 0 & \text{ha} \quad x \geq 18 \end{cases}$$

$$\mu_H(x) = \begin{cases} 0 \quad \text{ha} \quad x \leq 14 \\ \frac{x-14}{18-14} \quad \text{ha} \quad 14 < x \leq 18 \\ \frac{22-x}{22-18} \quad \text{ha} \quad 18 < x < 22 \\ 0 \quad \text{ha} \quad x \geq 22 \end{cases}$$

$$\mu_{OK}(x) = \begin{cases} 0 \quad \text{ha} \quad x \leq 18 \\ \frac{x-18}{22-18} \quad \text{ha} \quad 18 < x \leq 22 \\ \frac{24-x}{24-22} \quad \text{ha} \quad 22 < x < 24 \\ 0 \quad \text{ha} \quad x \geq 24 \end{cases}$$

$$\mu_M(x) = \begin{cases} 0 \quad \text{ha} \quad x \leq 22 \\ \frac{x-22}{24-22} \quad \text{ha} \quad 22 < x \leq 24 \\ \frac{28-x}{28-24} \quad \text{ha} \quad 24 < x < 28 \\ 0 \quad \text{ha} \quad x \geq 28 \end{cases}$$

$$\mu_{NM}(x) = \begin{cases} 0 \quad \text{ha} \quad x < 24 \\ \frac{x-24}{28-24} \quad \text{ha} \quad 24 < x < 28 \\ 1 \quad \text{ha} \quad x \geq 28 \end{cases}$$

A hőmérséklet-változásra értelmezett fuzzy halmazok:

$$\mu_{NCS}(x) = \begin{cases} 1 \quad \text{ha} \quad x \geq -0.8 \\ \frac{-0.4-x}{-0.4--0.8} \quad \text{ha} \quad -0.8 < x < -0.4 \\ 0 \quad \text{ha} \quad x \geq -0.4 \end{cases}$$

$$\mu_{CS}(x) = \begin{cases} 0 \quad \text{ha} \quad x \leq -0.8 \\ \frac{x--0.8}{-0.4--0.8} \quad \text{ha} \quad -0.8 < x \leq -0.4 \\ \frac{0-x}{0--0.4} \quad \text{ha} \quad -0.4 < x < 0 \\ 0 \quad \text{ha} \quad x \geq 0 \end{cases}$$

$$\mu_{NV}(x) = \begin{cases} 0 \quad \text{ha} \quad x \leq -0.4 \\ \frac{x--0.4}{0--0.4} \quad \text{ha} \quad -0.4 < x \leq 0 \\ \frac{0.4-x}{0.4-0} \quad \text{ha} \quad 0 < x < 0.4 \\ 0 \quad \text{ha} \quad x \geq 0.4 \end{cases}$$

$$\mu_N(x) = \begin{cases} 0 \quad \text{ha} \quad x \leq 0 \\ \frac{x-0}{0.4-0} \quad \text{ha} \quad 0 < x \leq 0.4 \\ \frac{0.8-x}{0.8-0.4} \quad \text{ha} \quad 0.4 < x < 0.8 \\ 0 \quad \text{ha} \quad x \geq 0.8 \end{cases}$$

$$\mu_{NN}(x) = \begin{cases} 0 & \text{ha} \quad x < 0.4 \\ \frac{x-0.4}{0.8-0.4} & \text{ha} \quad 0.4 < x < 0.8 \\ 1 & \text{ha} \quad x \geq 0.8 \end{cases}$$

Az U kimeneti univerzumon értelmezett fuzzy halmazok:

$$\mu_{TZ}(u) = \begin{cases} 1 & \text{ha} \quad u \geq 10 \\ \frac{30-u}{30-10} & \text{ha} \quad 10 < u < 30 \\ 0 & \text{ha} \quad u \geq 30 \end{cases}$$

$$\mu_{Z}(u) = \begin{cases} 0 & \text{ha} \quad u \leq 10 \\ \frac{u-10}{30-10} & \text{ha} \quad 10 < u \leq 30 \\ \frac{50-u}{50-30} & \text{ha} \quad 30 < u < 50 \\ 0 & \text{ha} \quad u \geq 50 \end{cases}$$

$$\mu_{K}(u) = \begin{cases} 0 & \text{ha} \quad u \leq 30 \\ \frac{u-30}{50-30} & \text{ha} \quad 30 < u \leq 50 \\ \frac{70-u}{70-50} & \text{ha} \quad 50 < u < 70 \\ 0 & \text{ha} \quad u \geq 70 \end{cases}$$

$$\mu_{NY}(u) = \begin{cases} 0 & \text{ha} \quad u \leq 50 \\ \frac{u-50}{70-50} & \text{ha} \quad 50 < u \leq 70 \\ \frac{90-x}{90-70} & \text{ha} \quad 70 < u < 90 \\ 0 & \text{ha} \quad u \geq 90 \end{cases}$$

$$\mu_{TNY}(u) = \begin{cases} 0 & \text{ha} \quad u < 70 \\ \frac{u-70}{90-70} & \text{ha} \quad 70 < u < 90 \\ 1 & \text{ha} \quad x \geq 90 \end{cases}$$

Az aktív szabályok, amelyre a szabály tüzelése nullánál nagyobb:

R_1 : **HA** $x_1 = NH$ **és** $x_2 = NV$ **AKKOR** $y = TNY$
R_2 : **HA** $x_1 = NH$ **és** $x_2 = N$ **AKKOR** $y = NY$
R_3 : **HA** $x_1 = H$ **és** $x_2 = NV$ **AKKOR** $y = NY$
R_4 : **HA** $x_1 = H$ **és** $x_2 = N$ **AKKOR** $y = K$

Az egyes szabályok tüzelési értékét a szabály előfeltételében szereplő nyelvi változók, fuzzy halmazok tagsági értékei közötti t-normaként határozzuk meg (11.29), (11.30), (11.31), (11.32). A tagsági értékeket a pillanatnyi bemenetekre számoljuk.

$$w_1 = t\left(\mu_{NH}\left(x_1\right), \mu_{NV}\left(x_2\right)\right) \tag{11.29}$$

$$w_2 = t\left(\mu_{NH}\left(x_1\right), \mu_N\left(x_2\right)\right) \tag{11.30}$$

$$w_3 = t\left(\mu_H\left(x_1\right), \mu_{NV}\left(x_2\right)\right) \tag{11.31}$$

$$w_4 = t\left(\mu_H\left(x_1\right), \mu_N\left(x_2\right)\right) \tag{11.32}$$

ahol x_1 a hőmérséklet, x_2 pedig a hőmérséklet-változás aktuális értéke.

A t-normára a Zadeh-féle minimumot alkalmazva (11.33), (11.34), (11.35), (11.36):

$$w_1 = \min\left(\mu_{NH}\left(x_1\right), \mu_{NV}\left(x_2\right)\right) \tag{11.33}$$

$$w_2 = \min\left(\mu_{NH}\left(x_1\right), \mu_N\left(x_2\right)\right) \tag{11.34}$$

$$w_3 = \min\left(\mu_H\left(x_1\right), \mu_{NV}\left(x_2\right)\right) \tag{11.35}$$

$$w_4 = \min\left(\mu_H\left(x_1\right), \mu_N\left(x_2\right)\right) \tag{11.36}$$

11.4.3. Következtetés

A szabályoknak megfelelően a kimeneti univerzum tagsági függvényein elvégezzük az alfa-vágást az egyes szabályok tüzelési értékével (11.37), (11.38), (11.39), (11.40):

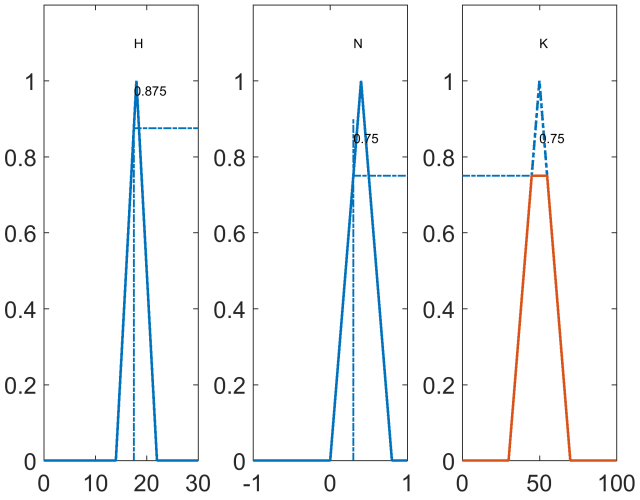
$$\mu^*_{B_1} = \min_{u\in[0\cdots100]}\left(w_1, \mu_{TNY}\left(u\right)\right) \tag{11.37}$$

$$\mu^*_{B_2} = \min_{u\in[0\cdots100]}\left(w_2, \mu_{NY}\left(u\right)\right) \tag{11.38}$$

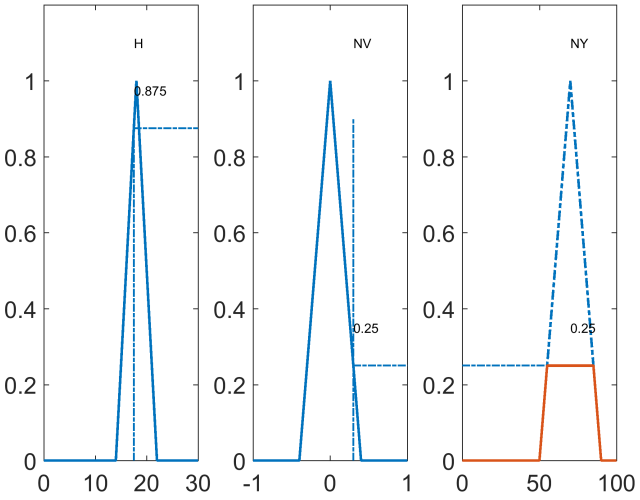
$$\mu^*_{B_3} = \min_{u\in[0\cdots100]}\left(w_3, \mu_{NY}\left(u\right)\right) \tag{11.39}$$

$$\mu^*_{B_4} = \min_{u\in[0\cdots100]}\left(w_4, \mu_{K}\left(u\right)\right) \tag{11.40}$$

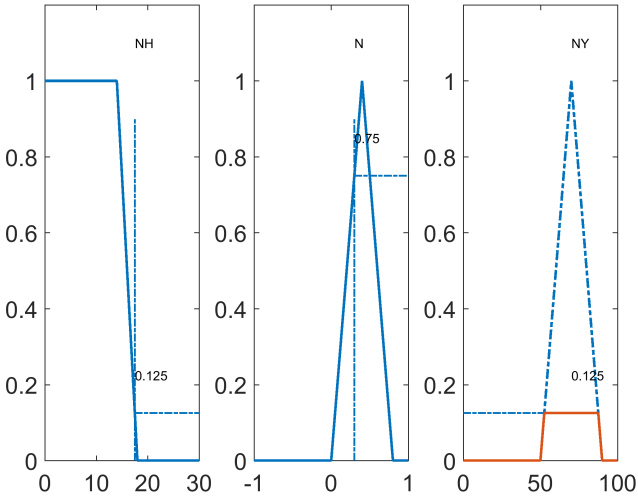
Az alfa-vágás a négy szabálynak megfelelően a 11.22, 11.23, 11.24, 11.25. ábrákon van szemléltetve.



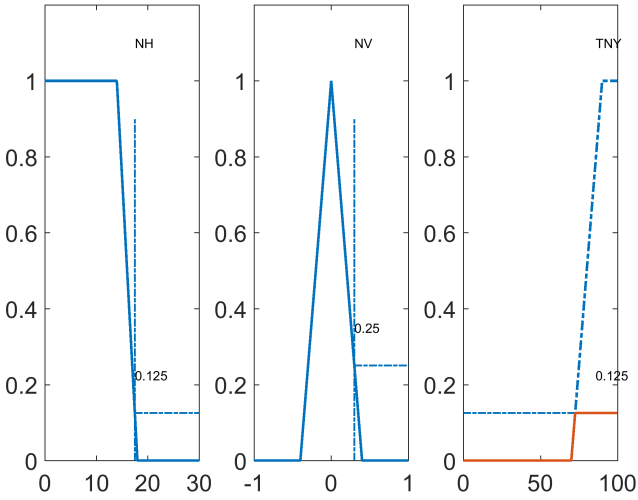
11.22. ábra. Első szabály – alfa-vágás



11.23. ábra. Második szabály – alfa-vágás



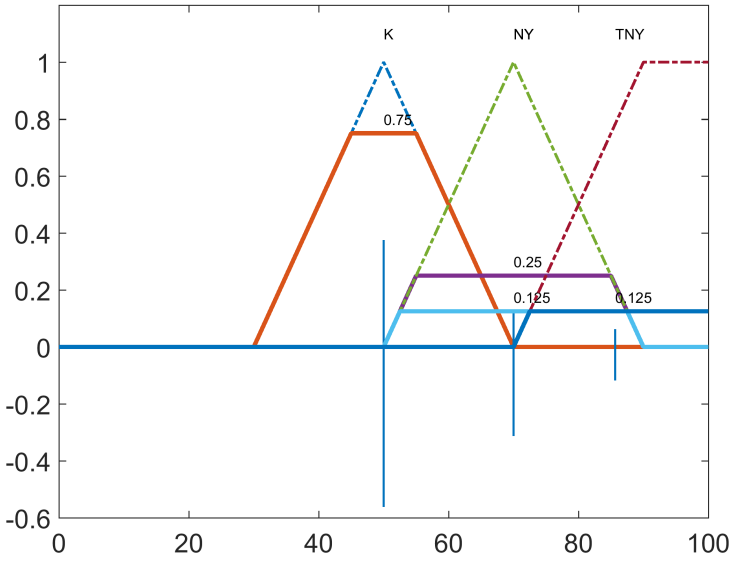
11.24. ábra. Harmadik szabály – alfa-vágás



11.25. ábra. Negyedik szabály – alfa-vágás

11.4.4. Defuzzyfikálás

Súlypont módszert alkalmazva a defuzzyfikálásra, a kimeneti univerzumon az egyes szabályok tüzelési értékével elvégzett alfa-vágást követően az alábbi részhalmazokból kell a kimenetet kiszámolni (11.26. ábra):



11.26. ábra. Kimeneti univerzumon a szabályoknak megfelelő aktív halmazok az alfa-vágást követően

Súlypontok számítása (11.41):

$$u_i^* = \frac{\int\limits_{u \in supp\left(B_i^*\right)} \mu_{B_i^*}(u)\, u du}{\int\limits_{u \in supp\left(B_i^*\right)} \mu_{B_i^*}(u)\, du} \tag{11.41}$$

Területek számítása (11.42):

$$T_i = \int\limits_{u \in supp(B_i)} \mu_{B_i^*}(u)\, du \tag{11.42}$$

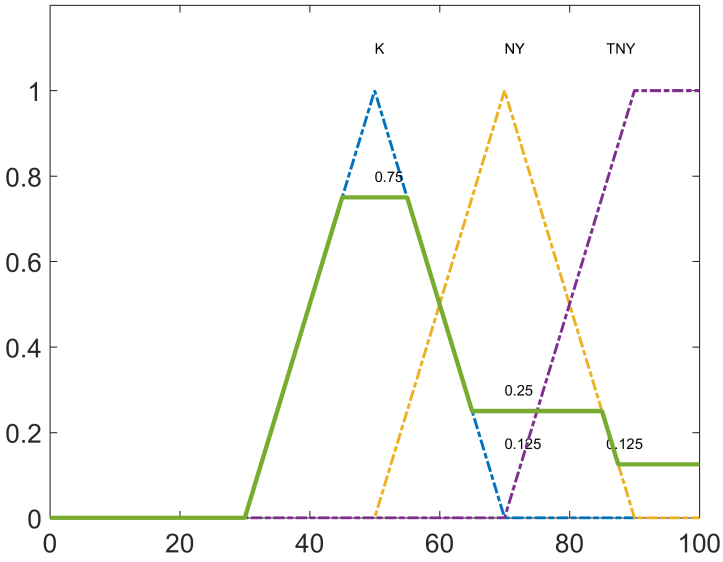
A súlypontmódszerrel a kiértékelés a (11.43) képlet szerint számolható:

$$u_{COG} = \frac{\sum_{i=1}^{r} T_i y_i}{\sum_{i=1}^{r} T_i} = \frac{T_1 u_1 + T_2 u_2 + T_3 u_3 + T_4 u_4}{T_1 + T_2 + T_3 + T_4} \tag{11.43}$$

Az egyes halmazokra a kiszámolt területek és súlypontok $u_1 = 50{,}00$, $u_2 = 70{,}00$, $u_3 = 70{,}00$,$u_4 = 85{,}64$, $T_1 = 18{,}75$, $T_2 = 8{,}75$, $T_3 = 4{,}68$, $T_4 = 3{,}60$.

A defuzzyfikálást követően a kiszámolt vezérlőjel u = 61.09

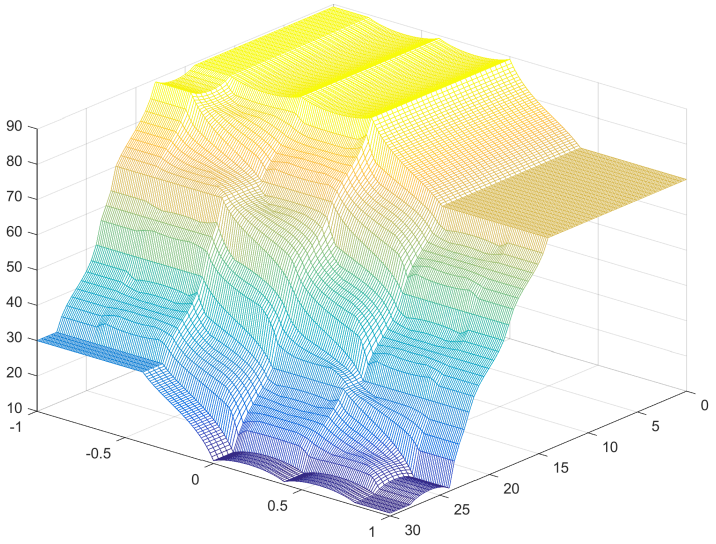
A terület középpont defuzzyfikálási módszerre a kimeneti univerzumon s-normát kell alkalmazni az egyes halmazok egyesítésére, majd ki kell számolni a halmazt leíró tagsági függvényre a geometriai középpontot (11.27. ábra).



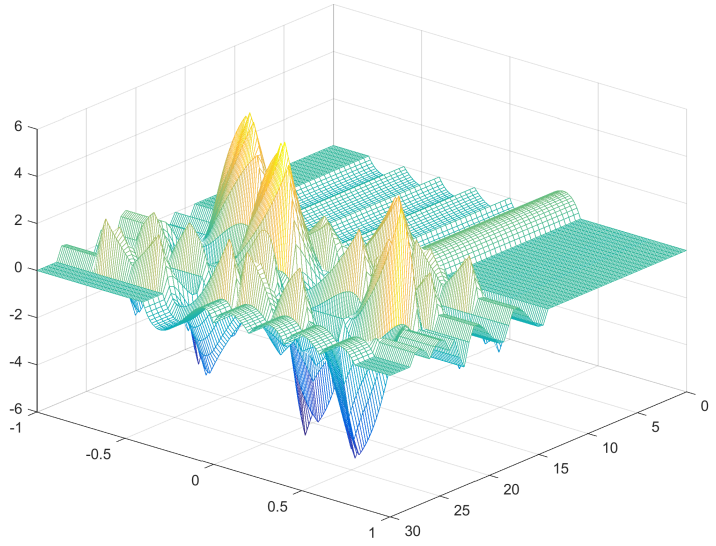
11.27. ábra. COA defuzzyfikálás

A COA defuzzyfikálást követően a vezérlőjel értéke 57,95.

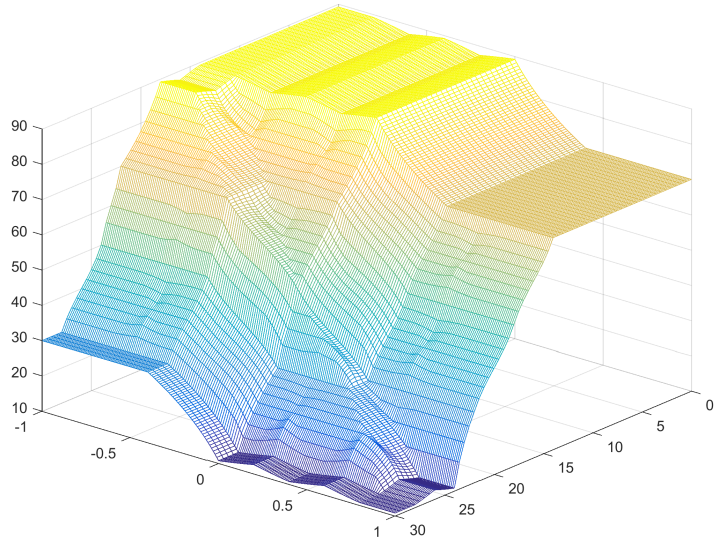
A 11.28. és 11.29. ábrákon a két defuzzyfikálási módszerre a szabályozási felületek vannak ábrázolva, valamint a két megoldás közötti eltérés (11.30. ábra).



11.28. ábra. Szabályozási felület COG-kiértékelési módszerrel



11.30. ábra. A két COG- és COA-kiértékelési módszer közötti különbség



11.29. ábra. Szabályozási felület COA-kiértékelési módszerrel

11.4.5. Lehetséges megoldások a fuzzy szabályozó hangolására

Egy Mamdani típusú következtető rendszer hangolásakor nagyjából kétféleképpen lehet beavatkozni:

- a tagsági függvények paramétereinek módosítása a kimeneti vagy bemeneti univerzumon,
- a szabálybázisból szabályoknak a változtatása.

Ha nagyobb mértékű beavatkozásra van szükség, abban az esetben ajánlott a szabálybázisból a szabályokat módosítani. A bemeneti vagy kimeneti univerzumon a következtető rendszer érzékenysége növelhető, ha több tagsági függvényt alkalmazunk az univerzum lefedésére. A tagsági függvényekre a következő műveletek alkalmazhatóak a szabályozó hangolására:

- kontrakció,
- dilatáció,
- a tagsági függvények jobbra vagy balra való eltolása.

Mérések alapján meghatározható, hogy melyek azok a tagsági függvények és szabályok a szabálybázisból, amelyek befolyásolják a pillanatnyi bemenetre

a szabályozó kimenetét. A fuzzy következtető rendszer pillanatnyi bemenetei alapján meghatározható, hogy melyek az aktív tagsági függvények. Az aktív tagsági függvények alapján a szabálybázisból meghatározhatók az aktív szabályok, majd a kimeneten az aktív tagsági függvények, melyek beavatkozást igényelnek. Az egyes szabályok tüzelési értékét figyelembe véve, az aktív szabályok közül, azon szabályokhoz kapcsolódó tagsági függvények paramétereit javasolt módosítani, amelyeknek a tüzelési értéke nagyobb. Például egy adott bemeneti párra, ha nagyobb kimenetre van szükség, a legaktívabb szabály következtető részében választható egy tagsági függvény, amely nagyobb értékeket fed le. Finomabb hangolás során alkalmazható a kimeneti univerzumból az aktív szabálynak megfelelő tagsági függvénynek jobbra való eltolása.

III. rész

Neurális hálózatok FPGA-alapú megvalósítása

12. fejezet

FPGA alapú neurális hardver

Ebben a fejezetben neuronhálók hardver alapú megvalósításával kapcsolatos ismereteket tárgyaljuk: a neuronháló különböző elemeiben kialakítható párhuzamosíthatóság tanulmányozása, aktivációs függvények hardveres megvalósítása, súlytényezők, különböző változók tárolása, az alkalmazott aritmetika, az egyes adatok tárolásához szükséges bitszélesség a kívánt megoldás elfogadható pontosságának eléréséhez, hardveresen megvalósított neuronhálók összefoglalása.

12.1. FPGA áramkörök

Az FPGA áramkörök egyre nagyobb szerepet kapnak a komplex, nagy számítási kapacitást igénylő algoritmusoknak valós időben való megvalósítása során. Nagyon fontos az FPGA áramkörök működésének, fontosabb alkalmazási területüknek az ismerete, a hardverleíró nyelvek feltérképezése, amelyek alkalmazhatóak az FPGA áramkörökben való tervezésre.

12.1.1. FPGA áramkörök szerkezete

Az FPGA digitális áramkör konfigurálható logikai komponenseket tartalmaz programozható összeköttetésekkel a komponensek között. Az FPGA áramkörök nagyon változatosan konfigurálhatóak számos feladat megvalósítására. A logikai komponensek és összeköttetéseik az FPGA legyártása után, vagyis a „felhasználás helyén” programozhatók. Beszélhetünk egyszer programozható vagy akár többször is újrakonfigurálható áramkörökről,

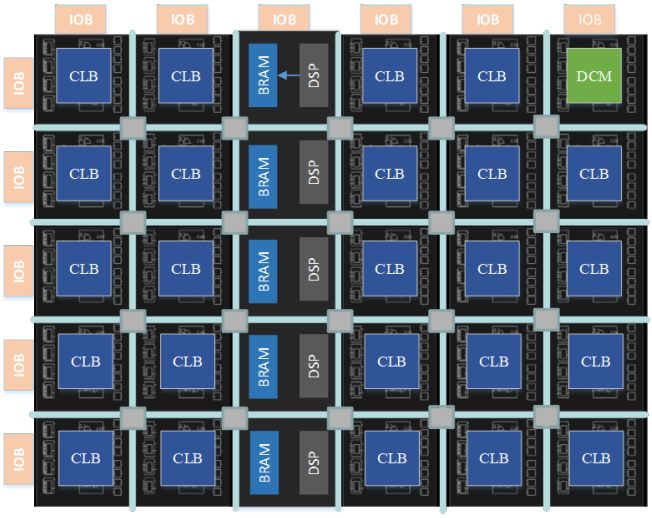
annak függvényében, hogy milyen technológiára épül az áramkör: FUSE, ANFIFUSE, PROM, EPROM, EEPROM, FLASH, SRAM [71].

Az FPGA áramkörök konfigurálható logikai tömbökből épülnek fel (CLB – Configurable Logic Block). A CLB tömbök vezetékek (függőleges és vízszintes) halmazával vannak összekapcsolva. A függőleges és vízszintes vezetékek találkozásánál programozható kapcsolómátrixok (Switch matrix) találhatóak. Az FPGA áramkörökből a jeleket a külvilág felé, valamint az FPGA áramkörbe a kimeneti/bemeneti (IOB – Input Output Block) modulokon keresztül lehet ki/be vezetni.

A konfigurálható logikai tömbök logikai kapukat, multiplexereket, keresőtáblázatokat, valamint bistabil áramköröket tartalmaznak. A CLB-ben található elemek, valamint a CLB modulok változatosan összekapcsolhatóak. Minden egyes beállítási lehetőség (kereső táblázat konfigurálása, bistabil kezdeti értéke stb.) hátterében valamilyen tárolóegység található, amely meghatározza az FPGA-ban kialakított áramkört. A RAM alapú FPGA áramkörökbe bekapcsolás után be kell programozni a bitfolyamot, amely konfigurálja az FPGA áramkört [72], [73].

Az FPGA áramkörök a következő fontosabb elemeket tartalmazzák:

- CLB – konfigurálható logikai blokk
  - Logikai kapuk
  - Kereső táblázat (LUT – Look Up Table), a LUT lényegében memóriában tárol kombinációs hálózatot = Igazság táblázat
  - Osztott memória (distributed RAM)
  - Bistabil áramkörök
  - Multiplexer áramkörök
- Ki/bemeneti tömbök (nagy sebességen működő IOB)
- Beágyazott RAM memória – Block RAM
- Digitális jelfeldolgozó modulok (DSP) vagy szorzó áramkörök (MUL)
- Órajelmenedzselő modulok (DCM Digital Clock Manager)
- Speciális modulok az FPGA áramkör családjának függvényében
  - Beágyazott processzormagok
  - Gigabit tranceiver modulok



12.1. ábra. FPGA áramkörök szerkezete

A digitális rendszerek tervezésére számos módszer alkalmazható, mint a hardverleíró nyelv alapú, kapcsolási rajz alapú, modellalapú tervezés (Matlab Simulink) vagy akár magas szintű programozási nyelvek (C, C++, SystemC). A hardverleíró nyelvek (VHDL, Verilog) kiemelkedő szerepet kapnak a hardvertervezés során. A hardverleíró nyelv (HDL) egy speciális számítógépes programozási nyelv, amelyet elektronikus áramkörök, a leggyakrabban digitális áramkörök struktúrájának és viselkedésének a leírására alkalmaznak. Egy hardverleíró nyelv lehetővé teszi pontos, formális leírását egy elektronikus áramkörnek, és egy megfelelő eszköztár alkalmazásával a HDL leírás alapján elvégezhető az elektronikus áramkör automatizált elemzése, szimulációja és fizikai megvalósítása. Valamely tervezési módszerrel elkészül az áramkör specifikálása, majd az alkalmazott módszer függvényében egy dedikált eszköztár (szintézis tool) elkészíti az FPGA áramkör programozásához szükséges bitfolyamot.

12.1.2. FPGA áramkörök alkalmazásának előnyei

Az FPGA áramkörök alkalmazásának előnyei:

– *Teljesítmény (Performance).* Az FPGA áramkör szerkezetéből, a hardveres párhuzamosságból adódóan az FPGA-k meghaladják a digitális jelfeldolgozó processzorok számítási teljesítményét. A

szekvenciális végrehajtással összehasonlítva, a párhuzamos kialakításból adódóan, órajelenként több műveletet képesek végrehajtani. Összehasonlítva egy processzor alapú rendszerrel, amelyen operációs rendszer fut, míg az egyik esetben folyamatszinten ütemezhetők a feladatok, az FPGA áramkörökben a „feladatok" órajelszinten ütemezhetőek. Számos alkalmazásban az FPGA áramkörök használata többszörösen meghaladja a számítási kapacitás/egységárat, összehasonlítva a digitális jelfeldolgozó processzorokkal. A ki- és bemenetek hardverszintű vezérlése gyorsabb válaszidőt biztosít, valamint az alkalmazás elvárásainak megfelelően speciális funkciók kialakítását teszi lehetővé.

– *Piacra kerülési idő* (Time to market). Az FPGA technológia flexibilis és gyors prototípus-készítést tesz lehetővé. Egy ötlet, egy elképzelés hardveresen gyorsan megvalósítható és tesztelhető, megspórolva az egyedi alkalmazásspecifikus integrált áramkörök során számítandó hosszú tervezési és gyártási folyamatot. Az FPGA áramkörökben a tervezésre és tesztelésre rendelkezésre álló eszközök segítik a prototípusok rövid időn belül való elkészítését. A magas szintű szoftver eszközök növekvő elérhetősége csökkenti a tanulási folyamatot, újabb absztrakciós szintek bevezetésével, hasznos IP (Intellectual Property) magok létrehozásával.
– *Költség (Cost)*. Az ASIC áramkörök gyártási költségei messze meghaladják az alkalmazás FPGA áramkörökkel való hardveres megvalósításának költségeit. Az ASIC áramkörökbe való beruházás csak évenkénti többezres eladás esetében kifizetődő a gyártók számára. Sok esetben egyes hardverfunkcionalitások megvalósítására tízes-százas nagyságrendben van igény. Ebben az esetben az ASIC áramkörökkel szemben az FPGA áramköröket kifizetődő alkalmazni. Sok esetben a rendszerrel kapcsolatos elvárások idővel módosulnak, az inkrementális változások költsége elhanyagolható az FPGA áramkörök alkalmazásával, összehasonlítva az ASIC áramkör újragyártásának nagy költségeivel.
– *Megbízhatóság (Reliability)*. A processzor alapú megvalósítások sok esetben több absztrakciós rétegre épülnek, segítve a folyamatok ütemezését és a folyamatok közötti erőforrás-megosztást. Minden egyes processzormag egy időben csak egyetlen utasítást képes végrehajtani, emiatt a processzoralapú rendszerek esetében folyamatosan felmerül a kérdés, hogy alkalmazhatóak-e időkritikus feladatok ütemezésére. Az FPGA-k esetében a valódi párhuzamos megvalósítással, minden

egyes feladat esetében dedikált determinisztikus hardveregységgel minimálisra csökkennek a felmerülő megbízhatósági aggályok.
– *Hosszú távú karbantartás (Long-term maintenance).* A hosszú távú karbantartás sokkal egyszerűbben megvalósítható, ha FPGA áramkört alkalmazunk, összehasonlítva az ASIC alapú megvalósítással. Míg az FPGA alapú hardver az FPGA újrakonfigurálását igényli, lehetővé téve a jövőbeli módosításokat, az ASIC alapú megvalósítás esetében minden egyes módosítás az áramkör újratervezését és gyártását jelentheti.

FPGA áramkörök alkalmazásából származó előnyök:
– Az FPGA áramkör szerkezetéből adódó magas fokú párhuzamosítás
– Pipeline és párhuzamos – pipeline architektúrák kialakítása, biztosítva a működési sebesség és hardveres erőforrásigény közötti középutat
  – A tervező dönti el az elvárt működési sebesség és a rendelkezésre álló hardver függvényében, hogy térben vagy időben terjeszkedik
  – Működés közbeni újrakonfigurálás
– Nagy működési sebesség
  – Minden órajelre több egység végez műveletet
  – Időkritikus részek külön hardveregységben vannak megvalósítva
  – Speciális elemek alkalmazása nagy sebességű modulok közötti adatátvitelre
– Gyors prototípus-tervezés
  – Speciális tervező- és tesztelőeszközök
  – Elkészített modulok újrahasznosítása (IP-magok újrahasznosítása)
– Egyszerű illesztés egy már meglévő hardvereszközzel
– Bármi megvalósítható, amit csak el tudunk képzelni
– Speciális megbízhatóságot igénylő rendszerek esetében dedikált újrakonfigurálható áramkörök állnak rendelkezésre.

12.2. Neurális hálózatok FPGA alapú megvalósításának előnyei

A beágyazott rendszerek egyre nagyobb teret hódítanak a különböző célfeladatokban való alkalmazásokban. Az FPGA áramkörök mind a működési sebesség, mind a kapacitás (ekvivalens logikai kapuk száma), mind a szerkezetük szempontjából nagyméretű fejlődésen mentek át. A digitális áramkörök prototípusának tervezése és tesztelése szempontjából az FPGA áramkörök egy hatékony eszközt jelentenek komplex alkalmazások (magas működési frekvencia, nagy számítási kapacitásigény) megvalósítása esetében.

A mesterséges neurális hálózatok masszívan párhuzamos architektúrájú soft-computing módszerek. A természet inspirálta soft-computing eszközök, ezek közül kifejezetten a mesterséges neurális hálók hardveres megvalósítása, teljes mértékben kihasználják az FPGA áramkörök nyújtotta számítási hatékonyságot. A mesterséges neurális hálózatok párhuzamos struktúrája alkalmassá teszi ezeket az újrakonfigurálható digitális áramkörön való megvalósításra. Az FPGA áramkörön való megvalósítás szempontjából a megvalósítandó soft-computing eljárás architektúrája nagymértékben befolyásolja az FPGA áramkörben elfoglalt elemek számát, illetve a rendszer sebességét. Erőforrás-kihasználtság, illetve sebesség szempontjából fontos szerepet játszik a pipeline (pipeline párhuzamos) architektúrák tanulmányozása, illetve megvalósítása.

A mesterséges idegsejtháló alapú alkalmazások párhuzamos implementálása nagyon fontos a valós idejű működés biztosításának szempontjából, a nagyszámú igényelt aritmetikai művelet miatt. A párhuzamos megvalósításra felhasználhatóak párhuzamos számítógépek, digitális és analóg hardvereszközök, de sok esetben a neurális hálók optikai eszközön való kialakításával is találkozunk. A 90-es években a legsikeresebb és valós mesterséges neurális hálókra alapuló alkalmazásokat szimulációszinten valósították meg számítógépeken. Ez a fajta megvalósítás nem tette lehetővé a neurális hálók struktúrája által biztosított párhuzamos működés kiaknázását.

A problémák, amelyek felmerültek egy hatékony neuronhálóra épülő alkalmazás megvalósítása során, megérthetők, ha áttekintjük a hardveres megvalósítással kapcsolatos legfontosabb elvárásokat:

- a hardver lehetővé kell tegye nagyszámú neuron egyidejű működését;
- nagyfokú kommunikációs kapcsolat biztosítása a neuronok között;

– az architektúra eléggé flexibilis kell legyen ahhoz, hogy különböző neuronháló-modellek és a tanító algoritmusok implementálására biztosítson hátteret;
– képes kell legyen az aktivációs függvények és tanító algoritmusokban megjelenő nemlineáris függvények megvalósítására;
– elegendő lokális memóriával kell rendelkezzen a nemlineáris aktivációs függvények, a súlytényezők és a tanító algoritmus által használt lokális információk tárolására.

Ezenkívül, ahogy bármilyen más architektúra esetében, a neuronhálóra épülő rendszer tervezése során is figyelembe kell venni a kivitelezési költségekkel, működési sebességgel, konkrét alkalmazásba való integrálással kapcsolatosan felmerülő kérdéseket.

12.2.1. Paraméterek értékelése és osztályozása

Egy neurális hálózat értékelhető/tanulmányozható a hálózat topológiája, az aktiváló függvény, a tanítási algoritmus, a bemenetek/kimenetek száma, a processzáló neuronok száma és szinaptikus kapcsolatai, rétegek száma szempontjából. Hardveres megvalósítás esetében ugyanakkor különböző megkötéseket tartalmazhat: a felhasznált technológia (analóg, digitális, hibrid, FPGA), az adatok kódolása (fixpontos, lebegőpontos), a súlytényezők tárolása, a pontosság, programozható vagy hardveres összeköttetések, on-chip vagy chip-in-the-loop tanítás, on-chip vagy off-chip aktivációs függvények, kereső táblázatok stb. Ezek alapján különbözőképpen osztályozhatók a felhasználható hardvereszközök. A legáltalánosabban használt referencia-paraméterek a különböző megvalósítások összehasonlítására a következők:

– Connections Per Second – másodpercenkénti kapcsolatok (CPS) – a feldolgozási sebességet jellemzi. Másodpercenként processzált szinaptikus kapcsolatok (szorzás, összegzés, MACC műveletek) számát méri a tesztelési fázis során. Jelzi, hogy egy specifikus algoritmus mennyire felel meg egy adott architektúrának.
– Connection Updates Per Second (CUPS) – a neurális hálózat tanulási sebességét jellemzi, a másodpercenként frissített súlytényezők számát jelenti.
– Szinaptikus energia: A súlytényezők számításához és frissítéséhez szükséges átlag energia, WCPS (watt per kapcsolatok per másodperc)-ben mérve, vagy J per kapcsolatok száma.

– Connection primitives per second (CPPS), ahol a bi és bw a bemenetek, illetve a súlytényezők pontosságát jelenti bitben.

12.2.2. Párhuzamosság a neurális hálózatokban

A neurális hálózatok többféle párhuzamosságot mutatnak, és a párhuzamosság alapos vizsgálatához egyrészt szükség van a legalkalmasabb hardverszerkezet kiválasztására, másrészt a neurális háló struktúrájának az adott hardverstruktúrára való legmegfelelőbb leképzésére. Általában csak kisméretű neurális hálózatok esetében valósítható meg a teljesen párhuzamos hardveres végrehajtás, nagyobb méretű hálók esetében látszólagos párhuzamosságra (pipe-line elvű megvalósítás) van szükség. A látszólagos párhuzamosság maga után vonja különböző részeknek a szekvenciális feldolgozását. A neurális hálókban előforduló párhuzamosságok:

– *Tanulási párhuzamosság:* különböző tanulási szakaszok párhuzamosan futtathatók, például SIMD vagy MIMD típusú processzorokon. A párhuzamosság szintje általában közepes, és szinte teljesen leképezhető egy nagy kapacitású FPGA áramkörbe. Rétegenkénti párhuzamosság: egy többrétegű neurális háló esetében a különböző rétegek párhuzamosan processzálhatóak. A párhuzamosság ezen a szinten tipikusan alacsony és korlátozott mértékű, de még mindig kihasználható a pipeline alapú feldolgozás.
– *Neuronszintű párhuzamosság*: Ez a szint, ami megfelel az egyes neuronoknak, talán a legfontosabb szintű párhuzamosság. Ha ezen a szinten ki van használva a párhuzamos processzálás, abban az esetben a magasabb szinteken is megoldható a párhuzamos feldolgozás. Az utóbbi időkben az FPGA áramkörök mind kapacitásuk (logikai kapuk száma), mind működési sebességük tekintetében nagymértékű fejlődést mutatnak, egyre nagyobb méretű neurális háló megvalósítását téve lehetővé.
– *Súlytényező szintű párhuzamosság*: a kimenetszámítás során a súlytényezők és a bemenetek szorzása párhuzamosan van megvalósítva, és a részszorzatok összeadása is magas fokú párhuzamosítással valósítható meg.
– *Bit szintű párhuzamosság*: a megvalósítás szintjén sokféle párhuzamosság alkalmazása áll rendelkezésre, ha figyelembe vesszük az egyes funkcionális egységek működését [74].

12.3. FPGA alapú neurális hardvermegvalósítások

Eldredge sikeresen alkalmazta a backpropagation algoritmust, melyhez Xilinx XC3090 CPLD áramkörökből platformot épített, és melynek elnevezése Run-Time Reconfiguration Artificial Neural Network (RRANN) lett [75]. Eldredge architektúrája alapján Beuchat és társai kidolgoztak egy RENCO elnevezésű FPGA-s platformot. A RENCO rendszer négy Altera FLEX 10K130 FPGA-t tartalmaz, melyek egy LAN hálózaton keresztül összekapcsolhatók (pl. internet vagy más), egy 10Base-T interfészen át újrakonfigurálhatók és monitorizálhatók. A RENCO-t írásfelismerésre fejlesztették ki.

Ferucci és Martin [76], [77] egy FPGA-s platformot fejlesztettek ki, melynek neve Adaptive Connectionist Model Emulator (ACME) és több Xilinx XC4010 FPGA-ból áll. Sikeresen alkalmazták a 2 bemenetes XOR probléma megoldására. Egy 3 bemenetű, 3 rejtett rétegű, 1 kimenetű hálót implementáltak. Skrbek ezt a struktúrát arra használta fel, hogy bizonyítsa a saját fejlesztésű backpropagation algoritmus működését FPGA megvalósításban [78]. Skrbek FPGA áramkörre épülő rendszere, az ECX kártya, képes radiális bázisfüggvények implementálására, és sikeresnek bizonyult számos feladat megoldásában, mint például számfelismerés, hangfelismerés.

A GANGLION egy, az IBM kutatórészlege által kifejlesztett alkalmazás, amely a Xilinx XC3000 sorozatból származó FPGA áramkörökön alapul. A rendszer öt ilyen elemből épül fel, melyek egy VME-alapú nyomtatott áramkörön vannak elhelyezve. Az így kapott rendszer egy 12 bemenetű, 14 rejtett neuronnal rendelkező, 4 kimenetű MLP. Az alkalmazás teljesen párhuzamos, 8 bites bemeneti adatokkal és 8 bites súlyokkal dolgozik. Így az elért sebesség 4,48 GCPS [79].

Működésüket tekintve a neurális hálók különböző feladatok megoldására alkalmasak: társítás, osztályozás, nemlineáris függvény approximáció. A szerzők [80] különböző feladatokra külön algoritmust implementáltak FPGA áramkörben. E célból egy dinamikusan újrakonfigurálható hardvergyorsítót használtak, a RAPTOR2000-et, mely alaplapból és hat darab alkalmazásspecifikus modulból áll. Az alaplap biztosítja PCI buszon keresztül a modulok közötti kommunikációs infrastruktúrát. A hat modul gyűrűtopológiában van összekötve. Minden modul egy Xilinx Virtex XCV1000 FPGA-ból és 256 Mbyte SDRAM-ból áll. Minden modul egy helyi közös buszhoz van kötve, így kommunikálnak más eszközökkel vagy modulokkal és a gazda számítógéppel egy PCI buszon keresztül. A gazda rendszermemóriájához való gyors hozzáférés érdekében Dual portos SRAM-et használ, mely elérhető bármely

modul számára. A RAPTOR2000-n található modul újrakonfigurálását a gazda számítógép kezdeményezi.

Poormann rendszerében [80] 11,3 GCPS sebesség érhető el, mely jóval nagyobb, mint a személyi számítógépek által elérhető 80 MCPS (AMD athlon, 800 MHz) sebesség volt. Poormann önszervező Kohonen-hálót implementált a RAPTOR2000-re, különböző Xilinx Virtex eszközöket használva. Az aritmetikai műveleteket 16 bit pontosságú fixpontos ábrázolással oldja meg. Neurális Asszociatív Memória NAM implementálása esetén a RAPTOR2000-en hat modul van elhelyezve. Ebben az esetben minden modullal létrehozható 512 neuron. Egy olyan innovatív FPGA-s megoldás is született, mely dinamikusan módosítja méretét, melynek elnevezése FAST (rugalmasan alkalmazkodó méretű topológia) [81].

A legtöbb neurális háló-modell esetében felmerülő problémák a következők: a rétegek számának meghatározása, egy réteg neuronjai számának a meghatározása, a kapcsolódások módjának meghatározása. Az ontogenetikus neurális hálók azért születtek, hogy ezeket a problémákat kiküszöböljék, azáltal, hogy lehetővé teszik a topológia dinamikus változtatását. Az ART (adaptív rezonencia elmélet) és a GAR (Grow and Represent) ilyen típusú hálók, és a FAST rendszer is ezekből ered. A [81] munkában egy ontogenetikus neurális hálót implementáltak egy FPGA-s platformra, melynek neve FAST (rugalmasan alkalmazkodó méretű topológia). A FAST-ot három típusú neuronháló implementálására használták: ART, AHC (Adaptive Heuristic Critic) és DYNA-SARSA.

Az inverz inga problémája az instabil rendszerek klasszikus példája, és nagyon sok szabályozási algoritmus tesztelésére használják [82]. A FAST-AHC nem tud olyan gyorsan általánosítani, mint a hiba-visszaterjesztés algoritmus, de a tanítás gyorsabb és hatékonyabb. Ez annak köszönhető, hogy az AHC tanítási technika általánosítható úgy, mint a helyi tanítás egy formája, amikor csak a neuronháló aktív neuronjai vannak frissítve, ellentétben a hiba-visszaterjesztés algoritmussal, ahol a tanítás globális és minden neuron frissítése szükséges. A FAST architektúra az első volt ebből a kategóriából, mely nem felügyelt tanítású, ontogenetikus neuronhálókat használ, de megvannak a korlátai is: egyszerű feladatokat képes megoldani, melyek dinamikus vagy online osztályozást feltételeznek.

A [83]-ban egy valószínűségi neuronhálót (PNN) implementáltak FPGA-ra multispektrális képek osztályozására, melyeket a LANDSAT-2 EOS műholdaktól nyertek. A multispektrális képeket szkennerek képezik, és minden képsorozat egy sávspektrumnak felel meg. A PNN osztályozási

algoritmusa annak a valószínűségét számolja ki, hogy egy bizonyos pixel melyik osztályba sorolható. A valószínűség meghatározása a kép-vektorokkal végzett komplex kivonási, szorzási és exponenciális műveleteket feltételez. A PNN architektúrája két FPGA XC 4013E rendszerből áll (XFPGA és YFPGA), mindkettőn 13000 kapu van egy X213-as gyorsítón. A FPGA-n lévő általános és dedikált erőforrások korlátja miatt a kevésbé erőforrás igényes fixpontos számításokat használják a lebegőpontos helyett. De természetesen megvan a lehetősége a nagyobb pontosságnak, az erőforrások rovására.

De Garis és társai [84], [85] fejlett technikákon alapuló neuronhálót építettek, és így az áramkörben futó tanítást tudtak alkalmazni. De Garis egy FPGA-s platformra celluláris automatát (CBM-CAM Brain Machine) dolgozott ki, mely egy genetikus algoritmust alkalmaz egy celluláris automata tanításához. Bár a CBM-re úgy tekintenek, mint ami lehetővé teszi az on-chip tanítást, nem tartalmaz egyetlen tanítási algoritmust sem. Ehelyett a helyben tanítás genetikus algoritmussal van megoldva (ez inicializálja minden celluláris automata számára a konfigurálási adatokat, majd növekszik a neuronháló topológiája, mely a celluláris automata funkcionális jellemzője). A neurális hálózatok fenntartása CBM-mel elég nehézkes, ugyanis a modulok közötti kapcsolatot manuálisan, offline kell beállítani.

Nordstorm [86] egy FPGA-n alapuló, REMAP-nak elnevezett moduláris neuronhálót alakított ki. Nordstorm úgy gondolta, hogy az újrakonfigurálható számítás felhasználható különböző típusú modulok (különböző algoritmusok) támogatásához. Ez a modell felhasználható heterogén típusú moduláris neuronhálók esetében, mint az „expertek hierarchiája”, melyeket Jordan és Jacobs javasolt. A $REMAP-\alpha$ és $REMAP-\beta$ közötti különbség a felhasznált FPGA kapacitásában rejlik. A $REMAP-\alpha$ esetében FPGA Xilinx XC3090-et használtak a különböző neurális algoritmusok tipizálására, míg a $REMAP-\beta$ esetében először Xilinx XC4005, majd Xilinx XC4025 áramkört. A $REMAP\beta$-t neurális hálók hardveremulátorként és tanító eszközként használták, mely a következő típusú hálókat tudja implementálni: Adaline, Madaline, backpropagation algoritmus, kétirányú asszociatív memóriahálózatok, Hopfield asszociatív memória, counter propagation hálók, önszerveződő térkép (SOM), adaptív rezonancia elmélet (ART). A feldolgozási sebesség növelése érdekében megalkottak egy harmadik prototípust is, az ASIC alapú $REMAP-\gamma$-t.

Egy új típusú, FPGA-alapú neurális háló született, az RTR-MANN, mely a legújabb generációs eszközök és módszerek nagyobb hatékonyságát hivatott igazolni. Az RTR-MANN skálázhatósága és funkcionális sűrűsége sokkal jobb, mint a régebbi típusú FPGA-alapú neurális hálók.

A magas szintű, HLS követő nyelvek (Handel-C, Impulse-C, C/C++) a prototípus-fejlesztés gyorsítását szolgálta [87], [88]. Megállapítást nyert az a tény, hogy egy 32 bites lebegőpontos aritmetika nem annyira erőforrás-takarékos, mint egy 16 bites fixpontos aritmetika a legújabb generációs FPGA-k esetében. Ez nem jelent újdonságot ezen a területen, csak a legújabb generációs FPGA-kra vonatkozóan. A felhasznált aritmetika arra lett kidolgozva, hogy az RTR-MANN hasznosítsa – 16 bites fixpontos aritmetikát alkalmazó operátorokat használ, melyek a Xilinx XCV2000E Virtex-E FPGA-ra lettek optimizálva. Így egy sokkal hatékonyabb skálázhatóság és sűrűség érhető el.

A [88] munkában leírt kutatás részlegesen párhuzamos és teljesen párhuzamos architektúrákat ajánl a hiba-visszaterjesztéses algoritmus FPGA-n való implementálására. A projektek Handel-C-ben vannak kódolva, és egy Virtex2000e FPGA chipen leellenőrizve, különböző paraméterek alapján. A részleges, illetve teljes párhuzamosságú rendszerek 2,5, illetve 4-szer bizonyultak gyorsabbnak a szoftveres megoldásoknál. Egy másik munkában több típusú architektúrát ajánlanak a hiba-visszaterjesztés algoritmuson alapuló neurális háló Virtex2000 FPGA áramkörön való implementálására. Eredetileg kétsoros architektúrát elemeztek. Ezek célja a feladatok meghatározása és a párhuzamos architektúrák továbbfejlesztése volt. A szorzás-összeadás alapműveletek a hiba-visszaterjesztéses algoritmus végrehajtásában. A szorzás-összeadás művelet Branch-In és Branch-Out koncepciók alapján lett megvalósítva. A következő architektúrákat mutatták be:

- SIPEX és SIPOB: soros megközelítés, a kezdeti paraméterek tárolásában és a szigmoid függvény kereső táblázataiban (LUT) különbözik;
- PAROI, PARPP és PARIO: részlegesen párhuzamos megoldás, a műveletek Branch-In és Branch-Out modulokhoz való hozzárendelésében különbözik;
- FPAR: teljesen párhuzamos működést biztosító megvalósítás, az LFSR és szigmoid függvény által véletlenszerűen generált számok három változóval vannak lineárisan megközelítve.

A Handel-C nemcsak szimulációt biztosít, hanem C-> FPGA szintézist is. A fejlesztési idő sokkal rövidebb, mint VHDL-ben. A hibajavítási műveletek nagyon könnyen elvégezhetők. Fixpontos aritmetikát használ. A Handel-C fordító hibája, hogy nem tudja a számokat fixpontosan ábrázolni. A hiba-visszaterjesztéses algoritmus komplex számtani műveleteket igényel, a számok fixpontos ábrázolása nagyon fontos az algoritmus szempontjából.

A hiba-visszaterjesztéses algoritmus időben három fázisra osztható, ezáltal az FPGA-k működés közbeni újrakonfigurálása alkalmazható az algoritmus megvalósítására [87].

Bár a párhuzamosság a rétegek szintjén működött, a szoftveres megoldás esetében ez nem lehetséges. A kutatások kiterjeszthetők a pipeline módszer három fázisára: a kimenet kiszámítása, a hiba-visszaterjesztés, a súlytényezők adaptációja. A paraméterek, az eredeti topológia kombinálása és a súlytényezők BlockRAM memóriába való azonnali letöltése még kísérletek tárgyát képezheti. Így a RAM off-chip elérése nem szükséges, és az algoritmus végrehajtása ezáltal felgyorsul. A [87] dolgozatban bemutatott architektúrák leírására más leíró nyelvek is alkalmazhatók, mint például a VHDL. A Multi-FPGA rendszereket gyakran használják különböző tanító algoritmusok, mint például a hiba-visszaterjesztéses algoritmus implementálására. A Multi-FPGA rendszer nemcsak megengedi nagyméretű hálók implementálását, de nagy feldolgozási és tanulási sebesség érhető el ezek használatával.

Végkövetkeztetésként megállapítható, hogy az FPGA alapú neurális háló implementálása alapján osztályozhatók a különböző architektúrák. A neurális háló típusa függ a megoldandó feladathoz szükséges alkalmazástól. Az FPGA áramkörök elterjedésének kezdeti periódusában is próbálkoztak neuronhálók ilyen modelltípusú rendszereken való megvalósításával kisebb-nagyobb sikerrel. Az FPGA-k sűrűsége és sebessége nőtt, így már hatékonnyá vált ezek használata egyre nagyobb méretű neurális hálók megvalósításában.

A szorzás költséges művelet FPGA-k használatával, ugyanis egy neurális hálóban minden szinaptikus kapcsolat egy szorzót igényel, és ez a szám a neuronok számának négyzetével nő. A XILINX Virtex-5, Virtex-6 és a modern Zynq, Ultrascale SoC, UltraScale MPSoC sorozatú FPGA áramkörök több száz vagy ezer dedikált szorzóval vagy DSP modullal rendelkeznek. Krips és társai [89] egy olyan FPGA-s neurális háló megvalósítást mutatnak be, mely videoképeken valós időben érzékeli és követi a kezeket. Yang és Paindavoine [90] egy FPGA alapú beágyazott rendszerbe épített hardvert mutatnak be, mely 92%-os pontossággal ismer fel és azonosít arcokat videoszekvenciákban. Gadea és társai [91] bemutatják egy szisztolikus tömb implementálását egy többrétegű perceptron esetében egy Xilinx Virtex XCV400 FPGA-n egy soros online BP tanítási algoritmussal pipeline kialakításban. Huitzil és Girau [92] leképezik a LEGION (Local Excitatory Global Inhibitory Oscillator Network) spiking neuron modellt egy Xilinx Virtex XC2V1500FF896-4 eszközre, melyet képszegmentálásra használnak.

Rice és társai [93] úgy találták, hogy egy FPGA-alapú, biológiából inspirált kognitív modell 75-ször nagyobb átlagos teljesítményt mutat, mint a szoftveres megoldás egy Cray XD1 szuperszámítógépen. A hierarchikus Bayes háló-modellt alkalmazták, mely a George és Hawkins [94] által kidolgozott neokortexen alapul. Az egyszerű számítások és a magasfokú párhuzamosítás rendkívül hatékony hardver-implementációkat tesznek lehetővé. A legtöbb sztochasztikus neurális háló modell implementálása FPGA-n történik [95], [96], [97], [98]. Daalen és társai [97] egy FPGA-alapú bővíthető digitális architektúrát mutatnak be, melyben bit soros valószínűségszámítás oldja meg a párhuzamos szinaptikus műveleteket. A szerzők úgy vélik, hogy teljesen kapcsolódó többrétegű hálók implementálhatók időmultiplexeléssel, ezen architektúra felhasználásával.

Zang sztochasztikus számítási elméleten alapuló csökkentett számú hardvererőforrást felhasználva valósít meg nemlineáris aktivációs függvényeket, és egy masszívan párhuzamos nagyméretű neurális hálót FPGA áramkörön, kiváló hibatűrő képességekkel. A kialakított hardvert kisméretű szélturbinarendszer szélsebesség-érzékelő nélküli neurális hálóval történő vezérlésére alkalmazták [95].

A neurális processzorok hardveres implementálása komoly kihívás. Az újraprogramozható architektúrák kutatásában elért legújabb eredmények megkönnyítik ezt a feladatot, alapvető hardvertömböket biztosítva a neurális hálóstruktúrák megépítéséhez. A TOTEM neurális processzort véve alapul, a programozható logikai eszközökön implementált neurális hardverek (FPGA) előnyeit mutatja be [99].

Himavathi többrétegű előrecsatolt neurális hálók hardveres implementálását mutatja be FPGA áramkörök felhasználásával. A bemutatott alkalmazás célja a szükséges eszközök körének leszűkítése, úgy, hogy a sebesség ne csökkenjen, és hogy megoldhatóvá váljon bármely nagyméretű neurális háló egyetlen áramkörben történő megvalósítása. A rétegek szekvenciálisan vannak feldolgozva, a rétegek multiplexelését használva fel a nagyméretű neuronhálók implementálásához. Egy teljes háló létrehozása helyett csak a legnagyobb réteg van implementálva. Egy vezérlő modul irányításával ez az egyetlen hálóréteg különbözőképpen fog működni. A vezérlőtömb feladata, hogy a megfelelő működéshez biztosítsa a szükséges bemeneteket, súlyokat, aktivációs függvényeket. Az említett többrétegű háló Xilinx FPGA "XCV400hq240-n van implementálva [100].

Egy négyrétegű Regressziós Neurális Háló FPGA áramkörön való megvalósításával kapcsolatosan kapunk leírást a [101] publikációban. A neuronháló a következő négy rétegből tevődik össze: bemeneti réteg, mintaréteg,

összegző-, valamint kimeneti réteg. A neuronháló implementálására VHDL nyelvet alkalmaztak, a jelek kódolása fixpontos aritmetikával történt. A neuronháló-minták osztályozására volt tervezve, egyéb regressziós neurális hálókkal megoldható feladatokra is alkalmazható. A javasolt rendszer rugalmas és skálázható, könnyen módosítható a bemenetek és az osztályozandó minták száma.

A [102] dolgozat egy wavelet alapú neurális háló FPGA alapú implementálását tartalmazza. Egy gradiens algoritmust, melynek lokális minimuma is lehet, nem egyszerű elektronikus áramkörben megvalósítani. Egy másik használható és alkalmasabb módszer, amelyet megvalósítottak a dolgozatban, a részecskeraj-optimalizálás (PSO), mely egy populációalapú optimalizáló algoritmus. Az alkalmazott nemlineáris aktivációs függvény minél pontosabb megközelítésében Taylor sorba fejtést és look-up kereső táblázatot (LUT) használtak.

A [103] dolgozat egy EEG jelek osztályozására használt FPGA-s implementáció megtervezését és létrehozását mutatja be, kihasználva a hardver-szoftver tervezés, az újrakonfigurálható környezet előnyeit, egy olyan spiking neurális hálót (SNN) alkalmazva, mely meg tud különböztetni kép formájában ábrázolt EEG jeleket, mely kép a jel wavelet transzformációjából származtatott mintát tartalmazza. A rendszer néhány modulja futás közben részleges újrakonfigurálást használ az FPGA eszköz minél hatékonyabb kihasználása érdekében. A felhasznált hardvereszköz egy Opus Virtex 5 FPGA fejlesztő rendszer, PowerPC 440 processzormaggal. A kifejlesztett SNN felhasználja mind a processzormagot, mind a Virtex5 áramkör újrakonfigurálható eszközeit, és egy nagymértékben párhuzamosított architektúrát implementál a kitűzött feladat megoldása érdekében.

A tartó vektor gép (SVM) egy rendkívül erős tanító eszköz, mely nagy pontosságot biztosít számos osztályozási feladat megoldásában. Ez a dolgozat egy teljes mértékben skálázható FPGA-alapú architektúrát mutat be, melynek célja az SVM osztályozás gyorsítása, mely kihasználja az eszköz heterogenitását és a dinamikus különbségeket az adatsorok attribútumai között. Egy adaptív és teljesen testreszabott feldolgozóegységet ajánlanak, mely felhasználja az FPGA rendelkezésre álló heterogén erőforrásait, a megoldandó feladat igényei szerint alakítva azokat. Az alkalmazás eredményei bizonyítják a heterogén architektúra hatékonyságát, a CPU-s alkalmazáshoz képest két-háromszoros sebességet érve el. Az ajánlott architektúra hétszeresen jobb bármely más FPGA és grafikus processzor alapú megközelítésnél [104].

A SpinNaker egy új, ARM processzor alapú chip, mely nagyméretű spiking neurális hálók szimulációjára lett kifejlesztve. Néhány tulajdonsága alapján, több SpinNaker chip összekapcsolásával, nagy párhuzamosságú rendszerek alakíthatók ki. A fő cél egy 109 neuronból álló háló szimulálása, ami azt jelenti, hogy egy hasonló méretű biológiai neuronháló ugyanolyan sebességgel működne. Ebben a munkában le van írva ezen rendszerek leképzése, és az, hogy miként használhatók ki az áramkör erőforrásai [104].

Természetazonos felépítésű mesterséges idegsejtháló megvalósításáról a dolgozatban kapunk részletes betekintést. Az első fázisban szoftveres szimulációval történt a háló implementálása, majd hardveralapú megvalósításra is sor került [105], [106], [107].

Horváth Gábor különböző típusú CMAC (Cerebellar Model Articulation Controller) hálók újrakonfigurálható áramkörön való hatékony megvalósításáról számol be. A tervek Handle-C-ben voltak megvalósítva és hardver-eszközként Virtex alapú FPGA áramköröket alkalmazott [108].

Korábbi kutatásaink során több típusú neurális háló hardveres megvalósításával foglalkoztunk: CMAC, RBF, Kohonen, valamint MAMDANI típusú Fuzzy következtető rendszert [109] is sikeresen implementáltunk és különböző feladatokban alkalmaztunk. A CMAC típusú megvalósítások esetében a neurális háló leírása VHDL nyelven volt megvalósítva, a paraméterek kódolására egész számként való ábrázolás volt alkalmazva. A tanító algoritmus is az FPGA áramkörben volt megvalósítva. A háló függvényapproximációs és szabályozási feladatok [110], [111], [112] területén volt tesztelve. A CMAC architektúrája nagyon masszív párhuzamosságot mutat, így egy nagyon gyors rendszer létrehozását tette lehetővé. A kimenetszámítás és a tanítás egy adott bemeneti pontra szekvenciálisan van végrehajtva és összesen 8 órajelet igényel a kitűzött feladat megoldása érdekében. Az RBF háló nagyon hasonlít a CMAC hálóra, azonban van egy lényeges különbség a bázisfüggvények elhelyezéséből adódóan. A CMAC háló esetében a bázisfüggvények több rétegre vannak szétosztva, a rétegek párhuzamosan processzálhatók, és egy adott rétegen belül nem fedik át egymást a bázisfüggvények. Az RBF háló esetében a bázisfüggvények egy rétegben vannak elhelyezve és átfedődnek, amiből adódóan csak szekvenciálisan processzálhatóak. Az FPGA áramkörben megvalósított RBF hálóra is egy hardverben implementálható pipeline architektúra volt tervezve. Az RBF esetében is a tanítás az áramkörben van megvalósítva. Ellentétben a CMAC hardveres megvalósításával, az RBF esetében a tanítás párhuzamosan történik a kimenetszámítással, csak egy eltolás van a kimenetszámításra és a tanításra alkalmazott mintáknál. Az RBF két változatban is megvalósításra

került VHDL hardverleíró nyelven és System Generator [113] környezetben. VHDL-ben való implementálásnál egész számos számábrázolást alkalmaztunk, míg a SystemGenerator környezetben fixpontos aritmetikát [107]. Az RBF és CMAC hálók részletes bemutatásáról és összehasonlításáról a [114] dolgozatban olvashatunk.

A Kohonen-háló hardveres magvalósítását utazó ügynök típusú feladat megoldására probáltuk alkalmazni. A rendszert szintén VHDL hardverleíró nyelven valósítottuk meg. A Kohonen-háló esetében a kialakított hardveregység IP core magként volt integrálva egy Xilinx MicroBlaze^{TM} beágyazható soft-processor mag [115] sínrendszerére, hasonlóan az RBF hálós System Generator-os megvalósításhoz. A Kohonen-háló esetében párhuzamosan több fizikai neuron volt kialakítva, melyen szekvenciálisan történt a logikai neuronok feldolgozása. Hasonlóan az RBF megvalósításhoz, a kimenetszámítás és a tanulás párhuzamosan futottak. A párhuzamos tanítás alapját a Dual portos BRAM memóriák alkalmazása tette lehetővé, a memória egyik portján volt megvalósítva a kimenetszámítás, míg a második porton a súlytényezők frissítése ugyanarra az órajelre [37].

A hardveresen megvalósított CMAC, RBF és Kohonen-hálók struktúrája és három esetben a kimenetszámításra és a súlytényező-adaptációra kialakított pipeline csatornák a [116] dolgozatban van részletezve. Mindhárom megvalósítás esetében a hardveres neurális rendszerek paraméterei rugalmasan állíthatóak voltak (súlytényezők kódolására alkalmazott bitek száma, neuronok száma és a különböző hálóknak megfelelő specifikus paraméterek). A kialakított háló méretét általában a rendszer tervezésének pillanatában rendelkezésre álló FPGA áramkör mérete és erőforrásai jelentették. A súlytényezők és a bázisfüggvények tárolása BRAM típusú memóriában volt megvalósítva, és a legtöbb esetben a háló mérete szempontjából a korlátot az FPGA áramkörben található BRAM memóriák, illetve szorzó áramkörök vagy DSP modulok száma jelentette. A jelenleg rendelkezésre álló FPGA fejlesztőrendszereken jóval nagyobb méretben implementálhatóak az említett CMAC, RBF és Kohonen-hálók.

12.4. Hardvermegvalósíthatóság szempontjából fontos elemek tanulmányozása

A neurális hálózatok masszívan párhuzamos jelleget mutató szerkezete alkalmassá teszi ezeket újrakonfigurálható áramkörökben való megvalósításra. A neurális hálók hardverben való megvalósítása szempontjából több kérdés merül fel:

- Hogyan ábrázoljuk az adatokat a hardverben megvalósított neuronhálók esetében?
- Mekkora a neuronháló mérete (hány neuront tartalmaz)? Mennyire párhuzamosítható az algoritmus? Milyen adatfüggőségek jelentkeznek a kimenetszámítási és tanulási fázisokban? Érdemes-e a tanulási algoritmust az újrakonfigurálható áramkörben megvalósítani?
- Mekkora egy neuron környezete, receptív mezője?
- Milyen felbontásra, bitszélességre van szükség a különböző típusú adatok ábrázolására?
- Hogyan legyenek megvalósítva a szinaptikus kapcsolatok és hol legyenek tárolva?
- Hogyan implementálhatóak hatékonyan az aktivációs függvények? Milyen típusú aktivációs függvényt alkalmazzunk?
- Milyen architektúrát érdemes választani a neurális háló megvalósítására?
- Milyen technikák alkalmazhatóak a kialakítandó neurális háló processzálási sebességnövelésének és az FPGA áramkörben elfoglalt felület csökkentésének optimalizálására?
- Milyen módszerek alkalmazhatóak az újrakonfigurálható hardverben megvalósított neurális háló hibakeresésére és hatékony tesztelésére?

12.4.1. Adatok ábrázolása

A fejezet ezen részében a hardveres neurális számításban használt aritmetika osztályozását és hatásukat a felhasznált FPGA erőforrásokra nézve próbáljuk meg áttekinteni. Mivel a neurális háló algoritmusok végrehajtása (processzálása) számos aritmetikai műveletet igényel, a megfelelő aritmetika és adatábrázolási formátum kiválasztása fontos szerepet játszik egy neurális hardver tervezése során.

Hogyan lehet egyensúlyt kialakítani egy numerikus számítás még elfogadható pontossága (bitek száma), ami fontos a neurális háló működése

szempontjából, valamint a fokozott pontossággal járó nagyobb logikai felületből származó többletköltségek között? Hogyan válasszuk ki a megfelelő adatábrázolási formátumot, amelynek számrendszer-dinamika tartománya elég nagy ahhoz, hogy egy általános célú alkalmazás esetében biztosítsa a szaturáció elkerülését [117]?

A hardveralapú neurális háló teljesítménye nagymértékben függ az alkalmazási területtől és a különböző jelek ábrázolására használt pontosságtól. Mielőtt elkezdenénk egy FPGA alapú neurális háló hardveralapú tervezését, figyelembe kell venni a tárolási formátumot és pontosságot, amelyet a különböző paraméterek, bemenetek, kimenetek, súlytényezők, aktivációs függvények, aktivációs függvények deriváltjának és a hibának a kódolásánál alkalmazunk. Az adatok ábrázolására kiválasztott pontosságot általában a tanítási fázisban megkövetelt pontosság határozza meg.

A pontosságnak nagy hatása van a tanítási ciklusra, és a kisebb pontosság alkalmazása elegendő a kimenetszámítási fázisban. A lebegőpontos számábrázolás nyújtja a legszélesebb számrendszer-dinamika tartományt, alkalmas minden alkalmazásra. Több dolgozatban 32 bites lebegőpontos aritmetikának az alkalmazhatóságát tanulmányozták neurális hálózatok FPGA áramkörben való megvalósítására [118], [119]. Egyik fontos vitatott kérdés a neurális hálók hardverben való megvalósítása során meghatározni az adatok ábrázolási formátumának pontosságát, amely biztosít egy optimális kompromisszumot a pontosság és az erőforrás-szükséglet között, mely az implementáció során adódik. Az egyszeres pontosság vagy dupla pontosságú ábrázolás minimalizálja a kvantálási hibát, miközben jelentős hardvererőforrásokat igényel. Kevésbé pontos fixpontos ábrázolás kevesebb hardvererőforrást igényel, azonban növeli a kvantálási hibákat, ami gondot jelent a tanítás során, különösen a regressziós feladatok megoldásában.

A dolgozatban [120] egy MLP (többrétegű perceptron) neurális hálózatot valósítottak meg FPGA áramkörben, amely során fix, illetve lebegőpontos aritmetikát egyaránt alkalmaztak. Az eredmények azt mutatják, hogy az MLP fixpontos megvalósítása 12-szer gyorsabb a működési sebességet tekintve, 13-szor kisebb területet igényelt, lényegesen nagyobb feldolgozási sűrűség érhető el, összehasonlítva a lebegőpontos megvalósítással.

A hardveresen megvalósított neuronhálókban az adatok ábrázolásának szempontjából a lebegőpontos számábrázolással érhető el a legnagyobb pontosság. Az FPGA áramkörök lehetővé teszik a lebegőpontos aritmetika használatát, de jelentős számú erőforrást igényelnek az FPGA áramkör erőforrásaiból. A pulzuskódolt aritmetika az adatok továbbítására jóval kevesebb vezetéket (jelet) igényel, de mivel az információ továbbítása sorosan

történik, csökkenti a processzálási sebességet. Számításba véve a szükséges hardvererőforrásokat és a processzálási időt, levonható a következtetés, hogy a fixpontos aritmetika a leginkább ajánlott megoldás az adatok ábrázolására a neurális hálókban. Az alap aritmetikai műveletek általában kevés erőforrást igényelnek, de ami lényeges, hogy az FPGA áramkörökben dedikált egységek vannak beágyazva. Például a Spartan3, Spartan3E, Virtex II Pro dedikált MUL18x18 szorzó áramköröket tartalmaznak. A MULT18x18 előjeles 18-bites szorzó modulok, ezekből fejlődtek ki a mai DSP blokkok.

A fejlettebb újrakonfigurálható áramkörök dedikált DSP-egységeket tartalmaznak. A neuronhálókban általánosan kijelenthető, hogy az egyik alapvető művelet két vektor skalárszorzata, minden művelet visszavezethető szorzás és összeadásra (MAC). Ilyen egységek könnyen kialakíthatóak mind a MUL18×18, mind a DSP primitívekkel. A neurális háló implementálása során a hardver kiválasztásakor fontos tényező az áramkörben található dedikált szorzó áramkörök vagy DSP primitíveknek a száma.

A neurális hálók hardveres megvalósítása során egy ábrázolási mód létrehozásának/alkalmazásának a célja megfelelő pontossággal kódolni az adatokat, ami biztosítja a háló konvergenciáját és a felhasznált erőforrásokat az FPGA-ban rendelkezésre álló korláton belül tartja. Az adatok ábrázolására több megoldás alkalmazható:

- valós szám egész számként történő ábrázolása,
- fixpontos ábrázolás,
- lebegőpontos ábrázolás,
- impulzussorozat,
- aritmetikai léptetés (jobbra, balra).

12.4.1.1. Fixpontos ábrázolás

A fixpontos ábrázolás a leggyakrabban választott megoldás a neurális hálók hardveres megvalósítása során. A fixpontos aritmetika a felhasznált hardvererőforrások tekintetében jóval hatékonyabb, összehasonlítva a lebegőpontos formátummal. Mivel sok képfeldolgozási alkalmazás és neurális háló korlátozott alkalmazási területen kisebb pontossággal is elfogadhatóan működik és a lebegőpontos formátum nagyobb félvezető felületet igényel, a kutatók széles skálán alkalmazzák a fixpontos ábrázolást.

12.4.1.2. Lebegőpontos ábrázolás

A lebegőpontos formátum alkalmazása az adatok széles tartományban és nagy felbontással való ábrázolását teszi lehetővé, ám az FPGA áramkörből felhasznált felület egyáltalán nem elhanyagolható. Az FPGA áramkörök megjelenésének kezdetén, az áramköröket tekintve szerényebb erőforrásokkal, kisebb kapacitással (logikai kapuk száma) rendelkeztek. A lebegőpontos formátumot sem igazán támogatták a tervezőeszközök. A múltban a legtöbb kutató elkerülte a lebegőpontos aritmetika alkalmazását a nagyobb kapacitású FPGA áramkörök hiányának következményeként.

Például a [121] dolgozatban a backpropagation algoritmus megvalósítása során több konfigurációban is alkalmazták a lebegőpontos ábrázolási formátumot. A tanítási együtthatónál az exponenst 4 biten ábrázolták, az aktivációs értékek kódolására 3 bites exponenst, míg a gradiens értékek tárolására 5 bites exponenst alkalmaztak. Először a szerzők szoftveresen implementálták és validálták a hálót. A szerzők azt állítják, hogy hardveresen is könnyen megvalósítható volt.

Sahin és társai [117] tanulmányozták, mennyire flexibilisen alkalmazható a lebegőpontos aritmetika az FPGA áramkörökben. A 2000-es évek elején lebegőpontos összeadó, kivonó és szorzó egységeket terveztek VHDL-ben FPGA Xilinx XC4044XL áramkörre. A működési sebesség tekintetében 5-ször gyorsabban működtek, mint egy Pentium II 300MHz számítógépen szoftveresen végrehajtott műveletek.

12.4.1.3. Bit-soros aritmetika

Soros műveletek esetén az operandusok bitjein sorra, a legkisebb helyértéktől kezdve végezzük el a műveletet, míg a párhuzamos aritmetika esetében az összes biten egyszerre végezzük. A soros műveletet végző egységek kevés erőforrást alkalmaznak, de jóval lassabbak. Sorosan implementált műveletek több órajelet és csővezetékes kialakítást igényelnek. Ezt a módszert több neuronháló FPGA áramkörben való megvalósítása során is alkalmazták, mint például a klasszikus RRANN [122] RENCO architektúrák esetében. Ez a módszer lehetővé teszi több processzáló elemnek a kialakítását egy kisebb méretű FPGA áramkörben is.

12.4.1.4. Pulzuskódolt aritmetika

A pulzuskódolt neuronok alapötletét a biológiai neuron működése jelentette. A neuronba továbbított jelek kódolására több módszert is alkalmaznak:

- PWM – impulzus szélesség moduláció
- PCM – pulzus kód moduláció
- PFM – pulzus frekvencia moduláció

A PCM neuronokban a szorzás úgynevezett váltakozó órajelek kapuzásával van megvalósítva. A váltakozó órajelek jel/szünet aránya 1:1, 1:2, 1:4. Az órajelek egymáshoz képest szinkronizálva vannak, míg a bemeneti jelhez viszonyítva aszinkronok. Minden órajel logikai '1'-es szintje engedélyezi a bemenetnek az összegzőbe való jutását, míg az alacsony szint tiltja. Az összeadás a bitfolyamok egyszerű vagyolásával valósítható meg, a kimeneten a logikai '1'-esek valószínűsége arányos a jel értékével.

A pulzus frekvencia moduláció egy kódolási eljárás, amelynek során a minták amplitúdója impulzussorozat ismétlődési frekvenciáját módosítja. Ezzel a módszerrel kódolt jelek egyszerű logikai kapuk alkalmazásával összeadhatók és összeszorozhatók. Impulzus alapú neuron architektúrát dolgoztak ki [123], amelyben PFM kódolást alkalmaztak.

12.4.1.5. Szorzás megvalósítása eltolással

Ez a módszer a számokat 2 hatványainak függvényében ábrázolja. A szorzás és osztás műveletek egyszerű eltolással vannak implementálva. Molz és társai [124] az említett módszert alkalmazták, egy aktivációs függvény be- és kimeneteinek a kódolására 4 bitet alkalmazott.

Skrbek [78] „eltolás összeadás"-ra épülő neurális aritmetikát javasolt, teljes függvényhalmazzal, amely alkalmas az MLP FPGA áramkörben való megvalósítására. A javasolt aritmetika a $2\times$ és $\log(2\times)$ műveletekre alapoz. A függvények a megvalósítás során kombinálják az eltolási műveleteket a lineáris approximációval.

12.4.1.6. Online aritmetika

Boubaker és társai az LVQ algoritmust implementálták FPGA áramkörön online soros aritmetikát alkalmazva. Kielégítő teljesítményről számolnak be az online soros aritmetikát alkalmazó megvalósítás során elért eredményekről a sebesség, késleltetések, illetve az igényelt erőforrásokat figyelembe

véve, sikerült csökkenteni az áramkör méretét és az LVQ algoritmussal kompatibilis operátorokat tervezniük [125].

A számok ábrázolására redundáns módszert alkalmaz, ami nagyon gyors aritmetikai műveletvégző egységek megvalósítását teszi lehetővé. Ez az ábrázolási forma is soros aritmetikára épül, és a bit-soros aritmetika összes előnyét ötvözi, de jóval nagyobb processzálási sebesség mellett. Először az MSB magasabb helyértékű biteken végzi a műveletet, szemben az egyszerű bit-soros feldolgozással, amely először a legkisebb helyértékű biteket dolgozza fel. Girau és Tisserand [126] részletesen tárgyalja az összeadás, szorzás, tangens hiperbolikus függvény megvalósítását online aritmetikát alkalmazva, redundánsan radix-2 formában ábrázolva az adatokat.

VHDL-ben a valós típus csak szimulációban használható, FPGA szintézis során nem. Két különálló könyvtárcsomag volt tervezve a neurális hálók FPGA áramkörben való megvalósításának céljából. Az egyik könyvtárcsomag *uog_fp _arith* (University of Guelph Floating-Point Arithmetic) támogatja az IEEE-754 szabványt az egyszeres pontosságú lebegő pontos formátumra. A második könyvtár uog_fixed_arith (University of Guelph Fixed-Point Arithmetic) támogatja a 16 bites fixpontos aritmetikát. Ez a két ábrázolási forma a szakirodalomban közölt korábbi eredmények alapján volt kiválasztva [127]. Kimutatták, hogy minimum 16 bites fixpontos aritmetikára van szükség ahhoz, hogy a hiba-visszaterjesztéses algoritmus konvergáljon [128].

A lebegőpontos aritmetika esetében kisebb pontosságot is lehet választani, de nagyon valószínű, hogy a jövőbeli VHDL lebegőpontos implementációk követni fogják a standard ábrázolási formát.

12.4.2. Számítási pontosság

A hardveresen implementált neurális hálók esetében az adatok ábrázolásánál alkalmazott pontosság befolyásolja az igénybe vett erőforrások számát. Kevés számú bit alkalmazása esetén az approximációs hiba jelentős, és nem biztos, hogy az előírt paramétereknek megfelelően megoldható az alkalmazás. A probléma főleg a tanító algoritmusok esetében merül fel, kevés biten való ábrázolás esetében az algoritmus konvergenciája nem lesz biztosítva [129]. Túl sok bit alkalmazása esetében megnövekszik az igényelt erőforrások száma. A különböző típusú adatok esetében érdemes külön is tárgyalni a buszok szélességét:

- a neurális háló be- és kimenetei,
- a neurális háló súlytényezői,

– az aktivációs függvényeket megvalósító egységek be- és kimeneti jeleinek kódolása.

A neurális háló bemeneti jeleinek felbontását nagyrészt a feladatban alkalmazott bemeneti érzékelők határozzák meg. A leggyakrabban a rendszer bemenetén analóg digitális átalakítókat alkalmaznak, bár a modern érzékelők már digitális kimenetet biztosítanak. Az érzékelők felbontásánál nem érdemes nagyobb felbontást alkalmazni, ám egy kevesebb bitszámon való ábrázolás adatveszteséghez vezet.

A kimenetek esetében szintén a feladat határozza meg a kívánt felbontást. Például szabályozási feladatok esetében egy PWM vezérlő van alkalmazva. A PWM vezérlő által várt felbontásnál szintén nem érdemes nagyobb felbontást alkalmazni. Az aktivációs függvények és a súlytényezők ábrázolásánál alkalmazott bitek száma meghatározható, ha kiindulunk a kimenettől és haladunk a bemenetek irányába. A kimenetek felbontásából a feladatok többségében meghatározható a többi jelnek a felbontása is.

Az aktivációs függvények kimeneti értékeinek a felbontása megegyezik a kimenetek felbontásával, hiszen a kimenetet az aktivációs függvények után kapjuk. A súlytényezők és az aktivációs függvények bemenetei pontosságának a megválasztásánál figyelembe kell venni azt is, hogy a tanító algoritmusok konvergenciája biztosított legyen. A neuronháló hardveres implementálása előtt érdemes egy szimulációt elvégezni, amely során meghatározható a minimális bitszám, amely mellett az elvárásoknak megfelelően működik a háló. Ha a súlytényezők és az aktivációs függvények tárolására BRAM memóriák vannak használva, a memóriák méretezése/konfigurálása összefügg a súlytényezők, aktivációs függvények felbontásával.

Az aktivációs függvények bemeneteinek felbontása meghatározza a BRAM memória címvonalak bitszélességét, míg az aktivációs függvények kimenetének kódolására szánt bitek meghatározzák a memória adatvonalainak bitszélességét.

12.4.3. Súlytényezők leképzése az FPGA különböző erőforrásaira

A neurális háló FPGA alapú megvalósítása esetében a súlyok tárolására több megoldás lehetséges:

– a konfigurálható logikai tömbökben kialakított D-tárolókból, ezt hívjuk distributed esetnek,
– osztott RAM memóriák,
– Block RAM memóriák,
– külső, az FPGA áramkörhöz kapcsolt memóriák.

Az osztott memóriák az FPGA áramkör konfigurálható tömbjeiben vannak. Nagyszámú neuront tartalmazó neuronháló esetében, ha a súlytényezőket osztott memóriában vagy regiszterekben tárolnánk, az áramkör erőforrásainak nagy részét a súlytényezők tárolása foglalná el. Nem hatékony az osztott megoldás, helyette a nagyméretű, összefüggő BRAM memóriát kell alkalmazni.

Ez a módszer biztosítja a legnagyobb rugalmasságot a súlytényezők párhuzamos hozzáférése szempontjából, és, elméletileg, a legnagyobb feldolgozási sebességet, összehasonlítva a többi megoldással. Az osztott memória vagy regiszterek alkalmazása esetében egyéb erőforrásokra van igény a súlytényezők inicializálása és kimentése céljából, ellenkező esetben minden alkalommal újra kellene tanítani a rendszert.

A BRAM memóriák alkalmazása egy elfogadható megoldás, nagyobb méretű hálók implementálását teszi lehetővé, összevetve az előbbi megoldással. Feltételezve, hogy mindegyik neuronhoz hozzá van rendelve egy BRAM memória, a súlytényezők BRAM memóriába való tárolása maga után vonja a szekvenciális processzálást egy neuronra nézve, de az összes neuron párhuzamosan működik. Az osztott memóriás megvalósítás esetében például a hiba-visszaterjesztéses algoritmus alkalmazása az adatfüggőség szempontjából nem jelent gondot a kimenetszámítás és a tanítás párhuzamos végrehajtása szempontjából. A BRAM memóriák alkalmazásával nem igazán valósítható meg a kimenetszámítás és a súlytényező-módosítás párhuzamos működése.

A külső memória alkalmazása nem egy életképes megoldás, mivel nem teszi lehetővé a neuronok párhuzamos működését. A súlytényezők és aktivációs függvények BRAM memóriában való tárolása esetében, ismerve a feladat megoldásához szükséges háló méretét, az FPGA áramkör megválasztásánál figyelembe kell venni az áramkörben megtalálható BRAM memóriák és szorzó áramkörök vagy DSP modulok számát.

12.4.4. Aktivációs függvények hardveres megvalósítása

Hogyan implementálható az aktivációs függvény (milyen típusú aktivációs függvényt használunk)?

- Taylor-sorbafejtés,
- kereső táblázat,
- CORDIC algoritmus,
- szakaszonkénti lineáris megközelítés.

Az aktivációs függvények differenciálhatóak kell legyenek, tehát a backpropagation-alapú háló esetén nem használhatunk nem folytonos aktivációs függvényeket. Két, a backpropagation-alapú hálókban gyakran használt folytonos, nemlineáris aktivációs függvény a logisztikus függvény (12.1) és a hiperbolikus tangens (12.2).

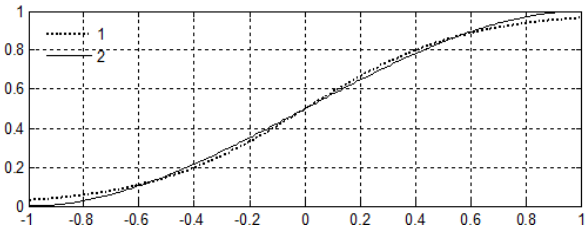
$$\varphi(x) = \frac{1}{1 + e^{-ax}} \tag{12.1}$$

$$\varphi(x) = \frac{e^{ax} - e^{-ax}}{e^{ax} + e^{-ax}} \tag{12.2}$$

Elemi vagy majdnem elemi függvények approximációjára számos módszert ismerünk, mint a polinomiális approximáció, CORDIC algoritmus, kereső táblázat alapú approximáció, és sok más [130], [131]. A függvények hardverimplementációja során a legfontosabb tényezők a pontosság, a teljesítmény és az erőforrásigény.

12.4.4.1. Taylor-sorbafejtés

Ezzel az eljárással megvalósítható a szigmoid függvény approximációja (12.2. ábra), egy Taylor-sor első négy-öt elemét használva (12.3). A [132] cikkben a következő képletet használják a szigmoid függvény approximációjára a [-1, 1] tartományon belül.



12.2. ábra. A szigmoid függvény Taylor-megközelítése

$$\varphi(x) = 1/2 + 3/4x - 1/4x^3 \tag{12.3}$$

12.4.4.2. Kereső táblázattal való megvalósítás (LUT)

A kereső táblázatok használata egy egyszerű megoldás az aktivációs függvény értékeinek tárolására, az értékek áramkörön belüli BRAM vagy

külső RAM típusú memóriákban vannak eltárolva. Az aktuális függvényértékek a táblázatból cím alapján olvashatók ki. A címek generálása megoldható egy egyszerű komparátorokból és léptető regiszterekből álló áramkörrel. Ennek a megoldásnak az előnye az alacsony számítási költség, viszont nagyméretű memóriaterületet igényel a LUT létrehozásához.

A cikkben egy Xilinx Virtex-5 típusú FPGA áramkörön implementált backpropagation alapú többrétegű perceptron háló VHDL nyelven való implementálásáról számolnak be. Az alkalmazásban kétféle megoldást valósítottak meg, az egyikben LUT-okat használtak, a másikban pedig lineáris interpolációt alkalmaztak a LUT-ban tárolt értékekre. A lineáris interpoláció módszere azt jelenti, hogy gyakorlatilag összeköti egy egyenessel a LUT-ban tárolt értékeket, úgy, hogy az aktivációs függvény a (12.4) szerint alakul.

$$y = \frac{LUT(d+1) - LUT(d)}{2^{N-M}} q_e + LUT(d) \tag{12.4}$$

ahol LUT(d) az aktuális indexnek megfelelő érték a táblázatban, N a bitszélessége az egyes mezőknek, M a bitszélessége a LUT címvonalak, q_e pedig a kvantálási hiba (12.5).

$$q_e = \varphi(v) - LUT(d). \tag{12.5}$$

12.4.4.3. CORDIC-algoritmusra épülő módszer

A koordináta geometriai módszer (CORDIC – Coordinate Rotational Digital Computer) aritmetikai és bit szintű műveletekre épül. Az implementálás eléggé nagy területet igényel a szilíciumlapkán. Az approximációs hiba a bitek számától függ, amelyekkel az aktivációs függvény értékei vannak kódolva. A CORDIC módszer a következő esetekben alkalmazható [133]:

- térbeli és polárkoordináta-rendszerek közötti transzformáció,
- trigonometriai függvények számítása,
- hiperbolikus függvényszámítás,
- négyzetgyökvonás.

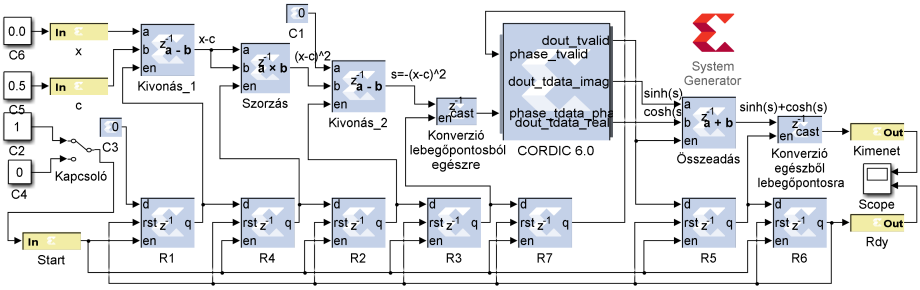
A tanulmányban 18 különböző típusú neuront terveztek és implementáltak, 2, 4 és 6 konstans és változó bemenettel változatban, valamint minden változatot három típusú aktivációs függvényre. A használt aktivációs függvények a radiális-bázisfüggvény, a szigmoid, valamint a tangens hiperbolikus függvények [134]. A függvényekben az a közös, hogy mindegyikben ki kell számolni e^x értékét. Az e^x kiszámítására (12.6) számos módszer van a szakirodalomban [135], mint például a kereső táblázatok használata.

Rendszertechnikai váza az e^{x-c^2} függvénynek a 12.4. ábrán, valamint Xilinx System Generatora által implementált blokk design a 12.3. ábrán van részletezve. A XILINX System Generatorhoz a Matlab Simulink csomagra is szükség van. A tanulmány célja az volt, hogy bemutassák az exponenciális aktivációs függvények használhatóságát a napjainkban hozzáférhető FPGA áramköröket használva. A tanulmány eredményei bebizonyították, hogy a legkisebb Virtex-6 típusú FPGA-n akár 10 neuron is implementálható, úgy, hogy akár 405MHz-en működjön.

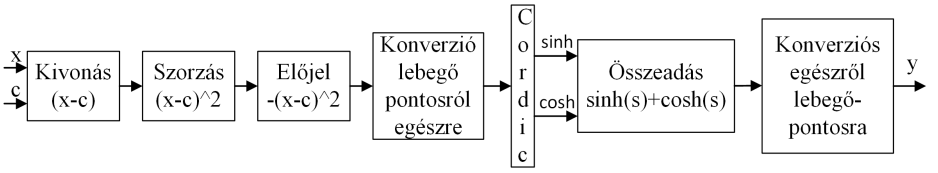
Az aktivációs függvényekben szereplő exponenciális tag kiszámítására egy CORDIC alapú rendszert terveztek, az alábbi egyenletben látható az exponenciális és a trigonometrikus függvények kapcsolata. A CORDIC képes kiszámolni a hiperbolikus szinusz és hiperbolikus koszinusz függvények értékét a $[-\pi/4, \pi/4]$ tartományban. A sinh(x) és cosh(x) kimenetek értékeit egy fixpontos összeadó áramkörrel adták össze.

$$e^x = sinh\,(x) + cosh\,(x)\,. \tag{12.6}$$

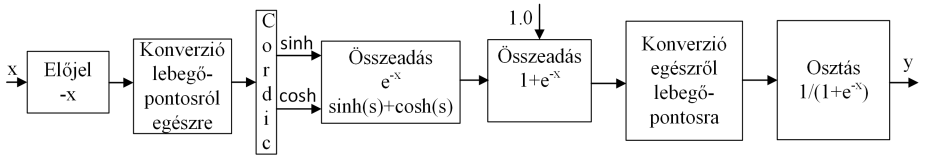
A 12.4, 12.5, 12.6. ábrákon a rendszer tömbvázlata látható a három típusú aktivációs függvényre: Gauss, logisztikus és tangens hiperbolikus aktivációs függvények. Mindhárom rendszernek 32 bites, lebegőpontos a bemenete és a kimenete. Mivel a CORDIC csak fixpontos műveleteket támogat, ezért fixpontossá kell alakítani az adatokat előtte, és visszaalakítani utána. Az átalakítások miatt az eredeti 8 tizedes pontosság helyett a rendszer pontossága lecsökken 6 tizedesre, de ez még elfogadható neuronhálós alkalmazások esetén.



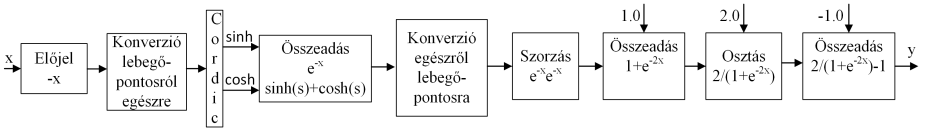
12.3. ábra. RadBas aktivációs függvény tömbvázlata



12.4. ábra. RadBas aktivációs függvény tömbvázlata



12.5. ábra. LogSig aktivációs függvény tömbvázlata



12.6. ábra. TanSig aktivációs függvény tömbvázlata

12.4.4.4. Szakaszonkénti lineáris megközelítés

A módszer lényege az, hogy az aktivációs függvények értékeit kisebb tartományokon belül lineárisan közelítsük meg. Ez a megközelítési módszer megfelelőnek bizonyul a hardverkövetelmények és a pontosság szempontjából. A szakaszonkénti megközelítések lehetnek elsőrendűek vagy magasabb rendűek.

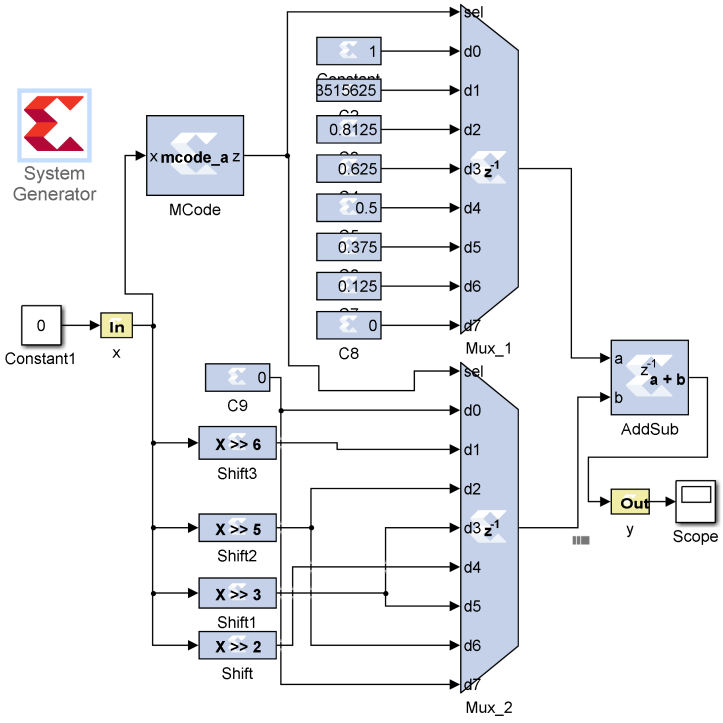
A szakaszonkénti lineáris megközelítés (PWL – Piecewise Linear Approximation) megoldást biztosít a függvények approximációjára alacsony hibaértékek és alacsony számítási igény mellett. A következő megközelítő módszerek egyike sem tartalmaz szorzást, ami egy nagy pozitívum a

hardverimplementálásnál. Mivel a szigmoid függvénynek van egy szimmetriapontja (0, 0,5)-ön, ezért elég az $x-y$ pároknak a felét kiszámolni, a másik felét a (12.7) képlettel lehet meghatározni:

$$y_{x<0} = 1 - y_{x\geq 0} \tag{12.7}$$

A-szabályon alapuló megközelítés

Myers és Hutchingtons tervezett egy megközelítést az A-szabály alapján [136]. Egy módosított görbe lett kifejlesztve úgy, hogy minden vonalszegmensnek a gradiense kettő hatványa. Ez lehetővé teszi, hogy shift regisztereket használjunk szorzóáramkörök helyett. A görbének hét törési pontja van (12.1. táblázat), System Generator alapú megvalósítása a 12.7. ábrán van szemléltetve.



12.7. ábra. A-szabály alapú approximáció összehasonlítása különböző fixpontos ábrázolási változatokra

12.1. táblázat. A-szabály 7 törési pont

x	-8,000	-4,000	-2,000	-1,000	1,000	2,000	4,000	8,000
y	0,0000	0,0625	0,1250	0,2500	0,7500	0,8750	0,9375	1,0000

Alippi-, Storti-Gajani-féle megközelítés

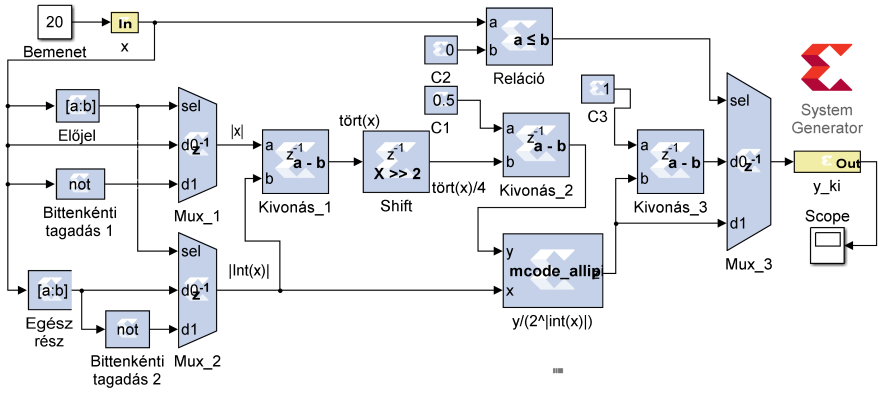
Alippi és Storti-Gajani a megközelítését egy egész számú törési pont kiválasztására és az y értékeket pedig kettő hatványaiként való meghatározására alapozta. A publikációik voltak az elsők, amelyek bemutatták a szigmoid függvény approximációját PWL módszert alkalmazva. A szerző sok munkát fektetett be a szigmoid függvény egyszerűsítésébe, egy olyan modell elérése érdekében, amely implementálható VLSI környezetben. A [137] publikáció a szigmoid függvény approximációját mutatja be PWL módszerrel, úgy, hogy adott számú x-re kiszámítja az y-t, a 2 hatványait felhasználva. A módszer szerint (12.8) minden intervallumra csak egy szegmenst veszünk figyelembe, tehát a függvény a következő képlettel írható le és linearizálható:

$$y\left(x\right) = 1 + \frac{1}{2^{|(x)|}}\left(\frac{\hat{x}}{4} - \frac{1}{2}\right) \tag{12.8}$$

$\hat{x}$: az x szám tizedes része
(x): az x szám egész része
Ha csak a negatív tengelyt vesszük figyelembe, akkor a (12.9) szerint alakul.

$$y\left(x\right) = 1 + \frac{1}{2^{|(x)|}}\left(\frac{\hat{x}}{4} + \frac{1}{2}\right). \tag{12.9}$$

Az előbbi bemutatott változat hardveresen könnyen megvalósítható egy számláló által vezérelt léptetőregiszterrel, kiküszöbölve a szorzóáramkör igénybevételét. A [137] dolgozatban a szigmoid függvény a [-8, 8] intervallumban 15 lineáris szakaszra volt bontva. A szerzők [138] által javasolt dolgozat egy módosított görbe az A-szabály alapján. Mindkét dolgozatban a szigmoid függvény deriváltjának a szakaszonkénti approximációja is szemléltetve van. A sigmoid Alippi-féle megközelítés System Generator alapú megvalósítása a 12.8. ábrán van szemléltetve.



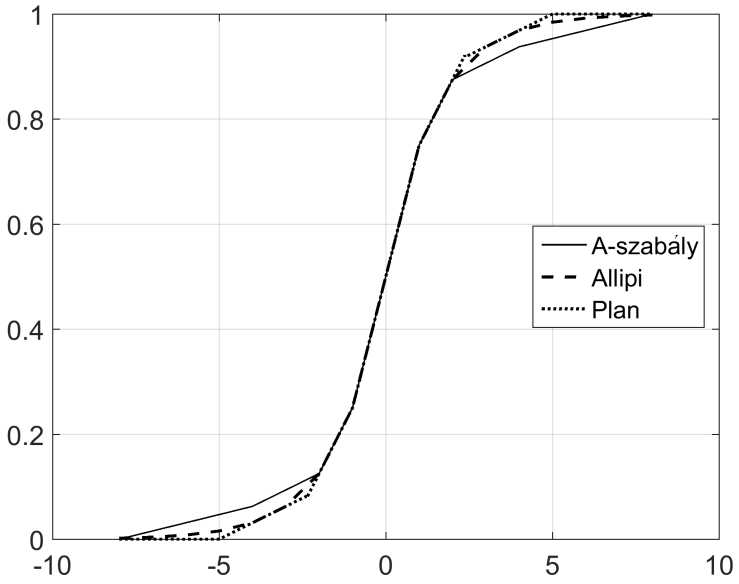
12.8. ábra. Allipi-approximáció hardverarchitektúrája

PLAN megközelítés

A PLAN (piecewise linear approximation of a nonlinear function) - féle megközelítést Amin, Curtis és Hayes-Gill javasolta (12.10). A PLAN megközelítés digitális kapukat használ arra, hogy az x értékeket közvetlen módon leképezze az y-ra. A megközelítést csak az x abszolút értékén kell végrehajtani.

$$f(x) = \begin{cases} 1 & \text{ha} \quad |x| \geq 5 \\ 0{,}03125|x| + 0{,}84375 & \text{ha} \quad 2{,}375 \leq |x| < 5 \\ 0.125|x| + 0{,}625 & \text{ha} \quad 1 \leq |x| < 2{,}375 \\ 0{,}25|x| + 0{,}5 & \text{ha} \quad 0 \leq |x| < 1. \end{cases} \tag{12.10}$$

A 12.9. ábrán a szigmoid függvény *A-szabály*, *Allipi* és a *Plan* módszerekkel való szakaszonkénti lineáris megközelítése vannak szemléltetve.



12.9. ábra. Szakaszonkénti lineáris megközelítés

12.4.4.5. Szakaszonkénti másodfokú megközelítés

A szigmoid függvényt meg lehet közelíteni másodfokú polinomokkal (12.11). Az egyértelmű hátránya: nem lehet elkerülni szorzás alkalmazását. Zhang, Vassiliadis és Delgado–Frias bemutattak egy szakaszonkénti másodrendű megközelítést, amely csak egy szorzást igényel. A [-4, 4] intervallumban a következőképpen néz ki a megközelítés:

$$f(x) = \begin{cases} \frac{1}{2}\left(\frac{x}{2^2} - 1\right)^2 & \text{ha} \quad -4 < x < 0 \\ 1 - \frac{1}{2}\left(\frac{x}{2^2} + 1\right)^2 & \text{ha} \quad 4 > x \geq 0 \end{cases} \tag{12.11}$$

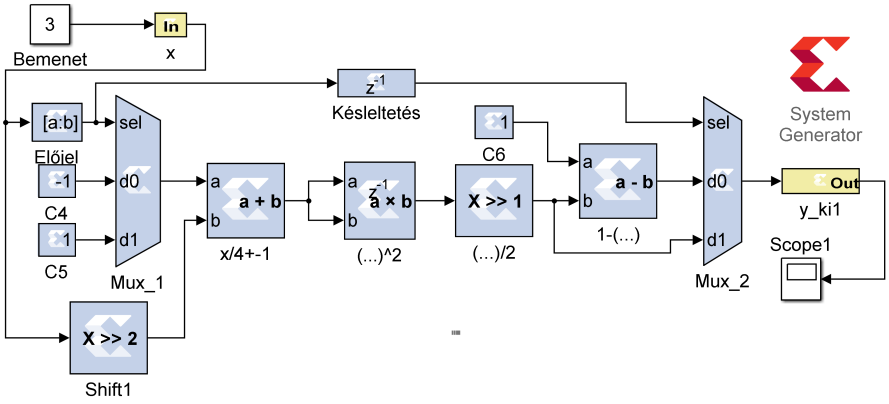
A 12.10. ábrán a Zhang-féle megközelítés van részletezve.

A [138] publikációban bemutatott modell javítja a függvényapproximációt a lineáris elsőrendű változathoz képest. Csökkenti az átlag hiba és a maximális hiba értékét, és mindössze egy szorzási műveletet igényel. E modell szerint a szigmoid függvény bemeneti értékei szegmensekre vannak felbontva és egy másodrendű approximáció van használva minden szegmens kiszámítására. Egy $[\alpha, \beta]$ szegmens esetén, ha $y(x)$ az approximált kimenet,

akkor a másodrendű approximáció eredménye a (12.12) szerint írható fel:

$$y(x) = A + B(x + C)^2, |C| = 2^{-n} \tag{12.12}$$

ahol C a 2 hatványa, tehát a szorzása elvégezhető eltolással.



12.10. ábra. Zhang-függvényapproximáció hardverarchitektúrája

A cikkben részletez egy algoritmust, amely alapján meghatározhatóak az A, B és C értékei. A szerző által használt képlet szerint a függvényt két szegmensre osztva közelíti meg (12.13):

$$f(x) = \begin{cases} \frac{1}{2}\left(1 - |\frac{x}{2^2}|\right)^2 & \text{ha} \quad -4 < x < 0 \\ 1 - \frac{1}{2}\left(1 - |\frac{x}{2^2}|\right)^2 & \text{ha} \quad 4 > x \geq 0. \end{cases} \tag{12.13}$$

A fenti modell használható előjelbites vagy kettes komplemens számábrázolással. Az alkalmazásban a számok a következő struktúra szerint vannak ábrázolva: 4 bites egész rész, egy előjel bit és 10 bit törtrész. A módszer alkalmazásával a maximális hiba értéke 0,0180, a hiba átlagértéke pedig 10^{-3} nagyságrendű, ami sokkal kisebb, mint az elsőrendű PWL alkalmazása esetén 10^{-2}.

12.4.4.6. Központosított rekurzív interpoláció (Centred recursive interpolation)

Basterrextea, Tarela és del Campo bemutattak egy rekurzív algoritmust a szigmoid függvény megközelítésére. Az algoritmus alapja a központosított

rekurzív interpoláció, amely rekurzívan próbálja növelni a pontosságot, minden újabb iterációval exponenciálisan növelve a lineáris szakaszok számát. A [139] cikk a szigmoid függvény és ennek a deriváltjának az implementálására mutat be egy vázlatot, az összpontosított rekurzív interpoláció módszerének alkalmazásával (CRI).

A CRI egy szekvenciális számítási módszer a PWL függvény meghatározására. A szerzők szerint a CRI modellen alapuló PWL approximáció alkalmas bármely nemlineáris függvény megközelítésére.

Az alkalmazott modell egy egyszerű három szegmenses approximáció, ahol két szegmens a kis és nagy bemenetek szaturációs értékeit képezi, a középső szegmens pedig a szigmoid függvény érintője $x = 0$ referenciapontban. Az approximációt a (12.14) egyenletek írják le.

$$\begin{aligned} y_1(x) &= 0 \\ y_2(x) &= \frac{1}{2}\left(1 + \frac{x}{2}\right) \\ y_3(x) &= 1. \end{aligned} \tag{12.14}$$

Az y tengelyen különböző $|x|$ értékekre húzott érintőpárok beiktatásával növelhető a szegmensek száma. Tehát a szegmensek száma 5, 9, 17, 33-ig növelhető $q = 1, 2, 3$ és 4 esetén. A [140] cikkben kijelentik, hogy az approximálási módszerekhez szükséges számítások eléggé bonyolultak. A szerzők egy sajátos szigmoid függvényt alkalmaztak (12.15):

$$f(n) = \frac{1}{1 + 2^{-n}}. \tag{12.15}$$

A fenti függvény és a hagyományos szigmoid között a különbség egy konstans skálázási tényező, amely a (12.16) összefüggés szerint kapható meg:

$$e^{-z} = 2^{\frac{-z}{ln(2)}}. \tag{12.16}$$

A fent említett módszerrel a maximális approximációs hiba értéke 0,0808. A [83] cikkben a [-8, 8] tartományban levő bemeneti értékekkel kísérleteznek, ahol 4 biten ábrázolják az egészrészt és 8 biten a törtrészt. A számított hiba, vagyis a különbség a folytonos és a kvantált függvény között a [-0,16, 0,16] intervallumban van.

Egyes szerzők egyszerűsített függvényváltozatokat használnak, mint a hiperbolikus tangens vagy a gyors szigmoid. A [140] cikk egy ilyen típusú módszert mutat be a szigmoid függvény deriváltjának számítására, de kiterjesztette a módszert más típusú aktivációs függvényekre is. Az említett

approximálási módszerek szigmoid függvényen való alkalmazása során fellép egy nullától különböző approximálási hiba. A neuronhálókat tekintve ennek az approximálási hibának nincs jelentősége, mivel a neuronhálót lehet tanítani, és ezek az approximálási hibák nem jelennek meg a háló kimenetén.

A fenti leírásokból kiderül, hogy a leggyakrabban tanulmányozott és implementált függvényapproximációs módszerek: a RAM-ban vagy ROM-ban létrehozott kereső táblázatok, a szakaszonkénti lineáris approximáció, magasabb rendű approximációk, Taylor-sorbafejtés, és más dedikált approximációs módszerek, mint a CORDIC módszer alkalmazása.

A CORDIC az FPGA-alapú neurális hálók hardveres implementációjában a leggyakrabban használt módszer. Az előnye az, hogy ugyanaz a hardver több típusú aktivációs függvény implementálására is alkalmazható, de teljesítmény szempontjából általában nem kielégítő.

A magasabb rendű polinomokra épülő approximációs módszerek a pontosságot tekintve kitűnőek, de a hardverimplementálás során a bonyolult aritmetikai műveletek miatt nehézségek merülhetnek fel, mivel nagyszámú hardveregységet igényelnek, vagy a teljesítmény nem lesz kielégítő.

A kereső táblázatok alkalmazásánál is valamilyen szinten hasonló probléma merül fel. Kisméretű táblázat alkalmazása esetén nem érünk el megfelelő teljesítményt, viszont a nagyméretű táblázatok használata költséges a hardvererőforrások szempontjából. Ezek a hardvererőforrásokkal kapcsolatos tényezők igen jelentősek az ASIC és FPGA áramkörök használata esetén.

A függvények implementálása polinomiális függvények kombinálásával nem egy új technika, viszont az a kérdés, hogy hogyan választjuk meg az interpolálási pontokat, úgy, hogy a kereső táblázat minél kisebb méretű legyen.

A polinomiális függvényekkel történő interpolációnak három jelentős előnye van: ugyanaz a hardverstruktúra használható több típusú függvény implementálására, csak a polinom együtthatói változnak (a táblázat tartalma); könnyen implementálható szorzó áramköröket, összegzőket és kereső táblázatokat tartalmazó FPGA áramkörökön, és alacsony számú logikai elem szükséges.

Az elvégzett kutatások fényében, az aktivációs függvények implementálása terén a kereső táblázatok használatára (az aktivációs függvény értékeinek eltárolása BRAM memóriában) vagy a szakaszonkénti lineáris approximációra szűkítjük a lehetőségeket. A szakaszonkénti lineáris approximációval kielégítő eredmények érhetők el a felhasznált erőforrások terén, és az approximálási hiba is elfogadható.

Abban az esetben, ha a rendelkezésre álló erőforrások nem teszik lehetővé, hogy minden neuron számára külön implementáljuk az aktivációs függvényt, alkalmazható egy pipeline struktúra. Ezzel a módszerrel az összes neuron használhatja ugyanazt az aktivációs függvényt. A neuronhálók FPGA áramkörökön való implementálása terén szerzett tapasztalatok alapján kijelenthető, hogy jelentős energiát kell befektetni a neuronháló tesztelésébe és a bemeneti, illetve kimeneti interfészek implementálásába.

A neuronháló interfészeinek tervezési és tesztelési idejét jelentősen lecsökkentheti a hardver coszimuláció alkalmazása. A neuronhálók hardver implementálásának szempontjából kutatásokat végeztünk RBF neuronhálók és feedforward típusú többrétegű neuronhálók terén. Tanulmányoztuk az RBF háló hardveres implementálását és alkalmaztuk terepmodellezési alkalmazásnál.

12.4.5. Különböző hardveres megvalósítások összefoglalása

Neuronhálós rendszerek tervezése során nagyon fontos az ilyen típusú hardverek jellemzőinek és változatainak az alapos ismerete. A neuronok száma direkt módon van meghatározva. A neuronok állapota tárolható áramkörön belüli (on-chip) vagy áramkörön kívüli (off-chip) tárolókban/memóriákban. A chipen belüli tárolás megvalósítható analóg vagy digitális formában. Analóg tárolás esetén az állapot kódolható: feszültségben (BELLCORE [141], ETANN), áramban (Catholic University of Louvain Processor Chip [142]) vagy frekvenciában.

A pontosság a bitek számától függ, amelyekkel a neuron kimenete van ábrázolva. A Hitachi digitális chipben 256 neuront implementáltak 8 bites pontossággal, és az állapotok tárolását a chipen belül valósították meg. A KAKADU esetében az állapotok a chipen kívül vannak tárolva ellenállásokat alkalmazva. Az RBF, CMAC, Kohonen [37], [116], [111], [112] esetén az állapotok az áramkörben vannak eltárolva, digitális formában.

A súlytényezők értékei is tárolhatók az áramkörön belül vagy ezen kívül. A SYNAPSE rendszerben a súlytényezők az áramkörön kívül vannak tárolva. Az analóg súlytényezők tárolására több módszer létezik: tárolhatók feszültség formában (Jet Propulsion Laboratory chip), elektromos töltés formájában (Mitsubishi Branch-Neuron-Unit, ETANN), vagy ellenállást alkalmazva (BELLCORE, Synaptics Object Recognizer Chip).

A digitális súlytényezők lehetnek statikusak vagy dinamikusak. A CMAC típusú rendszer esetén a súlytényezők regiszterekben vannak tárolva, valamint BRAM memóriákban [37], [111], [112], [116]. Az első

változatban, regiszterekben való tárolás esetén, csak kisméretű neuronháló hozható létre, mert az áramkör erőforrásainak nagy részét a súlytényezők tárolása és a bázisfüggvények tárolása veszi igénybe.

Az aktivációs függvények kiszámítása általában áramkörön belül történik, analóg vagy digitális megvalósításoknál egyaránt. Néha az aktiválási tömb kimeneténél vagy az áramkör bemenetein megjelenhetnek zajok, amelyek befolyásolhatják az aktiválási értéket, ami azt jelenti, hogy az eredmény probabilisztikus lesz. A Nestor Ni1000 [143] esetén az aktiválás kiszámítása digitálisan történik, míg a BELLCORE esetén analóg módon, a kimenetei probabilisztikusak.

Digitális megvalósítás esetén az átviteli függvényértékek kiszámítására használható kereső táblázat vagy elvégezhetők a számítások direkt módon is. Ha a megvalósítás analóg, akkor használhatók különböző áramkörök, mint például műveleti erősítők (EPSILON).

A saját RBF és CMAC rendszerekben az átviteli függvények áramkörön belül, regiszterekben vagy BRAM memóriában vannak tárolva, digitális formában. Abban az esetben, ha a neurális állapotok és a súlytényezők nem az áramkörön belül vannak tárolva, a processzálási sebesség lassúbb. A tömbök közötti információáramlás megvalósítható: szisztolikus tömb használatával [144], direkt kapcsolattal [145], [146] vagy pipeline technológiákkal [147].

A súlytényezők a tanítási folyamat során vannak számolva. Áramkörön belüli tanítás esetén a súlytényezők aktualizálása automatikusan az áramkörben történik. Áramkörön kívüli tanítás esetén egy dedikált rendszer számítja ki és aktualizálja a súlytényezőket.

A Lunero alkalmazásban a tanítás a chipen kívül van megvalósítva egy dedikált rendszer használatával. A BELLCORE rendszerben a tanítás a chipen belül hajtódik végre. A saját RBF és CMAC rendszerekben a tanítás az áramkörben történik, egy kontrollregiszter meghatározza, hogy aktualizálódjanak-e a súlytényezők vagy ne.

A tanítási idő alatt azt az időt értjük, ami alatt a rendszer kiszámolja és aktualizálja a súlytényezőket a tanítás során. Ez az idő megmutatja, hogy mennyire gyors a rendszer (CUPS). A végrehajtási idő a súlytényezőknek a bemenetekkel való összeszorzása és az átviteli függvény kiszámítása által igényelt időt jelenti, a mértékegysége a másodpercenként számolt kapcsolatokat jelenti. A CNAPS rendszer sebessége 1,6 CPS és 1,08 CUPS.

A kaszkád kapcsolás azt jelenti, hogy több hasonló chipet összekapcsolunk a neuronok számának növelése érdekében. Ha a kaszkád kapcsolás lehetséges, akkor a chipek kapcsolódhatnak lineáris sorozatban (CNAPS [148]), szisztolikus sorozatban (MA16), direkt módon (ETANN).

A SYNAPSE1 rendszerben az MA16 chipek szisztolikus mátrix formában vannak összekapcsolva. A saját tervezésű neuronhálókban megvalósítható a kaszkád kapcsolás és az implementált CMAC hálóból kialakíthatóak HCMAC (Hierarchical Cerebellar Model Articulation Controller) és GCMAC (Generalized Cerebellar Model Articulation Controller) típusú hálók. Az órajel-frekvenciája és az átviteli sebesség fontos tulajdonságai a hardveresen implementált neurális hálózatoknak. A CNAPS [149] 20MHz-en működik, 20 Mbyte/sec átviteli sebességgel. A saját fejlesztésű, FPGA áramkörön implementált rendszerek 100 MHz-es órajelen működnek. A CMAC háló egy bemenetsorozatot 80 ns alatt dolgoz fel, az RBF háló pedig a pipeline struktúra miatt 800 ns alatt.

A bemenetek, illetve kimenetek száma lehet rögzített, vagy újrakonfigurálható áramkörök esetén változtatható, az erőforrások határain belül. Általában a bemenetek száma szoros kapcsolatban áll a chipen implementált neuronok számával. Például az ANNA rendszernek 4096 súlytényezője van, a neuronok számának növekedésével (16, 64, 256) a bemenetek száma csökken (256, 64, 16).

A CMAC és RBF rendszerekben a bemenetek száma változtatható egy paraméter módosításával a VHDL kódban és a rendszer újrafordításával. Egy másik osztályozási kritérium a neuronháló bemeneteinek és kimeneteinek a típusa. A chip bemenetei, illetve kimenetei lehetnek digitálisak vagy analógok. Az ETANN ki/bemenetei analóg feszültségértékek, a CNAPS rendszerben pedig digitális értékek. A saját neuronhálók esetén a bemenetek és kimenetek digitális értékek.

13. fejezet

FPGA áramkörön megvalósított RBF neuronháló

Ebben a fejezetben az FPGA áramkörön megvalósított RBF típusú neurális hálót tárgyaljuk, többek között a hardvereszközön megvalósított neuronháló szerkezetét, az áramkörön megvalósított online tanulást, adatcsere kialakítását számítógép és az FPGA között.

13.1. Elméleti alapfogalmak

Egy RBF típusú FPGA áramkörön megvalósított neurális háló struktúráját és alkalmazását szemléltetjük. Az RBF háló egy nem lineáris bemeneti rétegből és egy aktív kimeneti rétegből áll. A bemeneti rétegben úgynevezett RBF típusú neuronok vannak. A háló bemenetei rávetítődnek a bázisfüggvényekre. Minden bázisfüggvényhez kapcsolódik egy súlytényező. A háló kimenetét a súlyvektor és a bázisfüggvények kimeneteiből számolt vektor lineáris kombinációja határozza meg (13.1):

$$y_i = \sum_{j=1}^{N} w_{i,j} g\left(\underline{x}, \underline{c}_i, \sigma_i\right). \tag{13.1}$$

Az RBF típusú hálók esetében több típusú bázisfüggvényt lehet alkalmazni, de a leggyakrabban a Gauss függvény használata terjedt el (13.2). A Gauss-függvény alkalmazása hardvereszközön implementált háló esetében nem a legelőnyösebb az elvégzendő szorzás és gyökvonás miatt. Sok logikai

elemet igényel a megvalósításhoz.

$$g(\underline{x}, \underline{c}_j, \sigma_j) = e^{-\frac{||\underline{x}-\underline{c}_j||^2}{2\sigma^2}}. \tag{13.2}$$

A háló bemeneti vektora és a bázisfüggvény középpontvektora közötti távolságot nem euklideszi norma szerint számoljuk, hanem egy egyszerűbb, ∞-norma (13.3) szerint:

$$\|\underline{x} - \underline{c}_i\|_\infty = max\left(|x_1 - c_{i1}|, |x_2 - c_{i2}|, |x_3 - c_{i3}|\right). \tag{13.3}$$

Ez alapján a háló kimenete a (13.4) és (13.5) szerint számítható:

$$y_i = \sum_{j=1}^{N} w_{i,j} e^{-\frac{||\underline{x}-\underline{c}_j||_\infty}{2\sigma^2}} \tag{13.4}$$

$$y_i = \sum_{j=1}^{N} w_{i,j} e^{\frac{-\max\limits_{k=1}^{M}\left(x_k - c_{jk}\right)}{2\sigma^2}} \tag{13.5}$$

g – bázisfüggvény,
w – súlyvektor,
y_i – a háló i-edik kimenete,
$\underline{x}$ – a háló bemeneti vektora,
$\underline{x} = [x_1, x_2, x_3]^T$, $\underline{c}_i$-edik bázisfüggvény középpont vektora.

A háló tanítása a delta-szabály szerint történik. Ha a rendszernek több kimenete is van, ebben az esetben a tanítási algoritmust a (13.6) képlet szemlélteti, ahol a (13.7) a bázisfüggvény a ∞-norma alapján.

$$w_{i,j} = w_{i,j} + \mu\delta_i g_\infty\left(\underline{x}, \underline{c}_j, \sigma\right) \tag{13.6}$$

$$g_\infty\left(\underline{x}, \underline{c}_i, \sigma\right) = e^{\frac{-\max\limits_{k=1}^{M}\left(x_k - c_{i,k}\right)}{2\sigma^2}}. \tag{13.7}$$

A rendszeren megvalósított tanító algoritmus a (13.8) képlet alapján történik, a tanítási együtthatót a (13.9) szerint határoztuk meg.

$$w_{i,j} = w_{i,j} + \frac{1}{2^{N_\mu}} sign\left(\delta_i\right) g_\infty\left(\underline{x}, \underline{c}_j, \sigma\right) \tag{13.8}$$

$$\mu = \frac{1}{2^{N_\mu}}. \tag{13.9}$$

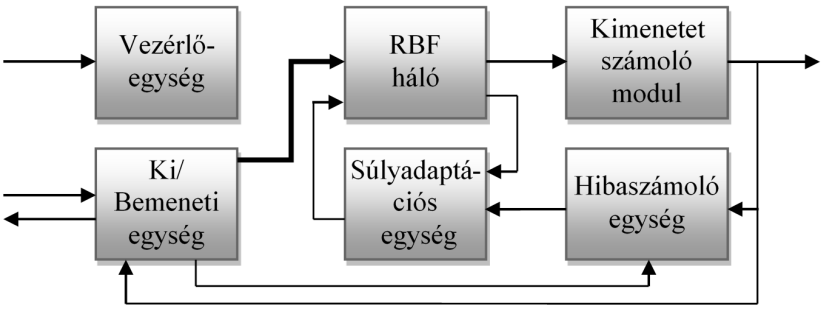
13.1.1. Az FPGA áramkörön megvalósított háló struktúrája

A hardveresen megvalósított RBF neuronháló tömbvázlata a 13.1. ábrán látható:

Vezérlőegység – szinkronizálja a rendszer moduljainak működését,

RBF háló – ez az egység tartalmazza a neurális hálót,

Ki/Bemeneti egység – ezen modul segítségével kapcsolódik a számítógéphez.



13.1. ábra. RBF hardveres megvalósításának tömbvázlata

Az FPGA áramkör a számítógép párhuzamos portjára van kapcsolva. A párhuzamos porton egy soros kommunikációs protokoll van implementálva, melyen keresztül vezérlőjeleket (írás, olvasás, start, reset) és két irányba adatokat lehet közvetíteni. Ezen az egységen keresztül kell felprogramozni a rendszer paramétereit, a súlytényezőket, bázisfüggvényeket, a háló bemeneteit és a bemeneteknek megfelelő előírt értéket, és visszaolvasni a háló kimenetét.

Súlyadaptációs egység – a tanító algoritmust megvalósító modul.

Hibaszámoló egység – az aktuális bemenetre és előírt értékre kiszámolja az előírt érték és a háló kimenete közötti hibát. A hibaszámítás és ennek megfelelően a háló tanítása is két üzemmódban működhet:

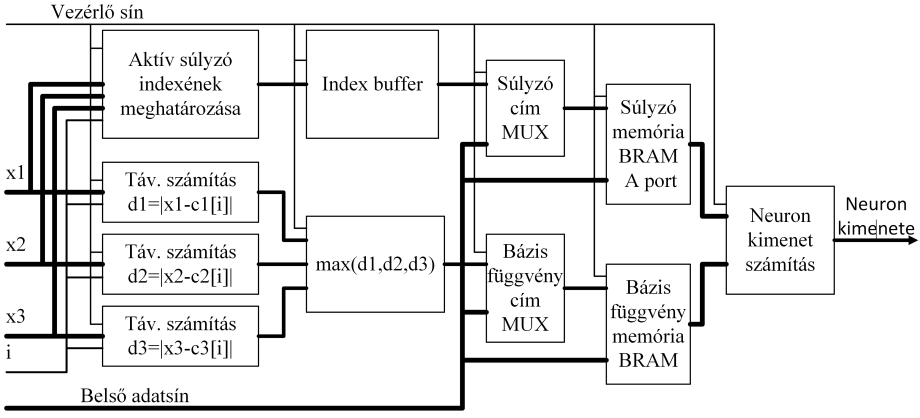
- a hibát az előírt érték és a háló kimenete között számoljuk (ezt a módot kell használni például approximációs, rendszer-identifikációs feladatok megoldásánál);
- a hibát egy szabályozandó rendszer előírt értéke és a rendszer kimenetén levő válasz közötti különbség alapján számoljuk. Ebben az üzemmódban a háló egy direkt szabályzóként működik.

Kimeneti egység – ez az egység számolja ki a háló kimenetét és továbbítja az eredményt a ki-bemeneti egységnek. A hálónak három bemenete

és egy kimenete van és összesen 4096 neuront processzál. Egy bemenetre (bemeneti vektorra) összesen 64 darab bázisfüggvény aktiválódik. Minden egyes bázisfüggvényhez kapcsolódik egy-egy súlytényező.

Hárombemenetű háló esetében a bázisfüggvények középpontját egy háromelemű vektor határozza meg. A bázisfüggvények a háromdimenziós térben egyenletesen vannak elosztva. A háló struktúrája egy 6 fázisú csővezetékes architektúrára alapul. Egy bemeneti elempárra a kimenet kiszámolásához 70 lépésre van szükség. 64 lépés a 64 darab aktív neuronból és 6 lépés a csővezeték hosszából adódik. A processzáló elem struktúrája az 13.2. ábrán látható:

1. a távolság kiszámítása a bemeneti vektor és a bázisfüggvény középpont-vektora között;
2. a bázisfüggvény értékének megcímzése és az aktív súlytényező címének a meghatározása;
3. a bázisfüggvény értékének és az aktív súlytényező értékének az előhívása;
4. a súlytényezőérték és a bázisfüggvény értékének összeszorzása;
5. a háló kimenetének frissítése (64 aktív bázisfüggvény van egy bemenetre, a kimenetet pedig egy soros összeadással valósítottuk meg; minden lépésben a kimenethez hozzáadjuk az aktuális súlytényező és bázisfüggvény szorzatát).



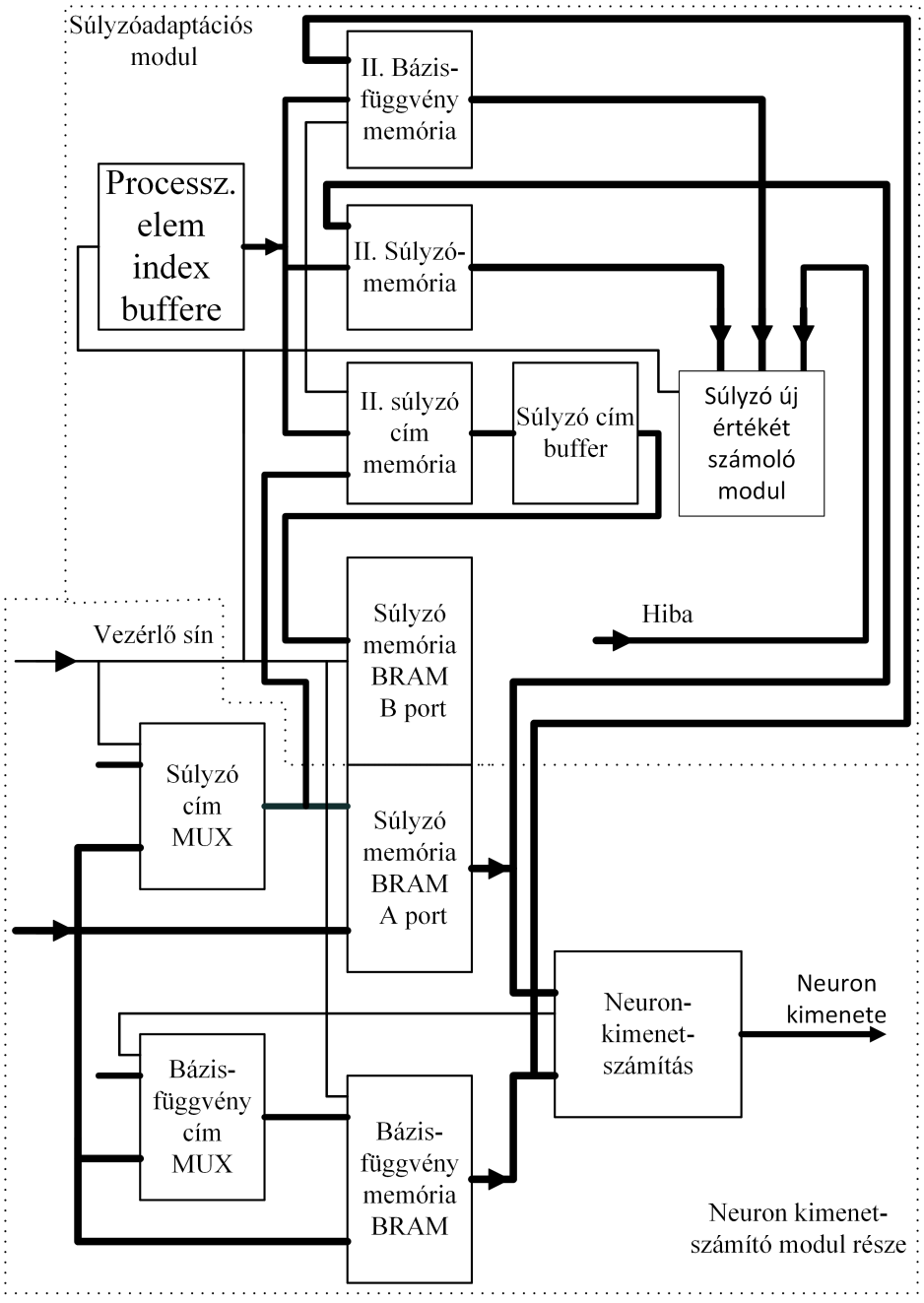
13.2. ábra. RBF processzáló elem tömbvázlata

A csővezeték maximálisan ki van használva. Csak az első lépésekben, amikor a csővezeték még nincs feltöltve, és az utolsó lépésekben, amikor már kezd kiürülni, nem minden fázisban végzünk műveletet. Az utolsó lépésben (70. lépés), miután megkaptuk a háló kimenetét, ki kell számolni az előírt érték és a kimenet közötti hibát. A háló egyik fontos jellemzője, hogy párhuzamosan történik a háló kimenetének a számítása és a súlytényezők tanítása, azzal a megjegyzéssel, hogy egy cikluskésés van a kimenetszámítás és a súlymódosítás között. A cikluskésésre azért van szükség, mert a súlymódosításhoz szükség van a hibára.

A párhuzamos tanítást és kimenetszámítást a súlytényezőknek a Dualportos BlockRAM memóriákban való tárolása teszi lehetővé. Ugyanarra az órajelre az aktuális súlytényezők értékét kiolvassuk a memóriából, és az előbbi ciklusban aktív súlytényezők címére beírjuk az új értékeket. Az előhívási fázis harmadik lépésében a kiolvasott bázisfüggvény értékét, a súlytényezők értékét, és a súlytényezők címét eltároljuk egy segédmemóriában. A következő ciklusban, mikor már ismerjük a hibát, kiszámoljuk az új értékeket és visszaírjuk a súlytényezőket tároló Dualportos BlockRAM memóriába. A súlyadaptáció lépéseit a 13.3. ábrán könnyen lehet követni. A neuronháló tesztelésére egy könyvtárkészlet segítségével lehet kommunikálni. A fontosabb utasítások a 13.1. táblázatban vannak összefoglalva:

13.1. táblázat. Hardveres RBF háló vezérlése

Funkcionalitás	I. param.	II. param.	III. param.	IV. param.
Bemenet-programozás	0	X_1	X_2	X_3
Előírt érték betöltése	1	előírt érték	-	-
Súlytényezők programozása	2	súlytényező címe	súlytényező értéke	-
Bázisfüggvény programozása	3	memória cím	bázis függvény értéke	-
Szimuláció indítása	4	-	-	-
Tanítás engedélyezése, letiltása	5	0/1	-	-



13.3. ábra. Kialakított architektúra a párhuzamos kimenetszámítás és tanítás megvalósítására

Az RBF neuronháló hardveres megvalósítására több modell is elkészült, amelyek különböző alkalmazásokban voltak tesztelve. Elsősorban a modell helyességét függvények megközelítésére teszteltük. A kutatásban egy mobil robot navigációs algoritmusában került alkalmazásra a hardveresen megvalósított RBF háló. A mobil robot navigációja során a környezet egy RBF hálóval volt modellezve, majd egy konvolúciós eljárással különböző bázispontokhoz képest, amelyekben ismertük a környezetet, meghatároztuk a robot pozícióját. A hardveresen megvalósított RBF modul egy processzor sínrendszerére van csatolva. A processzoron futó programból érhető el és vezérelhető az RBF modul. Egy újabb alkalmazása az RBF háló megvalósításának a hangjelek kódolására és dekódolására kidolgozott modell, amelyet egy diplomadolgozat keretében fejlesztettünk és teszteltünk. A neuronháló megvalósítása követte a felvázolt architektúrát, viszont System Generator környezetben, egy magasabb szintű absztrakciót alkalmaztunk a megvalósítására.

14. fejezet

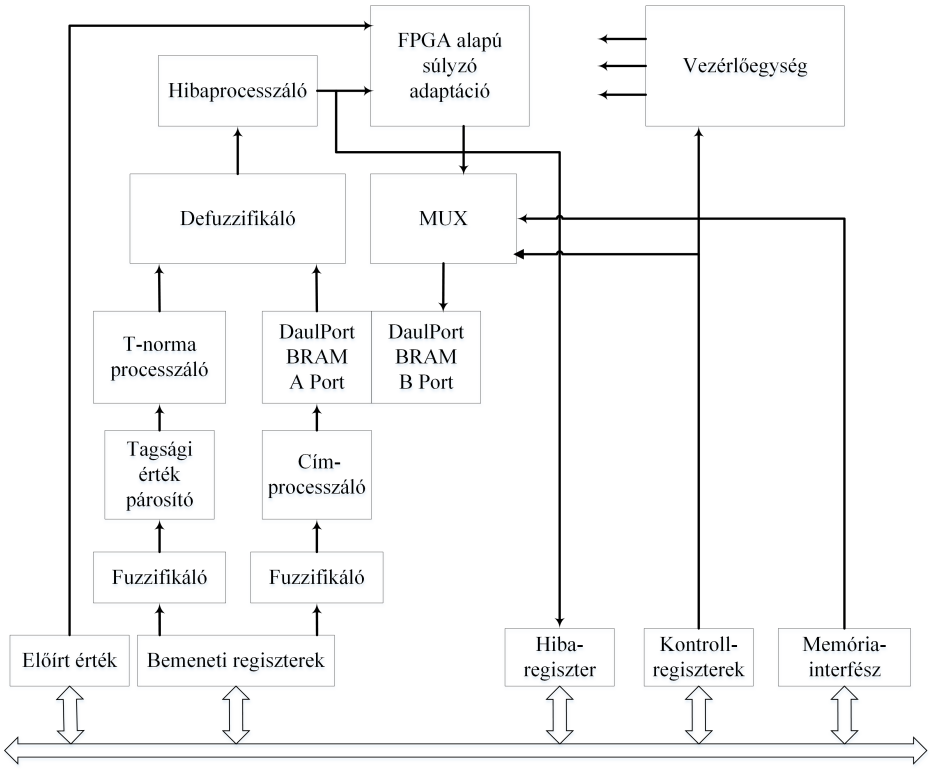
TAKAGI-SUGENO Fuzzy következtető rendszer hardvermegvalósítása

14.1. A tervezett rendszer architektúrája

14.1.1. A rendszer fontosabb alegységei

A pipeline rendszerben kialakított modellt több módszerrel is megvalósítottuk:

- VHDL hardverleíró nyelven;
- moduláris felépítést alkalmazva és a modellt System Generator-os változatban elkészítve;
- C-ben specifikálva az áramkört és a Vivado HLS 2015.2 magas szintű szintetizáló eszközt alkalmazva.



14.1. ábra. Hardvermegvalósításra javasolt rendszer egyszerűsített blokk diagramja

Mindegyik módszernél a Takagi Sugeno fuzzy rendszerből egy IP magot tervezünk, amelyet a processzor sínrendszerére csatolunk. Hardvereszközként újrakonfigurálható egységet is tartalmazó SoC típusú fejlesztőrendszeren implementáltuk. Jelen részben a System Generator változatban elkészített pipeline szerkezetű moduláris kialakítást szemléltetjük. A pipeline, hardveres kialakításra javasolt modell egyszerűsített tömbvázlata a 14.1. ábrán van szemléltetve és a következő fontosabb egységeket tartalmazza:

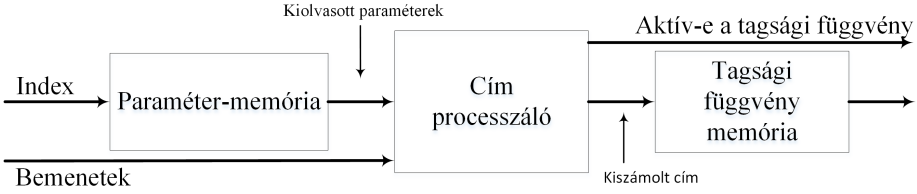
– Vezérlőegység. A vezérlőegység szinkronizálja az IP-mag alegységeit. Miután a bemenetek bekerülnek a megfelelő regiszterekbe, a kontrollregiszteren keresztül lehet parancsokat küldeni a vezérlőegységnek, valamint a státusregiszteren keresztül lekérdezni az IP-mag állapotát.

- Bemeneti kimeneti regiszterek. A bemeneti regiszterek x, y előírt érték tárolására szolgálnak; kontrollregiszter: a rendszer működésének konfigurálása, kimeneti regiszterek: hibaregiszter, a következtető rendszer kimenetét tároló regiszter, státusregiszter. Tesztelési fázisban a bemeneti regiszterek értékeit a processzorból programozzuk. Egy konkrét alkalmazás esetén a bemeneteket direkt módon a valós rendszer érzékelőiről a bemeneti regiszterbe lehet majd tárolni. Az előírt értékregiszterben a pillanatnyi bemeneteknek megfelelő elvárt kimenetet tároljuk, mivel az FPGA áramkörben megvalósított tanító algoritmusban szükség van az elvárt kimeneti értékre.
- A kontrollregiszter (regiszterek) segítségével állítjuk be az IP-mag üzemmódját (például a tanítás a processzoron történik-e vagy az áramkörben kialakított modulban). Szintén a kontrollregiszter segítségével irányítjuk a vezérlő modult. A státusregiszteren keresztül kérdezzük le a hardveres megvalósítású neuro fuzzy modul állapotát.
- Paraméter-memória: a következtetési rész paramétereinek a tárolására szolgál. A korábbi neurális hálók hardverkialakítása során elért eredmények alapján a kimenetszámítással párhuzamosan megvalósítható következtető rendszer kimenetszámítása, valamint a tanító algoritmus párhuzamos futtatása, ha a paraméterek tárolására dual portos BRAM memóriát alkalmazunk.
- Memória-interfész. A processzoron futó tanító algoritmus alkalmazásakor az újraszámolt paramétereknek ezen interfészen keresztül valósul meg a paraméter memóriába való írása.
- Fuzzyfikáló egységek – ezen modulokban vannak implementálva a tagsági függvények. Ha figyelembe vesszük, hogy csak a szomszédos tagsági függvények között van átfedés, abban az esetben mindegyik bemenetre legtöbb két tagsági függvény aktiválódik. A fuzzyfikáló eredményeként megkapjuk az aktív tagsági függvények indexeit, valamint az aktivizációs fokokat (tagsági értékeket).
- Defuzzyfikáló egység – a defuzzyfikáló egység a Takagi-Sugeno-modellben rögzített szabályok tüzelési értékei alapján, valamint a következtető rész paraméterei alapján kiszámolja a rendszer kimenetét. A hardveres megvalósítás során a O_3, O_4 és O_5 rétegek vannak implementálva. A hardveres változatban a súlytényezők normalizálása az utolsó rétegbe van eltolva. Ezzel egyszerűsítettük a pipeline csatorna kialakítását és az osztást is csak egy alkalommal kell elvégezni. Az eredeti Takagi-Sugeno-modell alapján a harmadik rétegben minden egyes súlytényező normalizálása igényel egy osztást.

14.1.2. Tagsági függvények megvalósítása

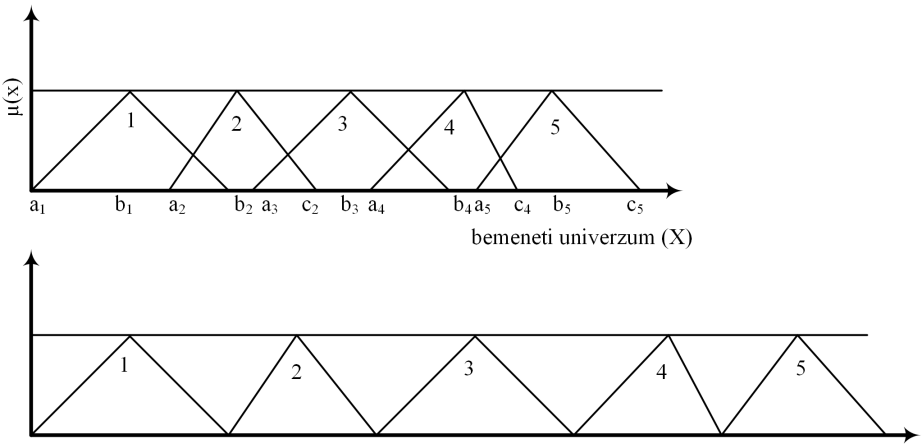
A tagsági függvények megvalósítására több lehetséges megoldás van (amint az előbbi fejezetben részleteztük). Először egyszerűbb megoldásként a tagsági függvények BRAM memóriában való tárolását alkalmazzuk. Előnye, hogy könnyen implementálható, hátránya viszont, hogy ebben az esetben a tagsági függvények paraméterei nem lesznek hangolhatók valós időben. Processzoron futó tanító algoritmust alkalmazva az új paraméterek szerint újra kell számolni a tagsági függvények értékeit és újra kell tölteni az IP mag fuzzyfikáló moduljában található paraméter memóriába. A tagsági függvények BRAM memóriában való tárolása esetében két memóriát használunk és az egyszerűsített tömbvázlat a 14.2. ábrán van szemléltetve:

- az egyik memóriában tároljuk a tagsági függvények paramétereit;
- a második memóriában pedig eltároljuk a tagsági függvények értékeit.



14.2. ábra. Tagsági függvény BRAM memóriában való megvalósítása

Háromszög típusú tagsági függvények alkalmazása, és a tagsági függvényeknek BRAM memóriában való tárolása esetén a paraméter-memória tartalma a 14.1. táblázatban van szemléltetve. A hardveres megvalósítás során több paraméter tárolható ugyanabban a BRAM vagy osztott memóriában, de minden paraméter tárolására külön memória is alkalmazható. Az univerzumoknak tagsági függvényekkel való lefedése és a tagsági függvényeknek memóriában való tárolása közötti összefüggést a 14.3. ábra szemlélteti. A bemeneti univerzum tagsági függvényekkel való lefedése során a tagsági függvények között átfedés van. A tagsági függvény memóriában a tagsági függvények egymás után vannak elhelyezve. Ha ki szeretnénk olvasni a tagsági függvény memóriából egy adott bemeneti értéknek megfelelő tagsági értéket, a tagsági érték memóriát a következő összefüggés szerint kell megcímezni:



14.3. ábra. Bemeneti univerzum lefedése tagsági függvényekkel, és tagsági függvény értékeinek a BRAM memóriában való tárolása

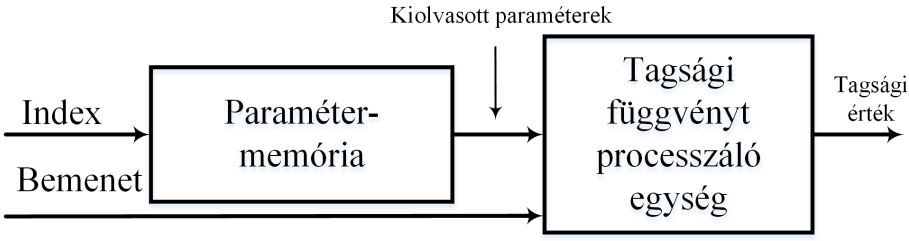
$C = (x - a_i) + kc_i$ ahol:
x – neuro fuzzy rendszer bemenete a_i – a tagsági függvény első értéke, mely különbözik nullától kc_i – az i-edik tagsági függvény kezdőcíme.

14.1. táblázat. Paraméter-memória

Index (i)	TF kezdete (a_i)	TF közepe (b_i)	TF vége (c_i)	TF kezdőcíme ($k_c i$) a tagsági memóriában
1	a1	b1	c1	kc1=0
2	a2	b2	c2	kc2= kc1+(c1-a1)
3	a3	b3	c3	kc3= kc2+(c2-a2)
4	a4	b4	c4	kc4= kc3+(c3-a3)
5	a5	b5	c5	kc5= kc4+(c4- a4)

A tagsági függvény hardveresen való megvalósításának az egyszerűsített tömbvázlata a 14.4. ábrán van szemléltetve. Az első modul, a paramétermemória tárolja a tagsági függvények paramétereit. A második modul tartalmazza a tagsági függvény hardveres megvalósítását. Haranggörbe System Generator-os megvalósítása pipeline kialakításban a 14.5. ábrán van szemléltetve (felső rész). A hatványozás hardveres implementálása körülményes feladat, a javasolt modellben a b paraméter tehát nem vehet fel bármilyen

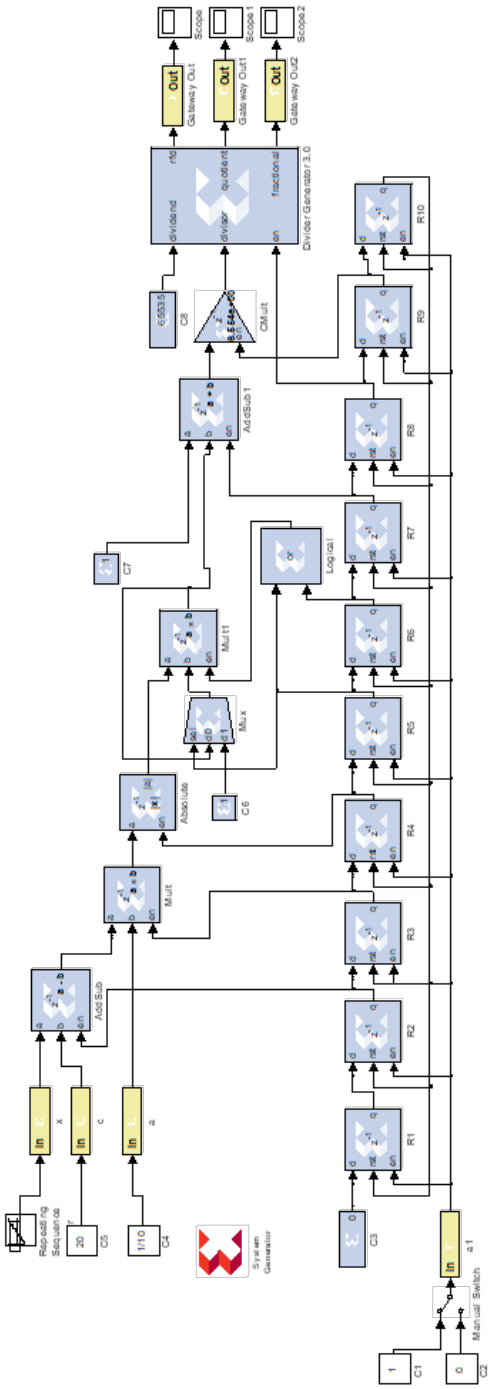
értéket. Ha a b értékei $\{0, 511, 522, 5 \ldots\}$, abban az esetben a hatványra emelés szorzással megoldható. A hardverben megvalósított haranggörbe működésének tesztelésére az egyes műveletvégző egységek szinkronizálására bistabil áramkörökből kialakított léptetőregiszter alapú vezérlő van alkalmazva (14.5. ábra alsó része). Az a_i-vel való osztást szorzással valósítjuk meg, és ebben az esetben az a_i bemeneti paramétert $1/a_i$-vel kell helyettesíteni.



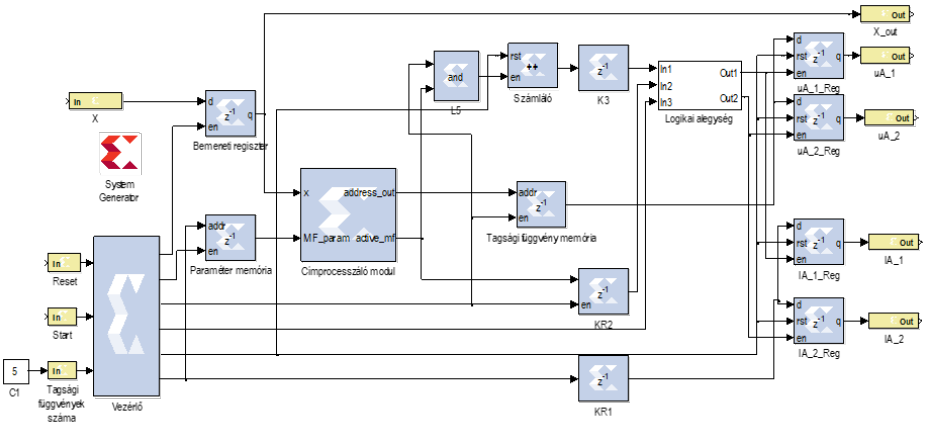
14.4. ábra. Tagsági függvény hardverarchitektúra egyszerűsített tömbvázlata

14.1.3. Fuzzyfikáló egység

A fuzzyfikáló egység System Generator-os tömbvázlata a 14.6. ábrán van szemléltetve. A pipeline struktúrában kialakított fuzzyfikálás öt fázisműveletet igényel (a fontosabb műveletek: pillanatnyi bemenet tárolása a bemeneti regiszterben, a processzálandó tagsági függvény címzése, paraméter-memória olvasása, tagsági érték memória olvasása, az aktív tagsági függvények indexének és tagsági értékének regiszterekbe való tárolása). A címprocesszáló és a vezérlő VHDL hardverleíró nyelven van implementálva.



14.5. ábra. Haranggörbe System Generator alapú megvalósítása



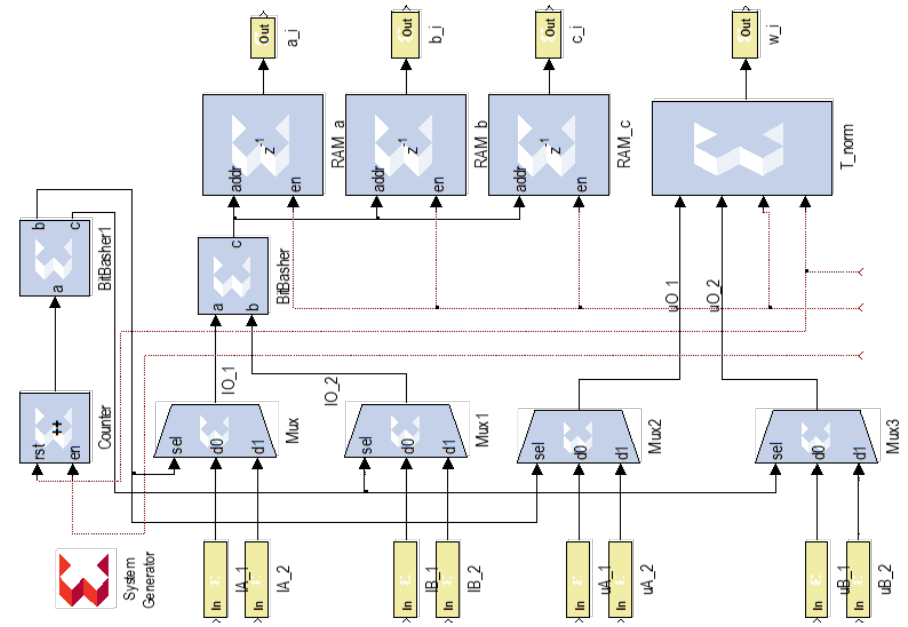
14.6. ábra. Fuzzyfikáló rész System Generator alapú tömbvázlata

14.1.4. T-norma számolása

A 14.3. táblázat szerint az indexek 1-gyel kezdődnek. A valós megvalósítás során figyelembe kell venni, hogy a tömbök címzése 0-val vagy 1-sel kezdődik. Tagsági érték párosító: egyenként kombinálja a fuzzyfikálás eredményeként kapott x bemenet tagsági értékeit az y bemenet tagsági értékeivel. A hardveres kialakítás egy kétbites számlálóval és két multiplexer áramkörrel valósítható meg. A kiválasztott aktiválási értékpárok a T-normát processzáló modul bemeneteire kerülnek. Címprocesszáló: hasonlóan a tagsági értékek párosításához, a két bemenetre aktív tagsági függvények indexét párosítja, az így kapott két index bitjeit egymás mellé helyezve egy egydimenziós tömbként (BRAM memóriaként) megvalósított paraméter memóriát címezzük meg kétdimenziós tömbként, elkerülve egy szorzó egység alkalmazását (14.7. ábra). T-norma processzáló: a pillanatnyi bemenetek alapján a T-normát számítja $O_{2,k} = w_k = \mu_{A_i}(x_1)\,\mu_{B_i}(x_2)$. A szorzat helyett több T-norma számítására megfelelő függvény is alkalmazható. Az implementáció során a T-normát több módszerrel is implementáljuk, az éppen alkalmazott változatot a konfigurációs regiszterrel választjuk ki.

14.2. táblázat. Fuzzyfikáló alegységek működése

Órajel	ϕ_1	ϕ_2	ϕ_3		ϕ_4		ϕ_5
0	bemeneti regiszter feltöltése	-	-	-	-	-	-
1	-	i= 0	-	-	-	-	-
2	-	i=1	paraméter-memória olvasás (i=0)	index-késleltető regiszter	-	-	-
3	-	i=2	paraméter-memória olvasás (i=1)	index-késleltető regiszter	Tagsági érték (i=0)	index-késleltető regiszter	-
4	-	i= 3	paraméter-memória olvasás (i=2)	index-késleltető regiszter	Tagsági érték (i=1)	index-késleltető regiszter	ha (i=0) TF aktív, a tagsági érték és index tárolása a regiszterekbe
5	-	i=4	paraméter-memória olvasás (i=3)	index-késleltető regiszter	Tagsági érték (i=2)	index-késleltető regiszter	ha (i=0) TF aktív, a tagsági érték és index tárolása a regiszterekbe
6	-	-	paraméter-memória olvasás (i=4)	index-késleltető regiszter	Tagsági érték (i=3)	index-késleltető regiszter	ha (i=0) TF aktív, a tagsági érték és index tárolása a regiszterekbe
7	-	-	-	-	Tagsági érték (i=4)	index-késleltető regiszter	ha (i=0) TF aktív, a tagsági érték és index tárolása a regiszterekbe
8	-	-	-	-	-	-	ha (i=0) TF aktív, a tagsági érték és index tárolása a regiszterekbe



14.7. ábra. Címprocesszáló, indexpárosító, T-norma és paraméter-memóriák System Generator alapú kapcsolási vázlata

14.3. táblázat. Fuzzyfikáló alegységek működése

Számláló értéke dec.	Számláló értéke bin.	b	c	IO_1	IO_2	uO_1	uO_2
0	00	0	0	IA_1	IB_1	uA_1	uB_1
1	01	0	1	IA_1	IB_2	uA_1	uB_2
2	10	1	0	IA_2	IB_1	uA_2	uB_1
3	11	1	1	IA_2	IB_2	uA_2	uB_2

14.1.5. Következtetés és defuzzyfikálás

A következtetés és defuzzyfikálás a defuzzyfikáló modulban van megvalósítva (14.8. ábra), amelyben az O_3, O_4 és O_5-ös rétegek számításait

végzi. Az O_3, O_4, O_5 rétegekre a kimenet számítását megvalósító hardvermodellt egy adatutas véges állapotú automatával szemléltetjük. Az O_5 rétegbe behelyettesítve az előbbi rétegek függvényeit, az alábbi, előbbi fejezetben bemutatott egyenleteket (14.1), (14.2) kapjuk:

$$O_{5,k} = \frac{\sum\limits_k w_k f_k}{\sum\limits_k w_k} \tag{14.1}$$

$$\begin{aligned} O_{5,k} = &\frac{w_1 \left(p_1 x + q_1 y + r_1\right) + w_2 \left(p_2 x + q_2 y + r_2\right)}{w_1 + w_2 + w_3 + w_4} + \\ &+ \frac{w_3 \left(p_3 x + q_3 y + r_3\right) + w_4 \left(p_4 x + q_4 y + r_4\right)}{w_1 + w_2 + w_3 + w_4}. \end{aligned} \tag{14.2}$$

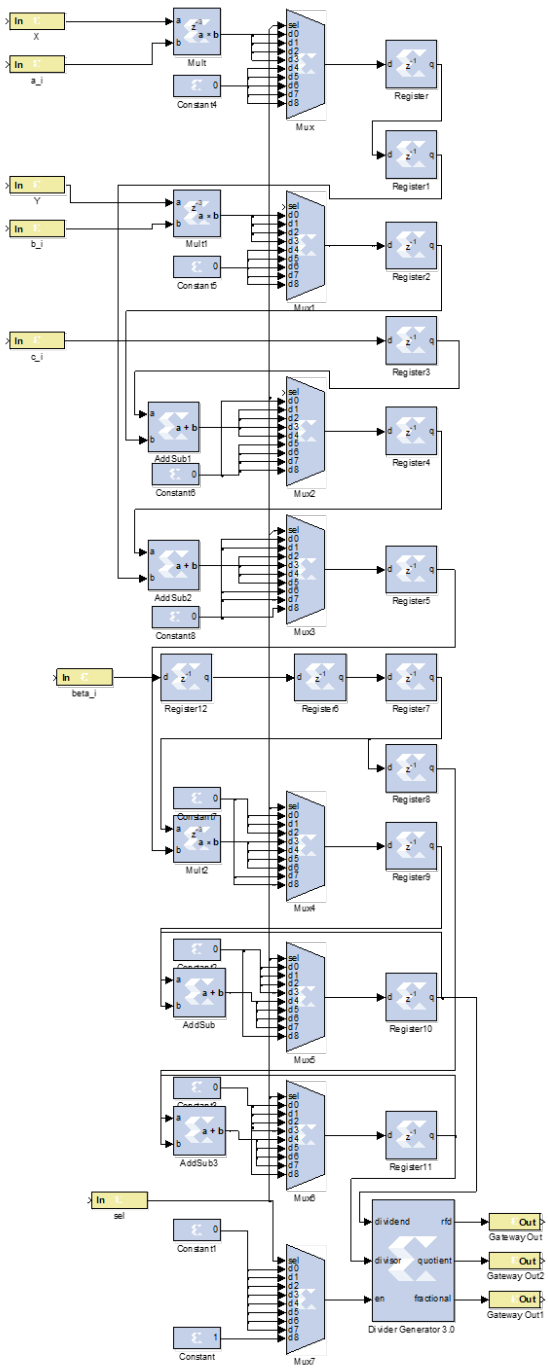
Az adatút implementálása során a következő regiszterjelöléseket alkalmaztuk: P szorzás eredményét tároló regiszter, S összegzés eredményét tároló regiszter, KR késleltető regiszter. Miután rendelkezésre állnak a paraméter-memóriából kiolvasott paraméterek és az aktivációs fokok, a következő regisztermű̈veletekkel kiszámítható az eredő kimenet.

$P_1 = p_i x$ $P_2 = q_i y$ $KR_1 = r_i$ $KR_2 = P_1$ $S_1 = P_2 + KR_1$ $S_2 = S1 + KR_2$ $KR_3 = w_i$ $KR_4 = KR_3$ $KR_5 = KR_4$ $KR_6 = KR_5$ $P_3 = KR_5 S_2$ $S_3 = S_3 + P_3$ $S_4 = S_4 + KR_6$ $Y = \frac{S3}{S4}$

14.4. táblázat. Következtető modul alegységeinek működése

-	P1	KR1	P2	KR2	S1	S2	P3	S3	KR3	KR4	KR5	KR6	S4	Y
1	x	-	x	x	-	-	-	-	x	-	-	-	-	-
2	x	x	x	x	x	-	-	-	x	x	-	-	-	-
3	x	x	x	x	x	x	-	-	x	x	x	-	-	-
4	x	x	x	x	x	x	x	-	x	x	x	x	-	-
5	-	x	x	-	x	x	x	x	-	x	x	x	x	-
6	-	-	-	-	-	x	x	x	-	-	x	x	x	-
7	-	-	-	-	-	-	x	x	-	-	-	x	x	-
8	-	-	-	-	-	-	-	x	-	-	-	-	x	-
9	-	-	-	-	-	-	-	-	-	-	-	-	-	x

A regisztermű̈veletek pipeline rendszerben a 14.4. táblázat szerint hajtódnak végre. Egyes mű̈veletek elvégzéséhez szükséges adatok nem a megfelelő időpillanatban állnak a rendelkezésre, és emiatt késleltető regiszterek beiktatására volt szükség. A System Generatorban kialakított pipeline



14.8. ábra. A következtető egységet megvalósító komplex állapotgép System Generator alapú blokk diagramja

struktúra a 14.8. ábrán van szemléltetve, és egy kilenc órajel alatt lefutó adatutas véges állapotú automata szerint volt tervezve. Hibaprocesszáló: az eredő kimenet kiszámítása után ez az egység kiszámolja az előírt érték, valamint a kiszámolt kimenet közötti különbséget. A hiba alapján történik a paraméterek hangolása. A tanítást két változatban szeretnénk megvalósítani:
1. A tanító algoritmus komplexitása miatt első fázisban a SoC rendszeren található processzoron szeretnénk futtatni a tanító algoritmust, az új paramétereket pedig felprogramozni a fuzzyfikáló, illetve következtető egységbe.
2. Az első változat alapján elért eredményeket felhasználva tervezzük meg a hardveresen implementálható, szintén pipeline struktúrában kialakított tanító algoritmust. A kutatás ezen részében nem foglalkoztunk a tanító algoritmusok részletes kidolgozásával, a kutatás következő fázisai a kimenetszámítás, illetve tanítási algoritmusok implementálását részletezik.

14.1.6. Következtetések

A neurális hálók hardvermegvalósítása szempontjából különböző típusú hardvereszközök állnak rendelkezésre. Az újrakonfigurálható rendszerek elterjedésével a kutatók elkezdték alkalmazni az FPGA áramköröket is neurális hálók hardveres megvalósítására, kezdetben kisebb-nagyobb sikerrel. A kis kapacitású FPGA áramkörök csak egyszerű neurális hálók megvalósítását tették lehetővé. A VLSI technológiák folyamatos fejlődése, mind kapacitás, mind szerkezeti felépítés és működési sebesség tekintetében, egyre komplexebb feladatok megvalósítását teszi lehetővé.

A neurális hálók hardveres megvalósítása terén is az FPGA áramkör az egyik legfontosabb hardvereszköznek számít és egyre több kutató alkalmazza. Nagyon sok neurális háló, tanító algoritmus, fuzzy, valamint neuro fuzzy rendszert modelleztek és implementáltak újrakonfigurálható digitális rendszeren: többrétegű perceptron, Kohonen, RBF, CMAC, LVQ algoritmusok, ANFIS.

A modern magas szintű szintetizáló eszközök már lehetővé teszik komplex rendszerek magas szintű programozási nyelven való modellezését és hardveres megvalósítását, lényegesen lecsökkentve egy új prototípus tervezési idejét.

A neurális hálók hardveres megvalósítása szempontjából tanulmányoztuk a fontosabb elemeket: alkalmazandó aritmetika, adatok ábrázolására igényelt felbontás, nemlineáris aktivációs függvények hardveres modellezése, súlytényezők tárolása, a tervezett rendszer hatékony tesztelése.

Az alkalmazott aritmetika szempontjából a legtöbb kutató a fixpontos aritmetika alkalmazását tartja legelőnyösebbnek, összehasonlítva a lebegőpontos ábrázolással. A szükséges hardvererőforrás lényegesen kisebb, és közel ugyanaz a pontosság érhető el, mint a lebegőpontos ábrázolás esetében. A mai modern FPGA és SoC rendszerek, melyek újrakonfigurálható részt is tartalmaznak, több száz, akár ezer szorzó vagy DSP-egységet tartalmaznak, és kapacitásukat is figyelembe véve a lebegőpontos aritmetika alkalmazása sem kizárt, igaz, jóval kisebb méretű rendszer megvalósítását teszi lehetővé.

Az utolsó részben Sugeno alapú ANFIS rendszer pipeline rendszerben kialakított modelljét tanulmányoztuk. Kidolgoztunk egy nagy működési sebességet biztosító pipeline struktúrát, mely a bemeneten 5 tagsági függvényt alkalmazva, megközelítőleg 16 órajel alatt biztosítja a következtető rendszer kimenetének a kiszámítását (az implementáció során az alegységek szinkronizálása miatt még egy pár órajel közbeiktatásával kell számolni).

IRODALOMJEGYZÉK

[1] H. H. Aghdam and E. J. Heravi, *Guide to Convolutional Neural Networks: A Practical Application to Traffic-Sign Detection and Classification*. Springer Publishing Company, Incorporated, 1st ed., 2017.

[2] S. S. Haykin, *Neural networks and learning machines*. Upper Saddle River, NJ: Pearson Education, third ed., 2009.

[3] M. Negnevitsky, *Artificial intelligence: a guide to intelligent systems*. Pearson Education, 2005.

[4] A. Márta and H. Gábor, *Neurális hálózatok*. Panem, 2006.

[5] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, Fast and accurate deep network learning by exponential linear units (elus), *arXiv preprint arXiv:1511.07289*, 2015.

[6] D. Kriesel, *A Brief Introduction to Neural Networks*. www.dkriesel.com, 2007.

[7] R. S. Sutton and A. G. Barto, *Reinforcement learning - an introduction*. Adaptive computation and machine learning, MIT Press, 1998.

[8] F. Rosenblatt, The perceptron: a probabilistic model for information storage and organization in the brain, *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[9] M. Minsky and S. A. Papert, *Perceptrons: An introduction to computational geometry*. MIT press, 2017.

[10] K. Ben and P. Van der Smagt, An introduction to neural networks, *University of Amsterdam*, 1996.

[11] E. Fiesler and R. Beale, *Handbook of neural computation*. CRC Press, 1996.

[12] G. Huba, Antal; Lipovszki, *Méréselmélet*. BME MOGI, 2014.

[13] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, Gradient-based learning applied to document recognition, *Proceedings of the IEEE*, vol. 86, no. 11, 2278–2324, 1998.

[14] M. Mitchell, *An introduction to genetic algorithms*. MIT press, 1998.

[15] M. Dorigo, M. Birattari, C. Blum, M. Clerc, T. Stützle, and A. Winfield, Ant Colony Optimization and Swarm Intelligence: 6th International Conference, ANTS 2008, Brussels, Belgium, September 22-24, 2008, *Proceedings*, vol. 5217. Springer, 2008.

[16] J. Kennedy, Particle swarm optimization, in *Encyclopedia of machine learning*, 760–766, Springer, 2011.

[17] M. Clerc, *Particle swarm optimization*, vol. 93. John Wiley & Sons, 2010.

[18] M. A. Nielsen, Neural networks and deep learning, 2018.

[19] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[20] R. E. Schapire and Y. Freund, *Boosting: Foundations and algorithms*. MIT press, 2012.

[21] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[22] S. Ioffe and C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, *arXiv preprint arXiv:1502.03167*, 2015.

[23] F.-F. Li and A. Karpathy, CS231n- convolutional neural networks for visual recognition, 2017.

[24] R. Rojas, *Neural networks: a systematic introduction*. Springer Science & Business Media, 2013.

[25] J. C. Principe, N. R. Euliano, and W. C. Lefebvre, *Neural and adaptive systems: fundamentals through simulations*, vol. 672. Wiley New York, 2000.

[26] O. De Jesus and M. T. Hagan, Backpropagation algorithms for a broad class of dynamic networks, *IEEE Transactions on Neural Networks*, vol. 18, no. 1, 14–27, 2007.

[27] MATLAB, Matlab neural network toolbox, version 8.4, (r2015b), 2015.

[28] S. Haykin, *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.

[29] J. Moody and C. J. Darken, Fast learning in networks of locally-tuned processing units, *Neural computation*, vol. 1, no. 2, 281–294, 1989.

[30] M. J. Orr *et al.*, Introduction to radial basis function networks, 1996.

[31] S. T. Brassai, L. Bako, G. Pana, and S. Dan, Neural control based on RBF network implemented on FPGA, in *11th International Conference on Optimization of Electrical and Electronic Equipment (OPTIM)*, 41–46, IEEE, 2008.

[32] S. T. Brassai, C. Enăchescu, and L. Losonczi, RBF network for mobile robot sonar based localization and environment modeling, in *13th International Conference on Optimization of Electrical and Electronic Equipment (OPTIM)*, 1499–1504, IEEE, 2012.

[33] S. T. Brassai, C. Enăchescu, L. Losonczi, and L. Márton, FPGA based embedded support for mobile robot sonar based navigation, in *16th International Conference on System Theory, Control and Computing (ICSTCC)*, 1–6, 2012.

[34] T. Kohonen and T. Honkela, Kohonen network, *Scholarpedia*, vol. 2, no. 1, p. 1568, 2007.

[35] T. Kohonen, The self-organizing map, *Proceedings of the IEEE*, vol. 78, no. 9, 1464–1480, 1990.

[36] E. Kuriscak, P. Marsalek, J. Stroffek, and P. G. Toth, Biological context of hebb learning in artificial neural networks, a review, *Neurocomputing*, vol. 152, 27–35, 2015.

[37] S. T. Brassai, FPGA based hardware implementation of a self-organizing map, in *18th International Conference on Intelligent Engineering Systems (INES)*, 101–104, IEEE, 2014.

[38] H. Card, S. Kamarsu, and D. McNeill, Competitive learning and vector quantization in digital vlsi systems, *Neurocomputing*, vol. 18, no. 1, 195–227, 1998.

[39] B. Avşar and D. E. Aliabadi, Parallelized neural network system for solving euclidean traveling salesman problem, *Applied Soft Computing*, vol. 34, 862–873, 2015.

[40] S. T. Brassai, B. Iantovics, and C. Enăchescu, Artificial intelligence in the path planning optimization of mobile agent navigation, *Procedia Economics and Finance*, vol. 3, 243–250, 2012.

[41] M. Kolasa, R. Długosz, and J. Pauk, A comparative study of different neighborhood topologies in WTM kohonen self-organizing maps, in *Solid State Phenomena*, vol. 147, 564–569, Trans Tech Publ, 2009.

[42] L. Dávid and L. Márton, *Rețele neuronale și logica fuzzy în automatizări.* Editura Universității Petru Maior, 2000.

[43] X. Guo, F. Merrikh-Bayat, L. Gao, B. D. Hoskins, F. Alibart, B. Linares-Barranco, L. Theogarajan, C. Teuscher, and D. B. Strukov, Modeling and experimental demonstration of a Hopfield network analog-to-digital converter with hybrid CMOS/memristor circuits, *Frontiers in neuroscience*, vol. 9, p. 488, 2015.

[44] B. W. Lee and B. J. Sheu, Design of a neural-based A/D converter using modified Hopfield network, *IEEE Journal of Solid-State Circuits*, vol. 24, no. 4, 1129–1135, 1989.

[45] S. T. Brassai and L. Bakó, Hardware implementation of CMAC type neural network on FPGA for command surface approximation, *Acta Polytechnica Hungarica*, vol. 4, no. 3, 5–16, 2007.

[46] S. T. Brassai, L. Bakó, and S. Dan, FPGA parallel implementation of CMAC type neural network with on chip learning, in *4th International Symposium on Applied Computational Intelligence and Informatics*, 111–115, IEEE, 2007.

[47] J. S. Albus, A new approach to manipulator control: The cerebellar model articulation controller (CMAC), *Journal of Dynamic Systems, Measurement, and Control*, vol. 97, no. 3, 220–227, 1975.

[48] H. Gábor, *Neurális hálózatok és műszaki alkalmazásaik.* BME, 1995.

[49] T. J. Ross, *Fuzzy logic with engineering applications.* John Wiley & Sons, 2009.

[50] R. T. Yeh and S. Bang, Fuzzy relations, fuzzy graphs, and their applications to clustering analysis, in *Fuzzy Sets and their Applications to Cognitive and Decision Processes* (L. A. Zadeh, K.-S. Fu, K. Tanaka, and M. Shimura, eds.), 125–149, Academic Press, 1975.

[51] L. Zadeh, Fuzzy sets, *Information and Control*, vol. 8, no. 3, 338–353, 1965.

[52] H.-J. Zimmermann, Fuzzy set theory, *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 2, no. 3, 317–332, 2010.

[53] A. Chandramohan and M. Rao, Novel, useful, and effective definitions for fuzzy linguistic hedges, *Discrete dynamics in nature and society*, vol. 2006, 2006.

[54] V.-N. Huynh, T. B. Ho, and Y. Nakamori, A parametric representation of linguistic hedges in Zadeh's fuzzy logic, *International journal of approximate reasoning*, 2002.

[55] L. A. Zadeh, A fuzzy-set-theoretic interpretation of linguistic hedges, *Journal of Cybernetics*, 1972.

[56] M. G. Bronstein I. N., Szemengyajev K. A. and M. H., *Matematikai kézikönyv*. TypoTeX Kiadó, Budapest, 2004.

[57] M. Hussain, Fuzzy relations, 2010.

[58] M. G. Simoes, Introduction to fuzzy control, *Colorado School of Mines, Engineering Division, Golden, Colorado*, vol. 8, 18–22, 2010.

[59] N. Sabri, S. Aljunid, M. Salim, R. Badlishah, R. Kamaruddin, and M. A. Malek, Fuzzy inference system: Short review and design, *International Review of Automatic Control*, vol. 6, no. 4, 2013.

[60] E. H. Mamdani and S. Assilian, An experiment in linguistic synthesis with a fuzzy logic controller, *International journal of man-machine studies*, vol. 7, no. 1, 1–13, 1975.

[61] T. Takagi and M. Sugeno, Fuzzy identification of systems and its applications to modeling and control, *IEEE transactions on systems, man, and cybernetics*, no. 1, 116–132, 1985.

[62] C.-C. Lee, Fuzzy logic in control systems: fuzzy logic controller. i, *IEEE Transactions on systems, man, and cybernetics*, vol. 20, no. 2, 404–418, 1990.

[63] L. T. Kóczy and D. Tikk, Fuzzy rendszerek, *TypoTEX, Budapest*, 2000.

[64] K. M. Passino, S. Yurkovich, and M. Reinfrank, *Fuzzy control*, vol. 20. Citeseer, 1998.

[65] T. J. Ross, *Fuzzy logic with engineering applications*. John Wiley & Sons, 2009.

[66] MATLAB fuzzy logic toolbox, 2016. The MathWorks, Natick, MA, USA.

[67] H. Godfrey, *Fuzzy Logic with MATLAB*. USA: CreateSpace Independent Publishing Platform, 2016.

[68] T. Tămas, S. Hajdu, and S. T. Brassai, Adaptive neuro-fuzzy structure based control architecture, *Procedia Technology*, vol. 22, 600–605, 2016.

[69] T. Tămas and S. T. Brassai, Hardware implementation of a neuro-fuzzy controller using high level synthesis tool, *MACRo 2015*, vol. 1, no. 1, 183–191, 2015.

[70] S. T. Brassai and T. Tămas, *Project: Future internet technologies, sub project: Development and application of a design method for pipeline multiprocessor systems with a task-dependant architecture in embedded high-speed target systems*, National Coordinating Center for Infocommunication, University of Pannonia, Veszprém, Hungary, Reasearch report paper, 2015.

[71] C. Maxfield, *The design warrior's guide to FPGAs: devices, tools and flows*. Elsevier, 2004.

[72] R. Sass and A. G. Schmidt, *Embedded systems design with platform FPGAs: principles and practices*. Morgan Kaufmann, 2010.

[73] F. Attila and V. Zsolt, Beágyazott rendszerek és programozható logikai eszközök: egyetemi tananyag, 2011. Bibliogr.: 250–251.

[74] A. R. Omondi and J. C. Rajapakse, *FPGA Implementations of Neural Networks*. Springer Publishing Company, Incorporated, 1st ed., 2010.

[75] J. G. Eldredge and B. L. Hutchings, RRANN: The run-time reconfiguration artificial neural network, in *Custom Integrated Circuits Conference, 1994, Proceedings of the IEEE 1994*, 77–80, IEEE, 1994.

[76] A. T. Ferrucci, M. Martin, T. Geocaris, M. Schlag, and P. Chan, *Acme: A field-programmable gate array implementation of a self-adapting and scalable connectionist network*, Master's thesis, University of California, Santa Cruz, 1994.

[77] P. K. Chan, *A reconfigurable hardware accelerator for back-propagation connectionist classifiers*. PhD thesis, University of California Santa Cruz, 1994.

[78] M. Skrbek, Fast neural network implementation, *Neural Network World*, vol. 9, no. 5, 375–391, 1999.

[79] C. E. Cox and W. E. Blanz, Ganglion-a fast field-programmable gate array implementation of a connectionist classifier, *IEEE Journal of Solid-State Circuits*, vol. 27, no. 3, 288–299, 1992.

[80] S. B. Reddy, *FPGA Implementation of Back-Propagation Algorithm of Artificial Neural Networks*. PhD thesis, San Diego State University, 2017.

[81] A. Pérez-Uribe and E. Sanchez, FPGA implementation of an adaptable-size neural network, in *Artificial Neural Networks — ICANN 96* (C. von der Malsburg, W. von Seelen, J. C. Vorbrüggen, and B. Sendhoff, eds.), (Berlin, Heidelberg), 383–388, Springer Berlin Heidelberg, 1996.

[82] A. Pérez Uribe, Structure-adaptable digital neural networks, tech. rep., Epfl, 1999.

[83] P. Y. Simard and H. P. Graf, Backpropagation without multiplication, in *Advances in Neural Information Processing Systems 6* (J. D. Cowan, G. Tesauro, and J. Alspector, eds.), 232–239, Morgan-Kaufmann, 1994.

[84] F. Gers, H. de Garis, and M. Korkin, A simplified cellular automata based NEU4 ron model, *Evolution Artificiele 97, Nimes, France*, p. 2114229, 1997.

[85] H. De Garis and M. Korkin, The cam-brain machine (CBM): an FPGA-based hardware tool that evolves a 1000 neuron-net circuit module in seconds and updates a 75 million neuron artificial brain for real-time robot control, *Neurocomputing*, vol. 42, no. 1–4, 35–68, 2002.

[86] T. Nordström, *Highly parallel computers for artificial neural networks.* PhD thesis, Luleå tekniska universitet, 1995.

[87] V. Pandya, S. Areibi, and M. Moussa, A handel-c implementation of the back-propagation algorithm on field programmable gate arrays, in *2005 International Conference on Reconfigurable Computing and FPGAs (ReConFig'05)*, IEEE, 2005.

[88] K. R. Nichols, *A reconfigurable computing architecture for implementing artificial neural networks on FPGA*. University of Guelph, 2003.

[89] K. Anton, L. Thomas, and K. Marco, FPGA implementation of a neural network for a real-time hand tracking system, *Electronic Design, Test and Applications, IEEE International Workshop on*, vol. 00, p. 313, 2002.

[90] F. Yang and M. Paindavoine, Implementation of an RBF neural network on embedded systems: real-time face tracking and identity verification, *IEEE Trans. Neural Networks*, vol. 14, no. 5, 1162–1175, 2003.

[91] R. Gadea, J. Cerdá, F. Ballester, and A. Mocholí, Artificial neural network implementation on a single FPGA of a pipelined on-line backpropagation, in *Proceedings of the 13th International Symposium on System Synthesis*, ISSS '00, (Washington, DC, USA), 225–230, IEEE Computer Society, 2000.

[92] B. Girau and C. Torres-Huitzil, Massively distributed digital implementation of an integrate-and-fire legion network for visual scene segmentation, *Neurocomput.*, vol. 70, 1186–1197, Mar. 2007.

[93] K. L. Rice, T. M. Taha, and C. N. Vutsinas, Scaling analysis of a neocortex inspired cognitive model on the Cray XD1, *The Journal of Supercomputing*, vol. 47, no. 1, 21–43, 2009.

[94] D. George and J. Hawkins, A hierarchical Bayesian model of invariant pattern recognition in the visual cortex, in *Conference on IEEE International Joint Neural Networks, IJCNN'05. Proceedings*, vol. 3, 1812–1817, IEEE, 2005.

[95] H. Li, D. Zhang, and S. Y. Foo, A stochastic digital implementation of a neural network controller for small wind turbine systems, *IEEE transactions on power electronics*, vol. 21, no. 5, 1502–1507, 2006.

[96] D. Kim, H. Kim, H. Kim, G. Han, and D. Chung, A simd neural network processor for image processing, in *International Symposium on Neural Networks*, 665–672, Springer, 2005.

[97] M. Van Daalen, P. Jeavons, and J. Shawe-Taylor, A stochastic neural architecture that exploits dynamically reconfigurable FPGAs, in *FPGAs for Custom Computing Machines, 1993. Proceedings. IEEE Workshop on*, 202–211, IEEE, 1993.

[98] S. L. Bade and B. L. Hutchings, FPGA-based stochastic neural networks-implementation, in *FPGAs for Custom Computing Machines, Proceedings. IEEE Workshop on*, 189–198, IEEE, 1994.

[99] S. McBader, P. Lee, and A. Sartori, The impact of modern FPGA architectures on neural hardware: A case study of the totem neural processor, in *IEEE International Joint Conference on Neural Networks Proceedings*, vol. 4, 3149–3154, IEEE, 2004.

[100] S. Himavathi, D. Anitha, and A. Muthuramalingam, Feedforward neural network implementation in FPGA using layer multiplexing for effective resource utilization, *IEEE Transactions on Neural Networks*, vol. 18, no. 3, 880–888, 2007.

[101] Ö. Polat and T. Yıldırım, FPGA implementation of a general regression neural network: An embedded pattern classification system, *Digital Signal Processing*, vol. 20, no. 3, 881–886, 2010.

[102] C.-J. Lin and H.-M. Tsai, FPGA implementation of a wavelet neural network with particle swarm optimization learning, *Mathematical and Computer Modelling*, vol. 47, no. 9–10, 982–996, 2008.

[103] L. Bako, E. Gyorgy-Mozes, S.-T. Brassai, L. Losonczi, and L. F. Marton, Neural network parallelization on FPGA platform for eeg signal classification, in *The International Conference Interdisciplinarity in Engineering INTER-ENG*, p. 370, Editura Universitatii Petru Maior din Tirgu Mures, 2012.

[104] M. Papadonikolakis and C.-S. Bouganis, Novel cascade FPGA accelerator for support vector machines classification, *IEEE transactions on neural networks and learning systems*, vol. 23, no. 7, 1040–1052, 2012.

[105] L. Bako and S.-T. Brassai, Spiking neural networks built in FPGAs: fully parallel implementations., *WSEAS Transactions on Circuits and Systems*, vol. 5, no. 3, 346–353, 2006.

[106] L. Bakó and S.-T. Brassai, Hardware spiking neural networks: parallel implementations using FPGAs, in *Proceedings of the 8th WSEAS international conference on Automatic control, modeling & simulation*, 261–266, World Scientific and Engineering Academy and Society (WSEAS), 2006.

[107] L. Bako, S. Brassai, I. Székely, and M. Baczo, Hardware implementation of delay-coded spiking-rbf neural network for unsupervised clustering, in *11th International Conference on Optimization of Electrical and Electronic Equipment*, 51–56, IEEE, 2008.

[108] G. Horvath and Z. Csipak, Fpga implementation of the kernel cmac, in *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on*, vol. 4, 3143–3148, IEEE, 2004.

[109] S. T. Brassai, FPGA implementation of fuzzy controllers and simulation based on a fuzzy controlled mobile robot, *Acta Universitatis Sapientiae, Electrical and Mechanical Engineering*, vol. 1, 93–104, 2009.

[110] S. Brassai, L. Dávid, and L. Bakó, Hardware implementation of CMAC based artificial network with process control application, *Timișoara, Transaction on Electronics and communication, Scientific buletin of the" Politehnica" University of Timisoara*, 209–213, 2004.

[111] S. Brassai and L. Bakó, Visual trajectory control of a mobile robot using FPGA implemented neural network, *Pollack Periodica*, vol. 4, no. 3, 129–142, 2009.

[112] S. T. Brassai, L. F. Márton, L. Dávid, and L. Bakó, Hardware implemented neural network based mobile robot control, in *14th International Symposium for Design and Technology of Electronics Packages (SIITME)*, 124–138, Sept 2008.

[113] Xilinx, *Vivado Design Suite User Guide: Model-Based DSP Design Using System Generator*. Xilinx.

[114] S. T. Brassai, *Sisteme Neuroadaptive Realizate cu Circuite FPGA cu Aplicații în Sistemele de Control Automat*. PhD thesis, Iniversitatea Transilvania din Brasov, 2008.

[115] Xilinx, *MicroBlaze Processor Reference Guide*. Xilinx.

[116] B. S. Tihamér, Parallel pipeline solution for hardware implementation of artificial neural networks within circuit real time weight update, in *8th International Conference on High-Performance and Embedded Architectures and Compilers (HiPEAC), 2nd Workshop on Design Tools and Architectures for Multi-Core Embedded Computing Platforms (DITAM'13)*, 2013.

[117] S. Sahin, Y. Becerikli, and S. Yazici, Neural network implementation in hardware using FPGAs, in *International Conference on Neural Information Processing*, 1105–1112, Springer, 2006.

[118] S. Hashemi, N. Anthony, H. Tann, R. I. Bahar, and S. Reda, Understanding the impact of precision quantization on the accuracy and energy of neural networks, *CoRR*, vol. abs/1612.03940, 2016.

[119] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, Quantized neural networks: Training neural networks with low precision weights and activations, *CoRR*, vol. abs/1609.07061, 2016.

[120] M. Moussa, S. Areibi, and K. Nichols, On the arithmetic precision for implementing back-propagation networks on FPGA: a case study, in *FPGA Implementations of Neural Networks*, 37–61, Springer, 2006.

[121] P. Y. Simard and H. P. Graf, Backpropagation without multiplication, in *Advances in Neural Information Processing Systems*, 232–239, 1994.

[122] J. G. Eldredge and B. L. Hutchings, RRANN: a hardware implementation of the backpropagation algorithm using reconfigurable FPGAs,

in *IEEE International Conference on Neural Networks, IEEE World Congress on Computational Intelligence*, vol. 4, 2097–2102, IEEE, 1994.

[123] P. Lysaght, J. Stockwood, J. Law, and D. Girma, Artificial neural network implementation on a fine-grained FPGA, in *International Workshop on Field Programmable Logic and Applications*, 421–431, Springer, 1994.

[124] R. F. Molz, P. M. Engel, F. G. Moraes, L. Torres, and M. Robert, Codesign of fully parallel neural network for a classification problem, in *International Conference on Information Systems, Analysis and Synthesis, Orlando, USA*, 2000.

[125] M. Boubaker, K. B. Khalifa, B. Girau, M. Dogui, and M. H. Bedoui, On-line arithmetic based reprogrammable hardware implementation of LVQ neural network for alertness classification, *IJCSNS International Journal of Computer Science and Network Security*, vol. 8, no. 3, 260–266, 2008.

[126] B. Girau and A. Tisserand, On-line arithmetic-based reprogrammable hardware implementation of multilayer perceptron back-propagation, in *Microelectronics for Neural Networks, Proceedings of Fifth International Conference on*, 168–175, IEEE, 1996.

[127] J. L. Holt and T. E. Baker, Back propagation simulations using limited precision calculations, in *Neural Networks, 1991., IJCNN-91-Seattle International Joint Conference on*, vol. 2, 121–126, IEEE, 1991.

[128] M. Moussa, S. Areibi, and K. Nichols, On the arithmetic precision for implementing back-propagation networks on FPGA: a case study, in *FPGA Implementations of Neural Networks*, 37–61, Springer, 2006.

[129] A. W. Savich, M. Moussa, and S. Areibi, The impact of arithmetic representation on implementing MLP-BP on FPGAs: A study, *IEEE transactions on neural networks*, vol. 18, no. 1, 240–252, 2007.

[130] A. R. Omondi, *Computer Arithmetic Systems: Algorithms, Architecture and Implementation*. Hertfordshire,: Prentice Hall International (UK) Ltd., 1994.

[131] D. Henderson, Elementary functions: Algorithms and implementation, *Mathematics and Computer Education*, vol. 34, no. 1, p. 94, 2000.

[132] P. Murtagh and A. Tsoi, Implementation issues of sigmoid function and its derivative for VLSI digital neural networks, *IEE Proceedings E-Computers and Digital Techniques*, vol. 139, no. 3, 207–214, 1992.

[133] Xilinx, *CORDIC v6.0 LogiCORE IP Product Guide*. Xilinx.

[134] I. Sahin and I. Koyuncu, Design and implementation of neural networks neurons with radbas, logsig, and tansig activation functions on FPGA, *Elektronika ir elektrotechnika*, vol. 120, no. 4, 51–54, 2012.

[135] E. Jamro, K. Wiatr, and M. Wielgosz, FPGA implementation of 64-bit exponential function for HPC, in *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on*, 718–721, IEEE, 2007.

[136] M. Tommiska, Efficient digital implementation of the sigmoid function for reprogrammable logic, *IEE Proceedings-Computers and Digital Techniques*, vol. 150, no. 6, 403–411, 2003.

[137] C. Alippi and G. Storti-Gajani, Simple approximation of sigmoidal functions: realistic design of digital neural networks capable of learning, in *IEEE International Sympoisum on Circuits and Systems*, 1505–1508, 1991.

[138] M. Zhang, S. Vassiliadis, and J. G. Delgado-Frias, Sigmoid generators for neural computing using piecewise approximations, *IEEE transactions on Computers*, vol. 45, no. 9, 1045–1049, 1996.

[139] K. Basterretxea, J. Tarela, and I. Del Campo, Approximation of sigmoid function and the derivative for hardware implementation of artificial neurons, *IEE Proceedings-Circuits, Devices and Systems*, vol. 151, no. 1, 18–24, 2004.

[140] V. Beiu, J. A. Peperstraete, J. Vandewalle, and R. Lauwereins, Closse approximations of sigmoid functions by sum of step for VLSI implementation of neural networks, *Sci. Ann. Cuza Univ.*, vol. 3, 5–34, 1994.

[141] J. Alspector, B. Gupta, and R. B. Allen, Performance of a stochastic learning microchip, in *Advances in neural information processing systems*, 748–760, 1989.

[142] M. Verleysen, P. Thissen, J.-L. Voz, and J. Madrenas, An analog processor architecture for a neural network classifier, *IEEE Micro*, vol. 14, no. 3, 16–28, 1994.

[143] Nestor, Inc, *NESTOR Ni1000 Recognition Accelerator*, 1996.

[144] U. Ramacher, W. Raab, J. A. U. Hachmann, J. Beichter, N. Bruls, M. Wesseling, E. Sicheneder, J. Glass, A. Wurz, and R. Manner, Synapse-1: a high-speed general purpose parallel neurocomputer system, in *Proceedings of 9th International Parallel Processing Symposium*, 774–781, April 1995.

[145] E. Sackinger, B. E. Boser, J. Bromley, Y. LeCun, and L. D. Jackel, Application of the anna neural network chip to high-speed character recognition, *IEEE Transactions on Neural Networks*, vol. 3, 498–505, May 1992.

[146] D. Hammerstrom, T. Leen, and E. Means, Dynamics and VLSI implementation of self–organizing networks, in *Advanced Neural Computers*, 185–192, Amsterdam: North-Holland, 1990.

[147] D. C. J. Naylor, S. Jones, D. Myers, and J. Vincent, Neural network feature detector for real-time video signal processing, *Int. J. Neural Syst.*, vol. 4, no. 4, 337–349, 1993.

[148] D. Hammerstrom and N. Nguyen, An implementation of Kohonen's self-organizing map on the Adaptive Solutions neurocomputer, in *Artificial Neural Networks*, vol. I, (Amsterdam, Netherlands), 715–720, North-Holland, 1991.

[149] D. W. Hammerstrom and S. Rehfuss, Neurocomputing hardware: present and future, *Artif. Intell. Rev.*, vol. 7, no. 5, 285–300, 1993.

ABSTRACT

Neural Networks and Fuzzy Logic

The volume entitled *Neural Networks and Fuzzy Logic* is divided into three parts: neural networks, fuzzy logic, and hardware implementations.

During 9 chapters deal with the basics of neural networks, and in two chapters the fuzzy logic and fuzzy inference systems are discussed.

In the firts part of the book within the part of neural networks, the basics of neural network topologies, neural network training solution, single-layer perceptron structures, and training are presented.

In the fourth chapter, the gradient-based optimization algorithm used for neural network parameter update, the cost fucntions used for neural network training, solutions for data normalization, and regularization solutions for neural network size optimization are discussed.

Chapter five describes the structure of the multilayer perceptron and its teaching with the backpropagation algorithm.

In Chapter 6, the RBF neural network structure, teaching, and application areas are detailed.

In Chapter 7, competitive neural nets are presented. In the framework of competitive neural nets, the Kohonen network topology and training as well as the application of the Kohonen Network for the travelling salesman problem are detailed.

In the next chapter, associative and auto-associative nets are being discussed.

The structure and teaching of the CMAC-type neuronal network are explained in Chapter 9.

In the 10^{th} and 11^{th} chapters, the basic knowledge of fuzzy logic, membership functions, fuzzy set operations, fuzzy relation with fuzzy set, and Mamdani- and Takagi-Suegno-type fuzzy inference systems are reviewed.

Chapter 11 provides insights to the application of the Mamdani-type fuzzy inference system.

In the third part, we discuss the hardware-based implementation of neural networks and fuzzy inference systems.

The constraints applied to the hardware implementation of neural networks, such as applied arithmetic, precision, and possible parallelization solutions, are detailed.

In the last chapters, the hardware implementation of an RBF neuron network and a Takagi-Sugeno fuzzy inference system are presented. The results related to the hardware implementation were achieved during the stientific research I have carried out.

In the book, I also tried to apply the experience gained during the teaching in the field of artificial intelligence at Sapientia University.

REZUMAT

Reele neuronale și logica fuzzy

Volumul intitulat *Rețele neuronale și logica fuzzy* este structurat pe trei părți: rețele neuronale, logică fuzzy și implementare hardware.

Pe parcursul a nouă capitole sunt tratate elementele de bază ale rețelelor neuronale, în două capitole logica fuzzy, respectiv sisteme de inferență fuzzy, iar în ultimele trei capitole implementarea hardware a rețelelor neuronale și sistemelor de inferență fuzzy.

În prima parte a cărții, în cadrul rețelelor neuronale sunt prezentate elementele de bază ale topologiilor rețelelor neuronale, soluții de antrenare a rețelelor neuronale, structura perceptronului cu un singur strat și antrenarea acestuia.

În capitolul al patrulea sunt discutați algoritmi de optimizare bazați pe gradient utilizat pentru actualizarea parametrilor rețelei neuronale, funcțiile de cost utilizate pentru antrenarea rețelelor neuronale, soluțiile pentru normalizarea datelor și posibilități de regularizare utilizate cu scopul de optimizare a dimensiunii rețelei neuronale. Capitolul cinci descrie structura perceptronului multistrat și antrenarea acestuia cu algoritmul backpropagation. În capitolul șase sunt detaliate structura rețelei neuronale RBF, antrenarea și domeniile de aplicare ale acesteia.

În capitolul șapte sunt prezentate rețelele neurale competitive. În cadrul rețelelor neurale competitive, sunt detaliate topologia rețelei Kohonen, antrenarea, respectiv aplicarea rețelei Kohonen pentru rezolvarea problemei agentului călător.

În capitolul următor, sunt discutate rețelele asociative și autoasociative.

Structura și antrenarea rețelei neuronale de tip CMAC este explicată în capitolul nouă.

În capitolele zece și unsprezece, sunt revizuite cunoștințele de bază ale logicii fuzzy, funcțiile de apartenență, operațiile cu mulțimi fuzzy, relații fuzzy cu mulțimi fuzzy, respectiv sistemele de inferență fuzzy Mamdani și Takagi-Suegno.

Capitol unsprezece oferă perspective asupra aplicării sistemului de inferență fuzzy de tip Mamdani.

În cea de-a treia parte este tratată implementarea hardware a rețelelor neuronale și a sistemelor de inferență fuzzy.

În capitolul doisprezece sunt detaliate constrângerile aplicate implementării hardware a rețelelor neuronale, cum ar fi aritmetica aplicată, precizia și posibilele soluții de paralelizare a algoritmilor.

În ultimele capitole sunt prezentate implementarea hardware a unei rețele neuronale cu funcții de bază radiale și a unui sistem de inferențe fuzzy de tip Takagi-Sugeno. Rezultatele prezentate în partea de implementare hardware au fost obținute pe parcursul cercetării științifice pe care am realizat-o.

Am încercat, în acest volum, să aplic experiența acumulată pe parcursul anilor în cadrul Universității Sapientia, în domeniul inteligenței artificiale.

A SZERZŐRŐL

A szerző a marosvásárhelyi Petru Maior Egyetem Villamosmérnöki Karán végzett és a Brassói Transilvania Egyetem Villamosmérnöki és Számítástechnika Tudományok Karán doktorált. 1997-től hat évig rendszergazdaként tevékenykedett a Dimitrie Cantemir Egyetem keretében. 1997-től társult oktatóként dolgozott egyetemi gyakornokként a Petru Maior Egyetemen és 1998-tól mint oktató a Dimitrie Cantemir Egyetemen. 2001-től társult oktatóként kezdett el dolgozni a Sapientia Erdélyi Magyar Tudományegyetemen, majd 2003 őszétől kezdődően főállásban egyetemi tanársegédként folytatta munkásságát a Sapientia EMTE Villamosmérnöki Karán. 2006-tól a Sapientia Erdélyi Magyar Tudományegyetem főállású adjunktusa. A fontosabb oktatott tantárgyak laborgyakorlatok szintjén számítógép- architektúrák, operációs rendszerek, mesterséges intelligencia, speciális fejezetek mesterséges intelligenciából, újrakonfigurálható digitális áramkörök, beágyazott elektronikai rendszerek, softcomputing módszerek; alapképzésen operációs rendszerek, újrakonfigurálható digitális áramkörök, valamint mesterképzésen beágyazott elektronikai rendszerek, fejlett mechatronikai rendszerek tantárgyakat oktatja. Doktori munkája során a neuronhálók FPGA áramkörben való megvalósítását tanulmányozta. A doktori munkája, majd azt követően tudományos munkája során RBF, CMAC, KOHONEN neuronhálókra, valamint Mamdani és Takagi Sugeno, ANFIS (adaptív neuro fuzzy következtető rendszer) modellekre, folytonos wavelet transzformáció, valamint egyéb algoritmusokra dolgozott ki párhuzamosított, FPGA áramkörben implementálható modelleket.

JEGYZETEK

Scientia Kiadó
400112 Kolozsvár (Cluj-Napoca)
Mátyás király (Matei Corvin) u. 4. sz.
Tel./fax: +40-364-401454
E-mail: scientia@kpi.sapientia.ro
www.scientiakiado.ro

Korrektúra:
Szenkovics Enikő

Műszaki szerkesztés:
Brassai Sándor Tihamér

Tipográfia:
Könczey Elemér

Sorozatborító:
Tipotéka Kft.

Nyomdai munkálatok:
F&F INTERNATIONAL Kft.
Felelős vezető: Ambrus Enikő igazgató

A könyv betekintést nyújt a többrétegű előrecsatolt neuronhálók, a radiális bázis függvényekből álló hálózatok, a nem ellenőrzött tanítású hálózatok, a Hopfield-hálók, a CMAC-típusú hálók, valamint a fuzzy következtető rendszerek világába. Részletezi a mesterséges neuron alkotóelemeit, az alapvető aktivációs függvényeket, a neuronhálók szerkezeti felépítését és tanítási módszereit. Bevezet a fuzzy logika világába, bemutatja a fuzzy logikában alkalmazott halmazműveleteket, az alkalmazott tagsági függvényeket, a klasszikus és fuzzy relációkat, és foglalkozik a MAMDANI- és Takagi–Sugeno-típusú következtető rendszerek tervezésével és alkalmazásával, valamint a neuronhálók és fuzzy következtető rendszerek FPGA alapú megvalósításával. Hasznos lehet mindazok számára, akik meg szeretnék érteni a neuronhálók és fuzzy következtető rendszerek elméleti alapjait, szerkezeti felépítését, a lehetséges tanítási módszereket, és gyakorlati tapasztalatra szeretnének szert tenni ezek szoftver-, illetve hardveralapú megvalósításában.

ISBN 978-606-975-021-6