**Rendszerirányításra és rendszermodellezésre alkalmazott időfüggő neurális hálók megvalósítása és tanulmányozása**



Diák: Varga Rajmond

Vezetőtanár: dr. Brassai Sándor Tihamér

Sapientia Erdélyi Magyar Tudományegyetem

Marosvásárhelyi Kar

# **1. Tartalomjegyzék:**

[**1. Tartalomjegyzék:** 2](#_Toc101102782)

[**2. Ábrajegyzék:** 4](#_Toc101102783)

[**3. Bevezető:** 5](#_Toc101102784)

[**4. A dolgozat célja:** 7](#_Toc101102785)

[**5. Elméleti megalapozás:** 8](#_Toc101102786)

[**5.1 Teljesen visszacsatolt neuronháló (Fully Recurrent Neural Network FRNN):** 8](#_Toc101102787)

[**5.2 Elman hálók** 9](#_Toc101102788)

[**5.3 Jordan hálók:** 10](#_Toc101102789)

[**5.4 Hopfield hálók:** 11](#_Toc101102790)

[**5.5 Echo state háló:** 11](#_Toc101102791)

[**5.6 Függetlenül visszacsatolt neuronhálók (independently recurrent neural network):** 12](#_Toc101102792)

[**5.7 Rekurzív neuronháló:** 13](#_Toc101102793)

[**5.8 Neural History compressor:** 14](#_Toc101102794)

[**5.9 Magasabb rendű visszacsatolt neuronhálók:** 15](#_Toc101102795)

[**5.10 Hosszú rövidtávú memória (Long Short-Term Memory: LSTM):** 15](#_Toc101102796)

[**5.11** **Kapuzott visszacsatoló egység:** 16](#_Toc101102797)

[**5.12 Bi-direkcionális visszacsatolt neuronháló (BRNN):** 17](#_Toc101102798)

[**5.13 Folytonos idejű visszacsatolt neuronhálók (CTRNN):** 18](#_Toc101102799)

[**5.15 Hierarchikus visszacsatolt neuronháló:** 18](#_Toc101102800)

[**5.16 Többszörös időskála modell (Multiple time scale modell, MTRNN):** 19](#_Toc101102801)

[**5.17 Neurális Turing gép (Neural Turing Machine):** 20](#_Toc101102802)

[**5.18 Differenciálható neurális számítógép (Differentiable neural computer):** 20](#_Toc101102803)

[**5.19 Neurális pushdown automata (Neural network pushdown automaton):** 21](#_Toc101102804)

[**6.A használt hálóstruktúrák általánosságban:** 21](#_Toc101102805)

[**7.Neuronhálók megvalósítása és tesztelése:** 24](#_Toc101102806)

[**7.1 Kibővített visszacsatolt preceptron háló:** 24](#_Toc101102807)

[**7.1.1 A háló létrehozása:** 24](#_Toc101102808)

[**7.1.2 A háló szimulálása:** 26](#_Toc101102809)

[**7.1.3 A háló tanítása:** 28](#_Toc101102810)

[**7.1.4 Mérések és elért eredmények:** 36](#_Toc101102811)

[**7.1.4.1 Elsőfokú rendszer átvitelifüggvénye:** 38](#_Toc101102812)

[**7.1.4.2 Fékezett inga:** 41](#_Toc101102813)

[**7.1.4.3 Szinuszjel generátor:** 42](#_Toc101102814)

[**7.1.4.4 PI szabályzó:** 44](#_Toc101102815)

[**7.2 Egyszerűsített teljesen visszacsatolt neuronháló:** 46](#_Toc101102816)

[**7.2.1 A háló létrehozása:** 46](#_Toc101102817)

[**7.2.2 Mérések, elért eredmények:** 48](#_Toc101102818)

[**7.2.2.1 PI szabályzó:** 49](#_Toc101102819)

[**7.3 Teljesen visszacsatolt neuronháló:** 50](#_Toc101102820)

[**7.3.1 A háló létrehozása:** 50](#_Toc101102821)

[**7.3.2 A genetikus algoritmus megvalósítása:** 51](#_Toc101102822)

[**7.3.3 Az utódok létrehozása:** 52](#_Toc101102823)

[**7.3.4 A mutációk:** 52](#_Toc101102824)

[**7.3.5 Mérések, elért eredmények:** 53](#_Toc101102825)

[**7.3.5.1 PI szabályzó:** 54](#_Toc101102826)

[**8.1 Neuronháló megvalósítása FPGA lapon:** 56](#_Toc101102827)

[**9. Következtetések:** 58](#_Toc101102828)

[**10. Irodalomjegyzék:** 60](#_Toc101102829)

# **2. Ábrajegyzék:**

[1. Ábra Neuron szerkezete 5](#_Toc101103943)

[2. Ábra Többrétegű perceptron háló 6](#_Toc101103944)

[3. Ábra Teljesen Visszacsatolt Neuronháló 7](#_Toc101103945)

[4. Ábra Elman háló 9](#_Toc101103946)

[5. Ábra Jordan háló 10](#_Toc101103947)

[6. Ábra Hopfield háló 11](file:///E:\Egyetem\AktualisFelev\Államvizsga\Dokumentáció\Dokumentáció6.0.docx#_Toc101103948)

[7. Ábra Echo state háló 12](#_Toc101103949)

[8. Ábra Függetlenül visszacsatolt neuronháló 13](#_Toc101103950)

[9. Ábra Rekurzív neuronháló 14](file:///E:\Egyetem\AktualisFelev\Államvizsga\Dokumentáció\Dokumentáció6.0.docx#_Toc101103951)

[10. Ábra Elsőrendű és Harmadrendű visszacsatolás 15](#_Toc101103952)

[11. Ábra LSTM neuron szerkezete 16](#_Toc101103953)

[12. Ábra Kapuzott visszacsatolt egység 16](#_Toc101103954)

[13. Ábra Bi-direkcionális visszacsatolt neuronháló 17](#_Toc101103955)

[14. Ábra Visszacsatolt többrétegű perceptron háló 18](#_Toc101103956)

[15. Ábra Hierarchikus visszacsatolt neuronháló 19](#_Toc101103957)

[16. Ábra Többszörös időskála modell 20](#_Toc101103958)

[17. Ábra Differenciálható neurális számítógép 20](#_Toc101103959)

[18. Ábra Kibővített visszacsatolt preceptron háló 22](#_Toc101103960)

[19. Ábra egyszerűsített teljesen visszacsatolt neuronháló 23](#_Toc101103961)

[20. Ábra Kibővített visszacsatolt preceptron hálón található visszacsatolások és súlyzók 26](#_Toc101103962)

[21. Ábra Kibővített visszacsatolt preceptron háló időbeli kiterítése 29](#_Toc101103963)

[22. Ábra Háló használatára folyamatábra 37](#_Toc101103964)

[23. Ábra 1. tanítás eredmény: elsőfokú rendszer átvitelifüggvény 38](#_Toc101103965)

[24. Ábra Háló kimenete: elsőfokú rendszer átvitelifüggvénye 39](#_Toc101103966)

[25. Ábra 2. tanítás eredménye: elsőfokú rendszer átvitelifüggvénye 40](#_Toc101103967)

[26. Ábra 3. tanítás eredménye: elsőfokú rendszer átvitelifüggvénye 41](#_Toc101103968)

[27. Ábra 1. tanítás eredménye: fékezett inga 42](#_Toc101103969)

[28. Ábra 1. tanítás eredménye: szinuszjel generátor 43](#_Toc101103970)

[29. Ábra Teszteredmény: szinuszjel generátor 44](#_Toc101103971)

[30. Ábra 1. tanítás eredménye: PI szabályzó 45](#_Toc101103972)

[31. Ábra Teszteredmény: PI szabályzó, irányított rendszer kimenete 46](#_Toc101103973)

[32. Ábra Egyszerűsített teljesen visszacsatolt neuronháló időbeli kiterítése 47](#_Toc101103974)

[33. Ábra Tanítás eredménye: PI szabályzó 49](#_Toc101103975)

[34. Ábra Teljesen visszacsatolt neuronháló második neuronstruktúra 51](#_Toc101103976)

[35. Ábra Mutáció 53](#_Toc101103977)

[36. Ábra Tanítás eredménye: PI szabályozó 55](#_Toc101103978)

[37. Ábra Hiba a neuronszám függvényében 56](#_Toc101103979)

[38. Ábra VHDL-re fordítandó háló Simulink teszteredménye 57](#_Toc101103980)

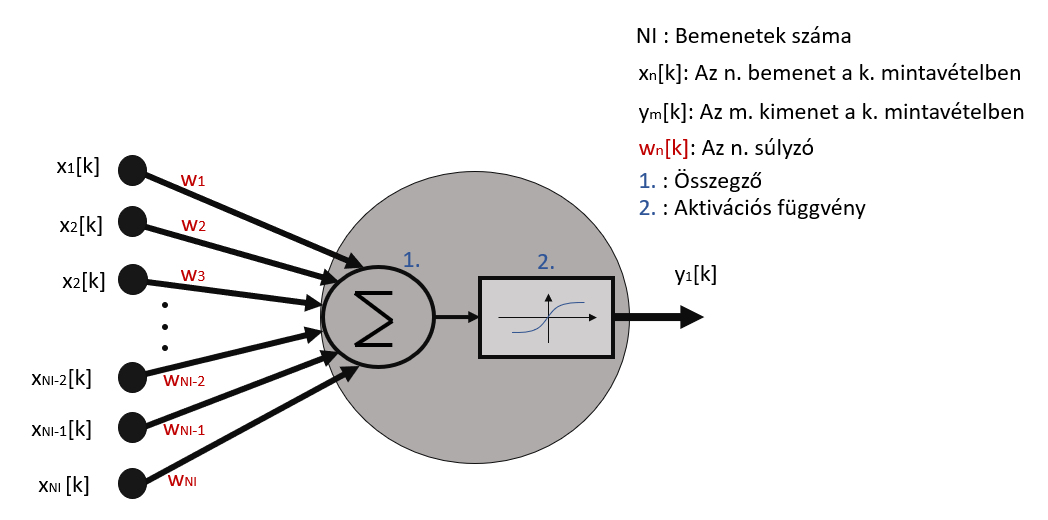
[39. Ábra Lapka kihasználtsága 58](#_Toc101103981)

[40. Ábra VHDL re fordított háló Vivado teszteredménye 58](#_Toc101103982)

# **3. Bevezető:**

A mesterséges neuronhálók egyre nagyobb teret nyernek. Ezen mesterséges struktúrák az agy működését próbálják valamilyen szinten utánozni, imitálva annak tanulásra való hajlamát, alkalmazkodó képességét és problémamegoldó képességét.

A mesterséges neuronhálók építő elemei a neuronok, melyeket valamilyen módon összekötünk, legalapvetőbb módon rétegekbe rendezzük. Egy ilyen neuron egy összegzőből és egy aktivációs függvényből áll, az összegző kimenetén megkapjuk az ingert, a bemenetek súlyozott összegét, az aktivációs függvény pedig az ingerből kiszámítja a kimenetet. Egy neuron felépítése az [1.ábrán](#ábra1) látható.

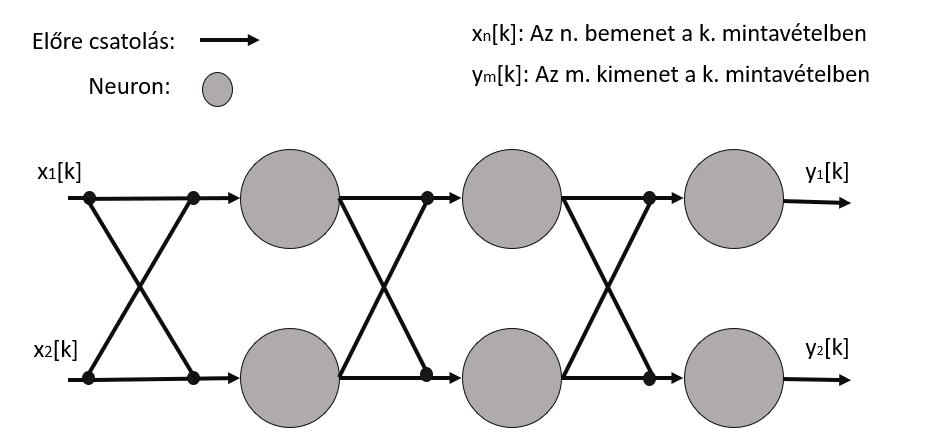


1. Ábra Neuron szerkezete

Sokféleségükből adódóan rengeteg különböző feladat elvégzésére alkalmasak, mint például mintafelismerés [[1]](#cite1), osztályozás [[2]](#cite2), jelfeldolgozás [[3]](#cite3), objektumok felismerése [[4]](#cite4), összefüggéseket képesek azonosítani nagy adathalmazok elemei között, de ezen kívül alkalmazhatóak asszociatív memóriaként [[5]](#cite5), lineáris és nem lineáris rendszerek modellezésre [[6]](#cite6), becslésére és irányítására is [[7]](#cite7).

Óriási előnyük, hogy képesek általános problémák megoldására, sok esetben a probléma beható ismerete nélkül is képesek alkalmazkodni az esetleges változásokhoz tanulás által. Képesek tanulni kizárólag mért adatókból, így sok problémamegoldást képesek felgyorsítani. Például, ha azt szeretnénk, hogy egy háló adott mintákat ismerjen fel, nekünk nem kell behatóan tanulmányozni a mintákat, és összefüggéseket felállítani közöttük, egyszerűen betanítjuk őket a hálónak és az képes lesz felismerni őket elég nagy hatékonysággal.

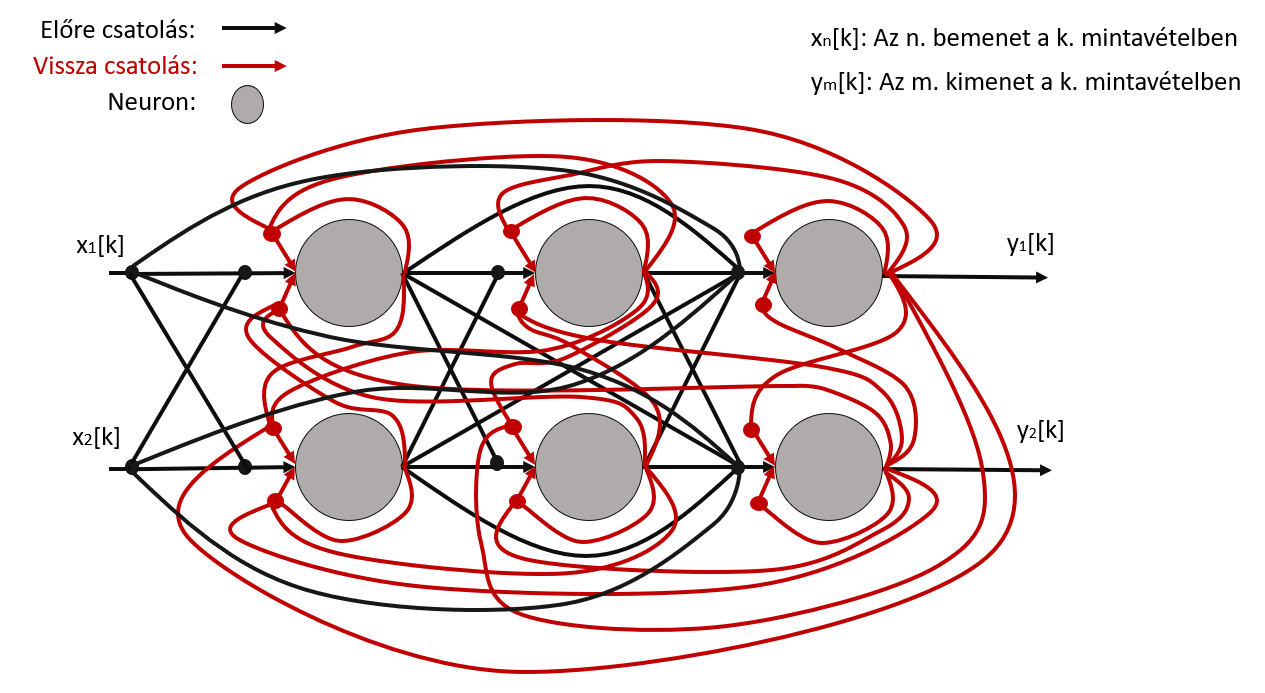
A neuronhálókat két főbb csoportba sorolhatjuk, vannak időfüggő és nem időfüggő neuronhálók. A nem időfüggő előrecsatolt neuronhálókban az információ áramlás egyirányú, a háló nem rendelkezik memóriával, a neuronháló kimenete csak a jelenlegi paramméterektől és a bemenetétől függ, nem függ a múltbeli állapotoktól és kimenetektől. Ilyen nem visszacsatolt neuronhálók például az egyrétegű vagy többrétegű perceptron, a radiális bázisfüggvényekből álló hálózat, a neurofuzzy rendszerek vagy a konvolúciós neuronháló. Több rétegű perceptronra a [2. ábrán](#ábra2) látható egy példa. A neuron tartalmazza az összegzőt és az aktivációs függvényt is.



2. Ábra Többrétegű perceptron háló

Az időfüggő neuronhálók, rendelkeznek memóriával, és képesek dinamikus rendszerként viselkedni, képesek megtanulni rendszereket csak mért adatok alapján, így megkönnyítve a rendszerek szimulációját, becslését, de irányítani is képesek a rendszereket nagy hatékonysággal.

Időfüggő neuronhálók többféle módszerrel is kialakíthatóak, a legalapvetőbb megoldás az, ha minden neuron kimenetét bekötjük minden neuron bemenetére, és a sajátjára is, és így létrehozunk egy teljesen visszacsatolt hálót. A legtöbb egyszerű visszacsatolt neuronháló struktúra kialakítható egy teljesen visszacsatolt neuronhálóból, azáltal, hogy elhagyunk csatolásokat, esetleg késleltetéseket rakunk két neuron közti csatolásra. Egy teljesen vissza csatolt neuronháló a [3. ábrán](#ábra3) tekinthető meg.



3. Ábra Teljesen Visszacsatolt Neuronháló

Ahogy a [3. ábra](#ábra3) mutatja, teljesen visszacsatolt neuronhálók esetén miniden típusú visszacsatolás megtalálható, a neuronok önmagukba való visszacsatolása, a neuronok rétegen belüli visszacsatolása, valamint a neuronok rétegek közti visszacsatolása.

A dolgozatban teljesen visszacsatolt és az ebből származtatható neuronhálók működése, megvalósítása és alkalmazása van bemutatva.

# **4. A dolgozat célja:**

A dolgozatban lineáris és nemlineáris rendszerek modellezése és irányítását tartalmazza, időfüggő neuronhálók segítségével.

A cél egy olyan környezet létrehozása, ahol különböző méretű és struktúrájú neuronhálók könnyedén létrehozhatóak, taníthatóak és szimulálhatóak, majd ezt követően ezen hálók könnyedén lefordíthatóak hardver leíró nyelvre, valamint futtathatóak újra konfigurálható digitális áramkörökön.

Az újra konfigurálható digitális áramkörökön való futtatás nagy fokú párhuzamosítást tesz lehetővé és ez nagy mértékben lecsökkentheti a háló kimenetszámításának az idejét.

A háló felépítése és tanítása Matlab környezetben történik két különböző módszer segítségével. A háló felépítésére kidolgozott első módszer során kívülről kell megadni, hogy mekkora és milyen szerkezetű hálót szeretnénk, míg a másodiknál a háló önmagától kiépül egy genetikus algoritmus segítségével, egy kiinduló hálóméretből.

A háló tanítására úgyszintén két különböző algoritmus lett kidolgozva. Az első a hagyományos gradiens visszaeresztés módszere, időben való kiterítéssel. Ennek nagy előnye, hogy viszonylag gyors és hatékony, de hátránya, hogy a dolgozatban taglalt megoldásban a hálóméret nem változhat a tanítás során. A második a már fentebb említett genetikus algoritmus, mely során nem csak kiépül a háló, hanem tanul is. Ennek nagy előnye, hogy itt a tanulás során a hálóstruktúra is módosulhat, a hátránya pedig, hogy nagyon erőforrás és időigényes. Emiatt az ésszerű megoldásnak tűnt, hogy a háló építését és tanításának elkezdését a genetikus algoritmus végzi, majd az így tanított és felépített hálót tovább tanítjuk a gradiens módszerrel, ami gyorsan és hatékonyan befejezi a tanítást.

A szimuláció Matlab és Simulink alatt egyaránt történhet. Simulink-ben könnyedén létrehozható a rendszer, amit a hálónak tanítunk, tanítottunk, ez pedig lehetővé teszi a rendszer és a háló egyidőben való párhuzamos tesztelését, így megkönnyítve a tanulás eredményének a kiértékelését.

A hardverleíró nyelvre való fordításra a Matlab biztosít eszközöket, ki tudja generálni a VHDL kódot amennyiben a Matlab kód megfelelő módon van megírva. Az így kigenerált kód már tesztelhető Xilinx Vivado környezetben és innen már egyszerűen feltelepíthető FPGA lapokra.

Több különböző teljesen visszacsatolt neuronhálóból származtatott hálóstruktúra is megvalósításra került, így megfigyelhetőek a különbségek közöttük, és van választási lehetőség, annak függvényében, hogy mire szeretnénk alkalmazni a hálót.

# **5. Elméleti megalapozás:**

Visszacsatolt Neuronhálók osztályozása:

## **5.1** **Teljesen visszacsatolt neuronháló (Fully Recurrent Neural Network FRNN):**

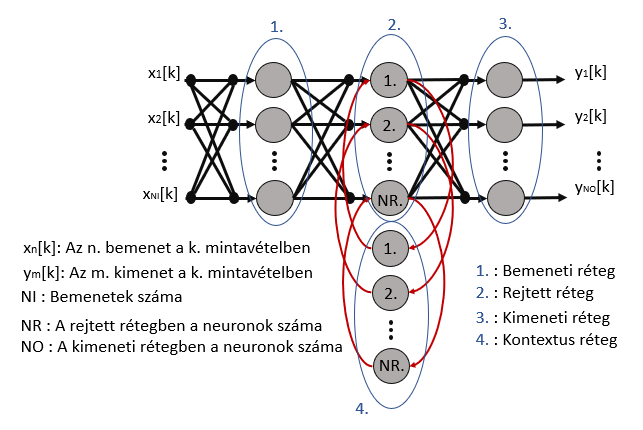
Ez a visszacsatolt neuronhálók leg alapvetőbb típusa, ebből a hálóból az összes többi visszacsatolt neuronháló kialakítható bizonyos súlyzók értékeinek a nullára való beállításával. Ezen hálóstruktúra esetében minden egyes neuron vissza van csatolva önmagába és csatolva van az összes többi hálóban levő neuronhoz is.

Ennek a hálótípusnak a legelterjedtebb ábrázolás szerint csak három rétege van, a bemeneti réteg, mely neuronok bemenetei között ott vannak a háló bemenetei, egy rejtett réteg, ahol minden neuron csatolva van a hálóban levő minden neuronhoz, mint a bemenete, mint a kimenete, valamint egy kimeneti réteg, melynek a kimenetei a többi neuron mellett, a kimenetre is rá vannak kötve.

Természetesen a megfelelő súlyzók nullára változtatásával egy több réteget tartalmazó struktúra könnyedén kialakítható.

Egy teljesen visszacsatolt neuronháló a [3. ábrán](#ábra3) látható.

## **5.2 Elman hálók**



4. Ábra Elman háló

Az Elman hálónak négy rétege van, egy bemeneti réteg, egy rejtett réteg, egy kimeneti rétege és egy kontextus rétege (context layer), mint ahogy az a [[8]](#cite8). cikkben is be van mutatva.

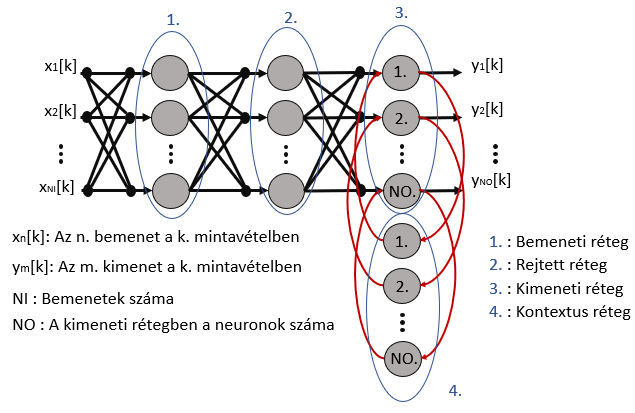
A contextus réteg memóriaként funkcionál a háló számára. Az információ a hagyományos módon halad a bemeneti rétegből a rejtett, majd a kimeneti rétegbe, azonban minden iterációban a kontextus réteg megkapja a rajtett réteg értékeit, és elmenti őket.

Ahogy a [4. ábra](#ábra4) mutatja, ezeket az elmentett értékeket majd súlyzókon keresztül visszatáplálja a rejtett rétegbe a következő iterációban.

Fontos, hogy a kontextus rétegnek ugyanannyi neuronja legyen, mint a rejtett rétegnek, hogy el tudja tárolni a rejtett réteg minden neuronjának az állapotát, mint ahogy a 4. ábrán látható.

## **5.3 Jordan hálók:**

A Jordan hálók nagyon hasonítanak az Elman hálókhoz, a kettő közt a különbség az, ahogy amíg az Elman hálók esetén a kontextus réteg a rejtett réteg kimeneteit kapja meg, addig a Jordan hálók, ahogy az [5.ábrán](#ábra5) látható, a kontextus réteg a kimeneti réteg értékeit (a háló kimenetét) értelemszerűen a contextus rétegnek itt annyi neuronja kell legyen, ahány neuron a kimeneti rétegben található.



5. Ábra Jordan háló

## A képen szöveg, óra látható Automatikusan generált leírás**5.4 Hopfield hálók:**

6. Ábra Hopfield háló

A Hopfield háló, egy viszonylag régi típusú háló. Bináris minták tárolására alkalmas, és egyfajta asszociatív memóriaként működik, ahol a memóriában nem cím szerint, hanem a memória tartalma szerint keresünk.

A hálónak lesznek stabil állapotai, és amennyiben a bemeneti adat hasonlít valamelyik ilyen állapotra, a háló el fog jutni abba a stabil állapotba, és maga a stabil állapot lesz az eltárolt bináris minta.

A stabilitás végett a neuronok nem csatolódnak vissza önmagukba, és a csatolásokon levő súlyzók szimmetrikusak, tehát két neuron között mindkét irányban ugyan az a súlyzó.

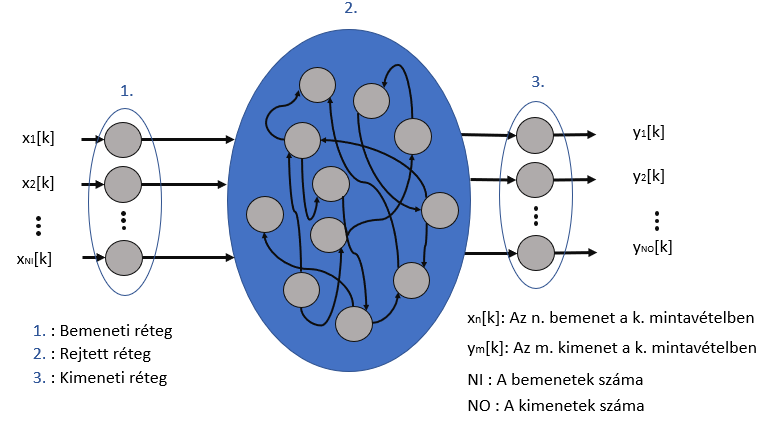
A Hoppfield hálóra egy példa a [6. ábrán](#ábra6) tekinthető meg és további információ róla a [[9]](#cite9) forrásban található.

Ennek egy továbbfejlesztett változata a Bi-direkcionális asszociatív memória, ahol a bemenet mérete nem kell megjegyezzen a kimenet méretével, mint ahogy a [[10]](#cite10) forrásban is látható.

## **5.5 Echo state háló:**

Az echo state háló egy érdekes elgondoláson alapszik. Lényegében három rétegünk van, mint ahogy a [7. ábra](#ábra7) mutatja, egy bemeneti réteg, egy random létrehozott gyéren csatol előre és visszacsatolt réteg (resovoir), és egy kimeneti lineáris réteg, ami a rejtett réteg adatait kapja meg. A rejtett rétegbe visszacsatolódnak a kimenetek is.

A rejtett rétegben a neuronok tipikusan 1%-ban vannak vissza és egymáshoz csatolva. Vannak jól megalapozott szabályok, ami szerinte ezt a rejtett réteget létre kell hozni azért, hogy a háló ne haljon el, és ne váljon instabillá, és az értékei ne fussanak el a végtelenbe. Úgy kell létrehozni ezt a réteget, hogy a bemenetek lengéseket indítsanak el itt, és azok a lengések sokáig fennmaradjanak. Amolyan neuron párok, különböző tulajdonságokkal rendelkező oszcillátorok jönnek létre, amik működését a bemenetek befolyásolják. Ebben a rétegben minden súlyzó random, egy bizonyos értéktartományon belül, de az fontos, hogy egy összekötés vagy legyen nagy értékű, hogy adatot hordozhasson, vagy legyen nulla. Ebben a hálóban elképzelhető, hogy egy bemenete több különböző belső folyamatot indít el, belső rezgéseket, amik nem is kapcsolódnak egymáshoz.



7. Ábra Echo state háló

Egy óriási előnye ennek a hálónak hagy nagyon rövid idő alatt tanul, ugyanis nem tanítom sem a bemeneti réteg súlyzóit, sem a rejtett réteg súlyzóit, egyedül a kimeneti rétegét, ami mivel lineáris, a tanulás nagyon egyszerű és gyors. A kimeneti réteg a rejtett réteg állapotaiból rakja össze a kimenetet.

A működési elv az, hogy a bemenetet a bemeneti réteg és a rejtett réteg úgymond felbontja apró részletekre, apró dinamikus folyamatokra, és ebből a sok apró részletből tudja a kimeneti réteg összeállítani a megoldást.

Ezzel a rendszerrel például megvalósítható egy jelgenerátor, ahol a bemenet a frekvencia, a kimenet pedig egy sinus jel, aminek a frekvenciája a bemenet értékével egyezik meg. Az ábra is ezt mutatja.

A hálóról további információk a [[11]](#cite11), [[12]](#cite12) és [[13]](#cite13) forrásokban tekinthetőek meg.

## **5.6 Függetlenül visszacsatolt neuronhálók (independently recurrent neural network):**

A függetlenül visszacsatolt neuronhálók legfőbb célja, hogy megoldást adjanak a gradiens eltűnés (gradient vanishing) és a gradiens felrobbanás (gradient expodig) problémákra melyekről részletes leírás a [[14]](#cite14) forrásban található. Ezek jelentős problémát jelenthetnek amennyiben a háló mély vagy sokszor terítünk vissza időben. Felrobbanó gradiens során az történik, hogy a gradiens értéke a visszaáramoltatás során egyre csak növekszik. Ez a tanítás instabilitásához vezethet. Ezzel szemben az eltűnő gradiens azt feltételezi, hogy a gradiens értéke egyre csökken, ahogy haladunk vissza rétegről rétegre, ez nagyban lelassíthatja, de akár le is állíthatja a tanulást.

A képen szöveg, sportjátékok, sport látható

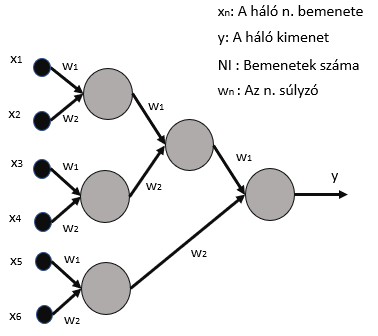
Automatikusan generált leírás

8. Ábra Függetlenül visszacsatolt neuronháló

Az IndRRN, mint ahogy a [[15]](#cite15) forrásban is részletezve van, megengedik, hogy szabályozzuk az a gradiens visszaáramlását, így elkerülve a fentebb említett problémákat, az által, hogy rétegenként a neuronok nincsenek egymáshoz csatolva, minden neuron csak a bemenetről, és a saját rejtett állapotából az előző mintavételből kap információt, min ahogy a [8. ábra](#ábra8) mutatja. A bemenet a háló bemenete, vagy az előző réteg kimenete. Egyes kutatások azt mutatják, hogy egy ilyen háló akár jobban is teljesíthet a fentebb említett problémák kiküszöbölésében, mint egy LSTM háló, ami egy szintén elterjedt megoldás a problémákra.

## **5.7 Rekurzív neuronháló:**

A hálón az információ áramlás úgy történik, hogy egy strukturált bemenetre alkalmazzuk újra és újra ugyanazon súlyzókat rekurzívan.

 A háló egy fa struktúrát reprezentál, amit balról jobbra járunk be, ahogy a [9. ábra](#ábra9) mutatja, és folyamatosan dolgozzuk fel az információt úgy, hogy mindig két gyermekből lesz egy szülő a globális súlyzómátrix és egy aktivációs függvény alkalmazásával.

9. Ábra Rekurzív neuronháló

A leggyakrabban természetes nyelv feldolgozásban használtak, mint ahogyan az a [[16]](#cite16) forrásban is le van írva.

A tanítását leggyakrabban sztochasztikus gradiens módszerrel végzik, és a gradienst egy módosított visszaterjesztés (backpropagation) algoritmus segítségével végzik.

## **5.8 Neural History compressor:**

Az ilyen típusú neuronhálókról több megközelítést is találni lehet, melyek kis mértékben térnek el egymástól.

Az egyikben a struktúrát több visszacsatolt neuronháló alkotja. Ezen hálókat nem felügyelt elő tanítással tanítottuk. A rétegek a következő bemeneteket próbálják jósolni, az előző bemenetek alapján és csak azok a bemenetek adódnak tovább a következő magasabb rendű rétegnek, amik hibát generálnak.

A másodikban ugyanúgy több visszacsatolt háló található, de itt a következő rétegnek véletlenszerűen kiválasztott kimenetek adódnak át, és a következő rétegnek, ami úgy a belső állapotát csak ritkítottan frissíti.

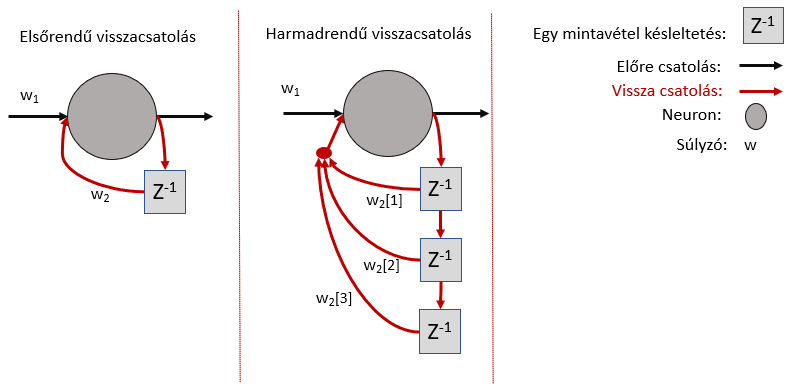
Ez a háló, az [[17]](#cite17) forrásban elmondottak alapján, nem veszít adatot a rétegeken keresztül, csak tömöríti azt. A bemeneti adat visszaállítható a legfelső kimenet alapján.

## **5.9 Magasabb rendű visszacsatolt neuronhálók:**

A magasabb rendű visszacsatolt neuronháló, mint ahogy a [[18]](#cite18) forrásban is olvasható, abban különbözik a hagyományos, elsőrendű visszacsatolt neuronhálótól, hogy itt nem csak az előző mintavételből csatoljuk vissza a háló állapotát, hanem az azt megelőző mintavételekből is.

Ezt úgy érjük el, hogy megtöbbszörözzük a visszacsatolásokat és minden visszacsatoláson növeljük a késleltetés mértékét.

A [10. ábrán](#ábra10) azt láthatjuk, hogy egy neuron önmagába való visszacsatolása hogyan változik, ha első rendű visszacsatolásról áttérünk harmadrendű visszacsatolásra. Itt fontos megjegyezni, hogy ez jelentősen növeli a háló súlyzóinak a számát, hiszen minden visszacsatolás külön súlyzón keresztül történik.

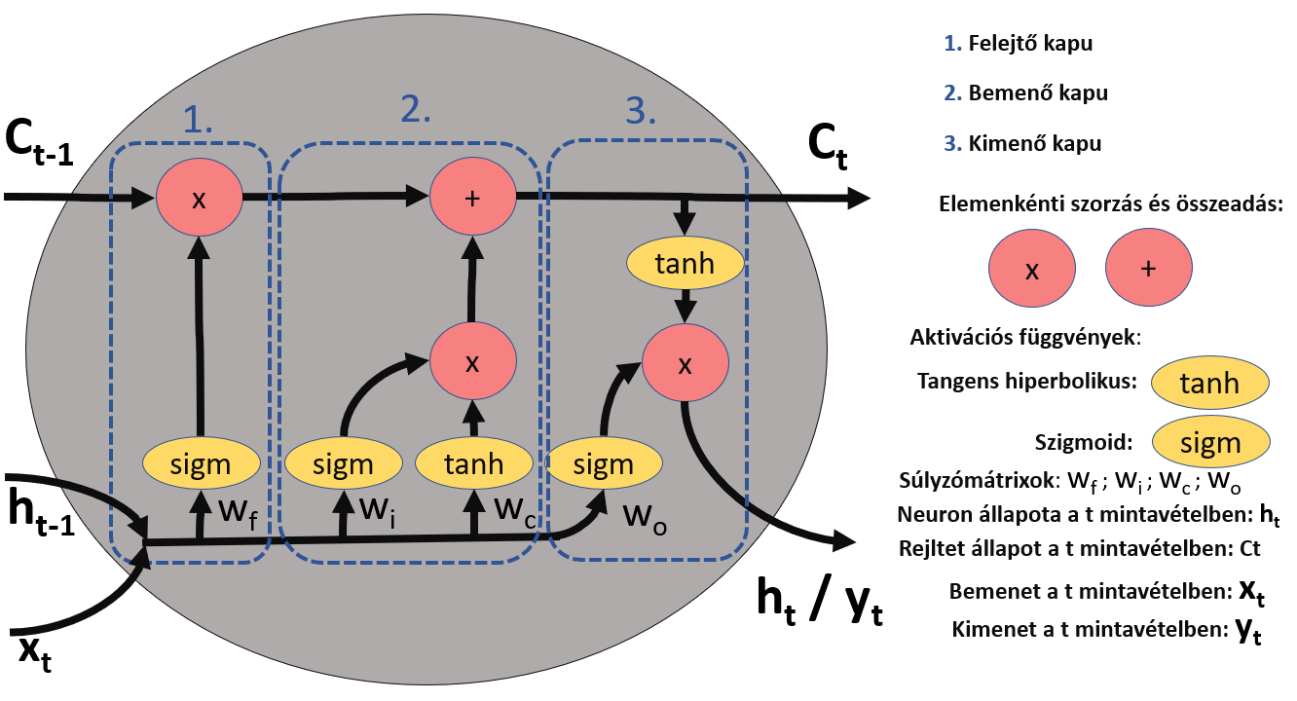


10. Ábra Elsőrendű és Harmadrendű visszacsatolás

## **5.10 Hosszú rövidtávú memória (Long Short-Term Memory: LSTM):**

Az LSTM neuronháló egy az előzőkénél jóval komplexebb neuronszerkezetet feltételez, ahogy az a [[19]](#cite19) cikkben is olvasható és a [11. ábrán](#ábra11) is látható. Első sorban az eltűnő és felrobbanó gradiens problémákat hivatott megoldani és lehetővé teszi a hiba visszaáramoltatását egy alternatív útvonalon.

Az által, hogy az információ áramlását több úton szabályozni tudjuk, bizonyos mértékben súlyozni tudjuk a jelenlegi és a múltbeli értékeket, így beállítva, hogy mennyire legyen hosszútávú vagy rövidtávú a háló memóriája. Ezek a szabályzások úgynevezett kapukon keresztül törnének, a felejtő, bemenő és kimenő kapukon.

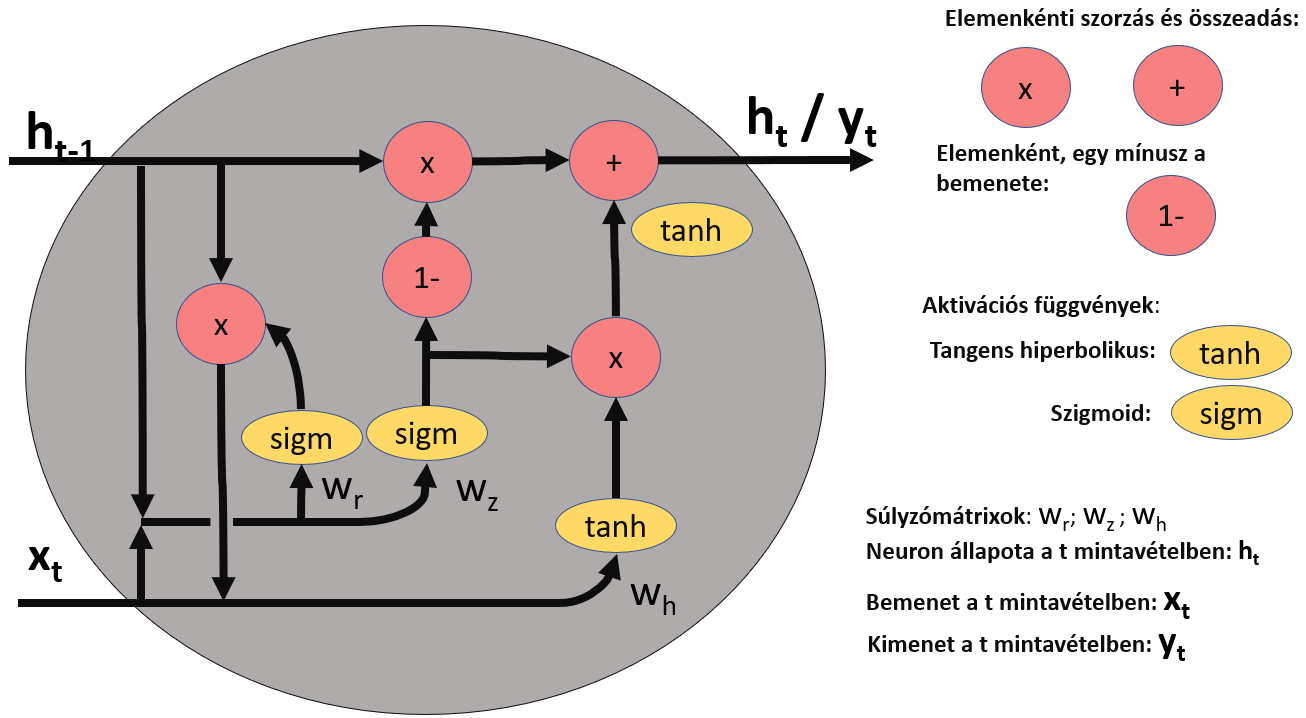


11. Ábra LSTM neuron szerkezete

Ahogy az ábra is mutatja, a neuronnak több bemenete és kimenete van, és ezeken keresztül több úton csatolódnak vissza és csatolódnak egymáshoz. Az információ és majd hiba, a neuronon belül két párhuzamos úton áramlik, és ahogy az látható, a felső csatornán nincs mátrixokkal való szorzás, így ott a visszaáramoltatás során nem áll fent a gradiens eltűnés vagy felrobbanás.

Ezenkívül fontos észrevenni, hogy egy LSTM neuron már több aktivációs függvénnyel is rendelkezik, a hagyományos neuronhoz képest és ahogy a [[20]](#cite20) forrásból is látszik, a hiba visszaáramoltatás is bonyolultabb lesz.

## **5.11** **Kapuzott visszacsatoló egység:**

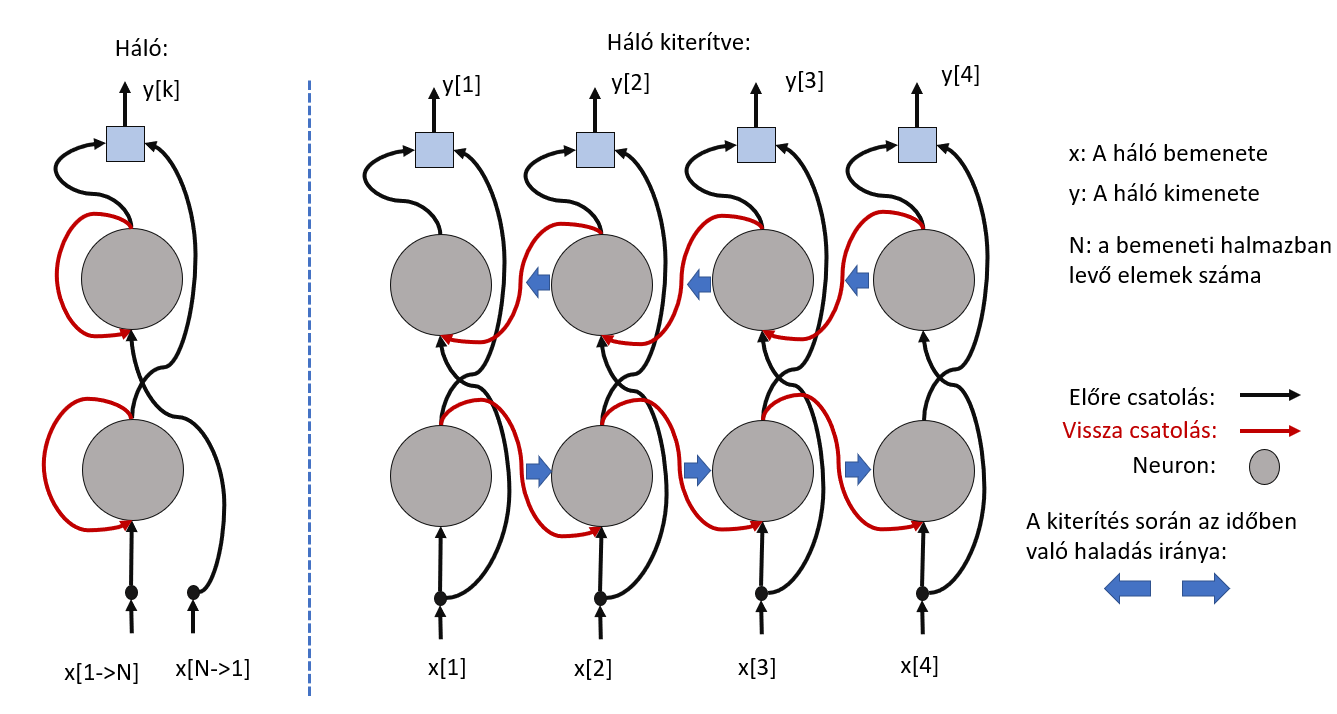
****

12. Ábra Kapuzott visszacsatolt egység

Ennek a hálónak többféle változata lehet. Egy alapvető neuron struktúra az alábbi képen látható.

Itt az LSTM hálóhoz hasonlóan a háló állapotát és kimenetét a bemente és az előző állapot különböző kombinációjából számolom. A fő különbség, hogy itt nincs kimeneti kapu, mint az LSTM esetében, mint azt a [12. ábra](#ábra12) is mutatja.

## **5.12 Bi-direkcionális visszacsatolt neuronháló (BRNN):**



13. Ábra Bi-direkcionális visszacsatolt neuronháló

Az egyszerű MLP és RNN hálókkal ellentétben ez a háló nem csak egy irányból, az aktuális és múltbeli bemenetekből, állapotokból vesz információt, hanem a jövőbelikéből is. Ezt úgy éri el, hogy két hálót kombinál, az egyik egy irányba járja végig a bemeneti halmazt, az elejétől a vége fele, a második pedig fordítva, a végétől az elejéig.

Bi-direkcionális visszacsatolt neuronhálóra a [13. ábrán](#ábra13) láthatunk egy példát. Fontos megjegyezni, hogy a kimenet tényleges kiszámításához várni kell amíg mindkét háló elér az adott bemeneti mintavételhez. A kimenetet úgy kapjuk meg hogy összevágjuk a két háló által számolt értékeket, így természetesen szükség van arra, hogy mindkét háló kimenete ki legyen számolva arra a mintavételre. A legelső kimenet amennyiben a két háló párhuzamosan számol, a halmaz közepén lesz, hiszen a halmaz két végéről induló hálók ott találkoznak először. Ezt követően minden mintavételben két kimeneti értéket számolhatunk ki.

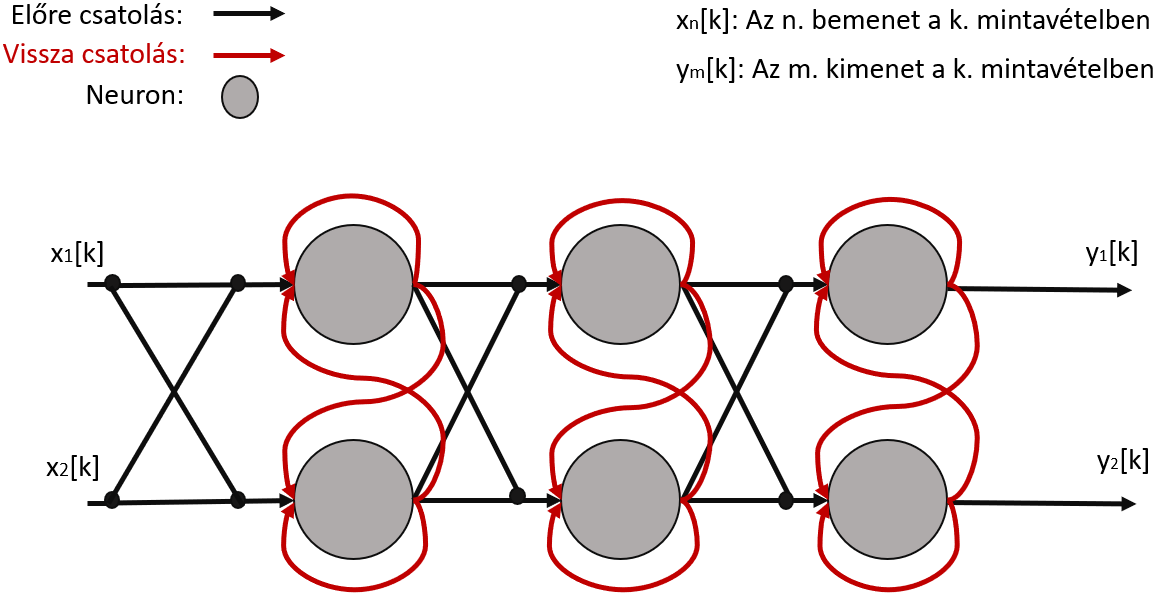
Ezen hálók különösen hatékonynak bizonyultak, ha kombinálták őket RNNs vagy LSTM típusú hálókkal.

## **5.13 Folytonos idejű visszacsatolt neuronhálók (CTRNN):**

A folytonos idejű visszacsatolt neuronhálók differencia vagy differenciálegyenleteket használnak a neuronok bemenet függvényében történő kimenetszámításához.

Ezen háló struktúrák jó eredményeket mutattak az evolúciós robotika területén, ahol a robotok kialakítása szimuláció útján történik természetes kiválasztódás útján. Hasonló alkalmazás a [[21]](#cite21) forrásban tekinthető meg.

**5.14 Visszacsatolt többrétegű preceptron háló:**



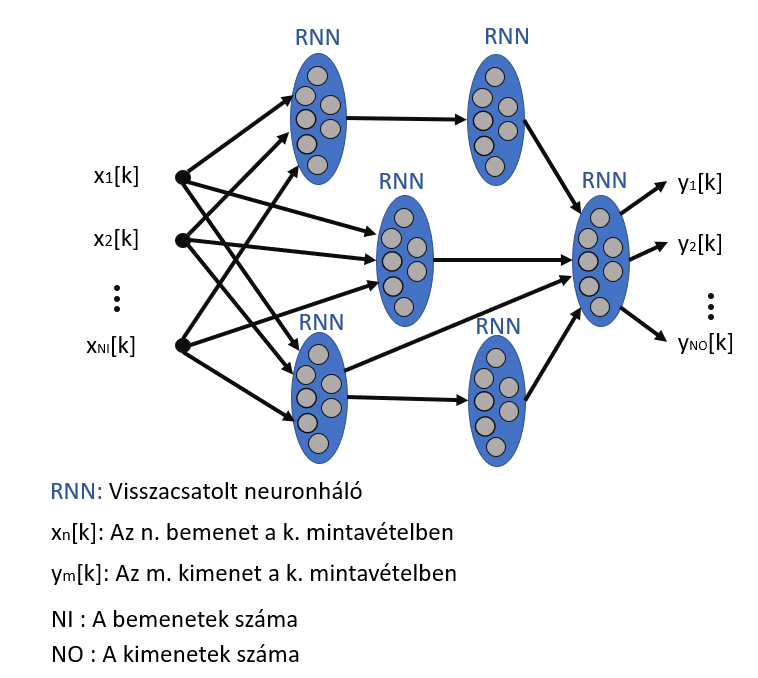
14. Ábra Visszacsatolt többrétegű perceptron háló

A visszacsatolt többrétegű perceptron, ahogy a [[22]](#cite22) forrásban is megtekinthető, több alhálóból épül fel Az alhálók egymás után sorba vannak kötve. Egy alháló, egy teljesen visszacsatolt neuronréteg, mint ahogy az a [14. ábrán](#ábra14) is látható. Az alhálók csakis előre vannak csatolva egymáshoz, kivéve az utolsó háló, ahol megengedett, hogy az vissza legyen csatolva az előző rétegekbe bizonyos esetekben.

## **5.15 Hierarchikus visszacsatolt neuronháló:**

A hierarchikus hálókban lényegében több alhálóra bontjuk a nagy egészet, ahogyan a [15. ábra](#ábra15) is mutatja. Minden alhálónak megvan a maga feladata, amit majd egy mások alháló összefoglal. Fontos megjegyezni, hogy hálók nem kell egyforma típusúak legyenek, például az egyik lehet egy teljesen visszacsatolt neuronháló, míg a másik egy Jordan háló

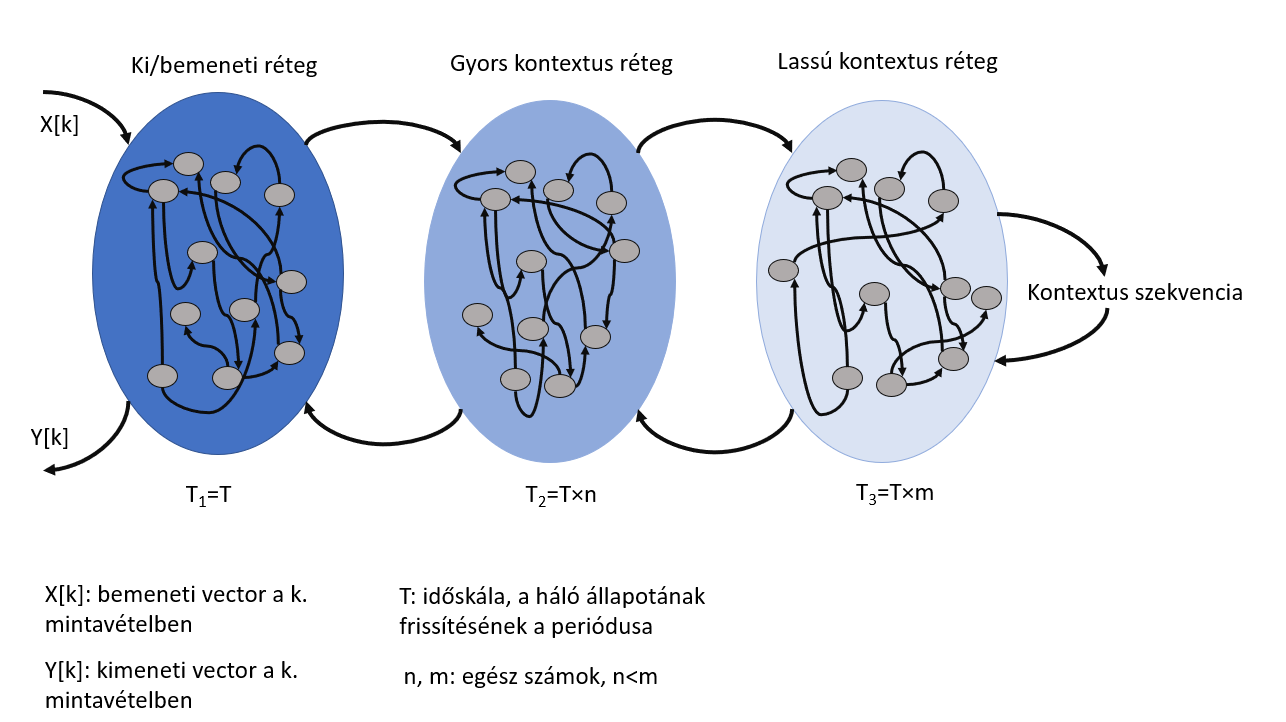
Hasznosnak bizonyult, a [[23]](#cite23) forrás alapján, például videófelvételek összefoglalásában, ahol egy alháló felelt azért, hogy kódoljon kis részleteket az eredeti videóból, amíg a másik alháló ezen réteg kimenetéből eldöntötte, hogy egy fontos részlet volt az eredeti videóból, vagy sem.



15. Ábra Hierarchikus visszacsatolt neuronháló

## **5.16 Többszörös időskála modell (Multiple time scale modell, MTRNN):**

Ez a háló az agy működését hivatott leutánozni valamilyen szinten. Több különböző időskálával rendelkező folytonos idejű visszacsatolt neuronháló együttes működését feltételezi, hagyományos esetben háromét, ahogy a [16. ábra](#ábra16) is mutatja, a leggyorsabb, egy ki/bemeneti réteg, a következő egy gyors kontextus réteg és a leglassabb egy lassú kontextus réteg. A leg lassabb réteg kiemeli az adat kontextusát, az utolsó mintavételi periódusából, valamint itt vihetünk be kívülről is adatokat a rendszerbe. A nyelvi adatok feldolgozása mellett kiválóan alkalmas motorvezérlésre [[24]](#cite24) a háló a kontextus kimenetén generálta a motorvezérlőjel szekvenciát. Ebben az esetben csak egy irányban használták a háló ezen ki/bemenetét.

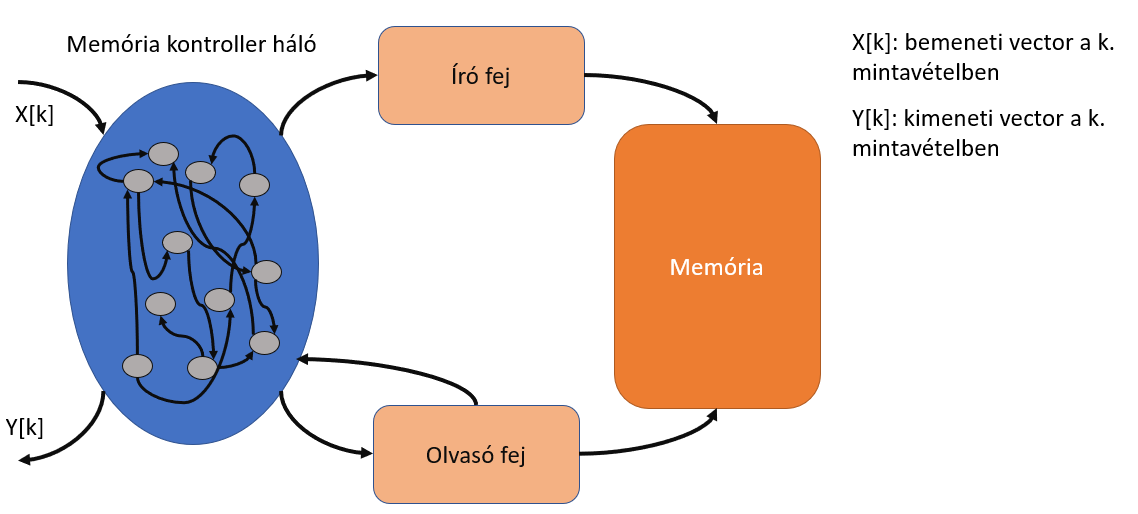


16. Ábra Többszörös időskála modell

## **5.17 Neurális Turing gép (Neural Turing Machine):**

A neurális Turing gép, első sorban abban különbözik egy hagyományos visszacsatolt neuronhálótól, hogy itt a neuronhálót hozzácsatoljuk egy külső memóriához. Az így kialakított modell tanítható gradiens módszerrel. Alkalmazható például egyszerűbb műveletek, mint másolás elvégzésére.

## **5.18 Differenciálható neurális számítógép (Differentiable neural computer):**



17. Ábra Differenciálható neurális számítógép

Az a háló a neurális Turing gép egy továbbfejlesztett változata. A legfőbb különbség az, hogy itt több módszer van a memória elérésére, ahogy a [17. ábrán](#ábra17) is látható, komplexebb módon tud maga a háló kommunikálni, az is el van tárolva, hogy milyen sorrendben voltak az adatok beírva és kiolvasva a memóriából.

A [[25]](#cite25) forrás alapján a háló működése nem függ a memória méretétől, amíg a memória meg nem telik. A háló egy memória kontrollerként viselkedik.

A differenciálható neurális számítógép nem minden esetben épül visszacsatolt neuronhálókra.

## **5.19 Neurális pushdown automata (Neural network pushdown automaton):**

Ez a hálótípus is hasonló a neurális Turing géphez. A legfőbb különbség köztük az, hogy itt a memória egy verem, a [[26]](#cite26) forrásban elmondottak alapján.

# **6.A használt hálóstruktúrák általánosságban:**

A kiinduló hálóstruktúra a [3. ábrán](#ábra3) szemléltetett, teljesen visszacsatolt neuronháló, a továbbiakban bizonyos egyszerűsítésekkel és módosításokkal ez van felhasználva.

Azért esett a választás erre a struktúrára, mert ebből könnyedén kialakíthatóak egyszerűbb struktúrák, akár a tanulás során is, leegyszerűsödhet a háló például egy visszacsatolt többrétegű preceptron hálóvá, azáltal, hogy bizonyos súlyzók értéke megközelíti a nullát.

A másik fontos szempont a háló struktúrájának az egyszerűsége, hiszen a cél az, hogy a hálót újra konfigurálható digitális áramkörön lehessen futtatni, amit nagyban megkönnyít és könnyebben kivitelezhetővé tesz, ha a neuron struktúrája egyszerűbb. Amennyiben egy LSTM háló lett volna felhasználva, a neuron belső szerkezete sokkal bonyolultabb lett volna, így sokkal erőforrásigényesebb lenne a háló. Minél egyszerűbb a struktúra, annál könnyebb implementálni hardver leíró nyelven és annál kevesebb erőforrást igényel a háló futtatása az FPGA lapokon.

A teljesen visszacsatolt hálóból többféle struktúra van kialakítva, ezekben különbözik a rétegek összekapcsolása és a bemenetek bekötése a hálóba. Ennek következtében különböző előnyöket és hátrányokat figyelhetünk majd meg az egyes hálóstruktúrák esetében.

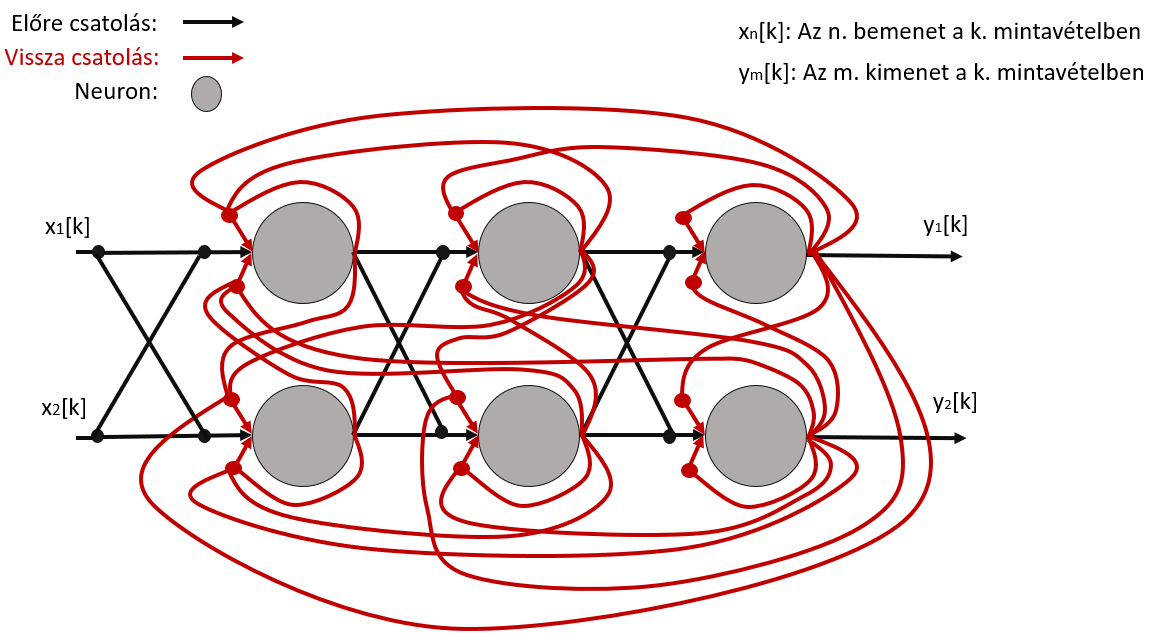
Ezen kívül különböző ábrázolási módok vannak alkalmazva, annak függvényében, hogy a tanítási algoritmusokat hogyan könnyebb megvalósítani.

Minden használt hálóstruktúra esetén a biasok úgy vannak alkalmazva, hogy a háló bemeneteit tartalmazó vektor és a kimeneti réteg kivételével, minden réteg kimenetét tartalmazó vektor ki van egészítve egy egyessel, mely az első elem lesz a vektorból.

Az **első** használt hálóstruktúra, melyet nevezhetünk **kibővített visszacsatolt preceptron hálónak**, hiszen hasonló ahhoz, abban különbözik a teljesen visszacsatolt háló struktúrájától, hogy a háló rétegelt szerkezetű, és minden réteg csak a következő rétegbe csatolódik előre, ahogy az [18. ábra](#ábra18) is mutatja. Ez azt jelenti, hogy az első réteg kimenete visszacsatolódik önmagába, és a második rétegbe, de nem csatolóik a 3. rétegbe. Viszont bemenetnek megkapja a háló bemenetét és az összes következő réteg kimenetét is. Tehát minden réteg visszacsatolódik az összes előző rétegbe, viszont a következő rétegek közül csak a közvetlen utána levő kapja meg a réteg kimenetét.

A rétegek azonban felfoghatóak úgy, mint teljesen visszacsatolt neuronhálók.

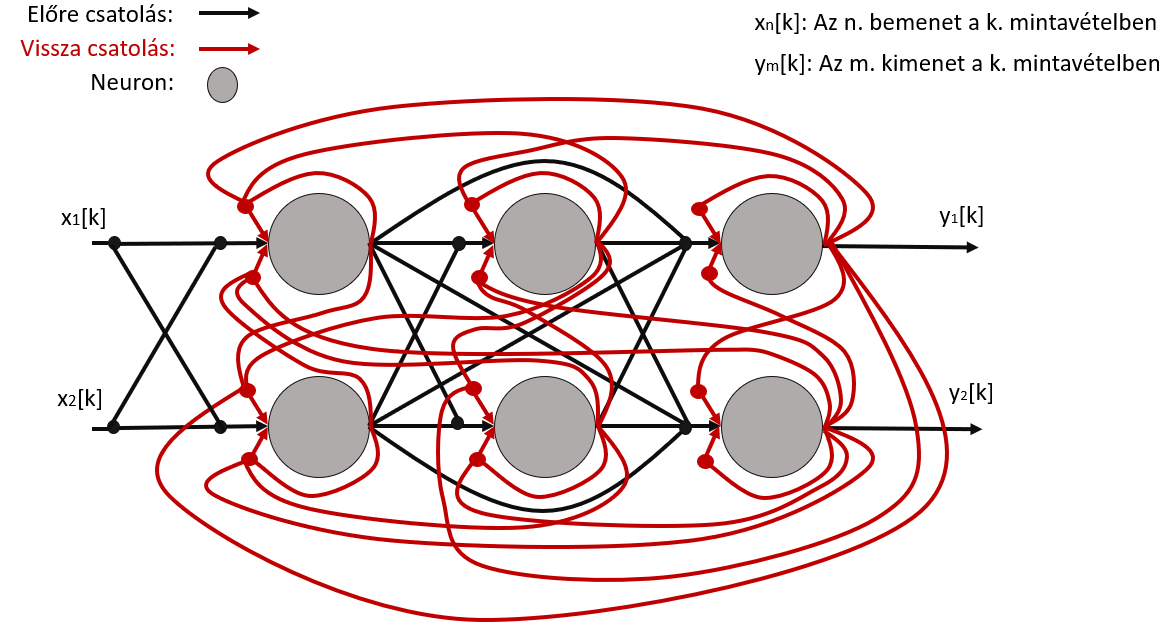
Ezen hálóstruktúra nagy előnye, hogy a rétegek számával kontrollálni lehet, hogy az információ hány iteráció alatt jut el a bemenetről a kimenetre, valamint a későbbi tesztelések arra is fényt derítettek, hogy ez beépíthet egyfajta szűrőszerű viselkedést a hálóba, a középső rétegek megszűrik a bemenetet, ennek hála a lépcsőszerű változások nem okoznak hirtelen ugrásokat a kimenetben, de ezt a későbbiekben részletesen bemutatásra fog kerülni.



18. Ábra Kibővített visszacsatolt preceptron háló

A **második** struktúra esetében, melyet nevezhetünk **egyszerűsített teljesen visszacsatolt neuronhálónak,** mivel már csak nagyon kis mértékben tér el tőle, minden réteg megkapta az összes előző réteg kimenetét is, de a bemenetet még mindig csak a bemeneti réteg kapja meg. Ez lecsökkenti a hálón az információ átjutásának idejét, úgy, hogy még mindig védi valamennyire a kimenetet a bemeneten megjelenő nagy ugrásoktól. Azonban itt fontos, hogy amennyiben háromnál több rétegünk van, a belső rejtett rétegek tekinthetőek egy rétegnek is.

Egy ilyen háló szerkezete az [19. ábrán](#ábra19) tekinthető meg.



19. Ábra egyszerűsített teljesen visszacsatolt neuronháló

A **harmadik** kipróbált struktúra, a kiinduló struktúra, a **teljesen visszacsatolt neuronháló**, amely a [3. ábrán](#ábra3) látható. Itt kerül bevezetésre, hogy már nem csak mátrixos formában tárolódik a háló, hanem egy neuronokat tartalmazó tömbként. Minden neuron egy különálló struktúra saját súlyzókkal és aktivációs függvénnyel. Ez a struktúra nagyban megkönnyíti a háló olyan módon való tanítását, ahol nem csak a neuronok súlyzói változnak a tanítás során, hanem akár az aktivációs függvények is, valamint a háló mérete sem állandó, hanem tanulás által a hálóba bejöhetnek új neuronok és kieshetnek régiek. Így a háló felépíti saját magát.

A háló tanítása evolúción alapuló darwini elvek alapján történik. Több háló verseng a populációban, a legjobbak maradnak és mutációkkal utódokat hoznak létre, míg a rosszabbak törlődnek. Az így felépített hálót ezt követően egy algoritmus segítségével mátrixos formára lehet hozni, hogy a kialakult hálót az előző struktúráknál használt gradiens módszerrel is tanítani lehessen.

Ez a háló viszonylag gyorsan és megbízhatóan tanult, de nagy hátránya, hogy amiatt, hogy a bemenetet minden neuron megkapja az ugrások a bemeneten azonnal tovább terjednek a kimeneten is.

Itt megjegyzendő, hogy amennyiben több rétegünk van, az egyenértékű egy olyan kétrétegű hálóval, ahol a kimeneti neuronokon kívül minden neuron az első rétegben van, mivel a bemeneteket minden neuron megkapja minden esetben a hálóban.

Ezenkívül fontos, hogy amennyiben egy rétegű hálóról beszélünk, a három struktúra megegyezik. A bevezetésükre azért volt szükség, hogy lehetőség, legyen a kimenet bemenettől való leválasztásainak különböző szintjeit tanulmányozni.

# **7.Neuronhálók megvalósítása és tesztelése:**

## **7.1 Kibővített visszacsatolt preceptron háló:**

A háló generálásához és használatához négy fontos feladat megértése szükséges, az első a háló létrehozása, a háló szimulálása, tanítása, majd tesztelése. Ezek közül kettő nagyon közel áll egymáshoz, a szimulálás és a tesztelés.

Első lépésben a háló felépítését és ábrázolási módját szükséges megismerni. A háló tárolásához, a súlyzókat és azok elhelyezkedését kell ismerni.

### **7.1.1 A háló létrehozása:**

A súlyzók mátrixok formájában vannak eltárolva. A háló két tetszőleges réteg közti kapcsolatot mindig egy súlyzómátrix írja le, tehát annyi súlyzómátrixra van szükségünk, ahány csatolt réteg van a hálóban. Első sorban minden rétegnek kell egy mátrix, ami az előző réteghez, vagy az első réteg esetén a bemenethez csatolja. Ezek a mátrixok fogják tartalmazni az előre csatolásokon levő súlyzók értékeit, valamint a biasokat bekötő súlyzók értékeit is, mint ahogy az [20. ábra](#ábra20) is mutatja. Az előre csatolásokat és a biasokhoz kötő súlyzókat tartalmazó mátrixok a nevű cella tömbben foglalnak helyet.

Ezt követően minden rétegnek kell annyi visszacsatolásokat tartalmazó mátrix, ahány réteg csatolódik vissza bele. Az [20. ábrán](#ábra20) látható, hogy ez a  nevű kétdimenziós cellatömb.

A súlyzó mátrixokat kezdetben random értékekkel töltjük fel a következő algoritmus alapján:

Minden i: 1->NL

Ha i=1

Máskülönben:

Vége ha;

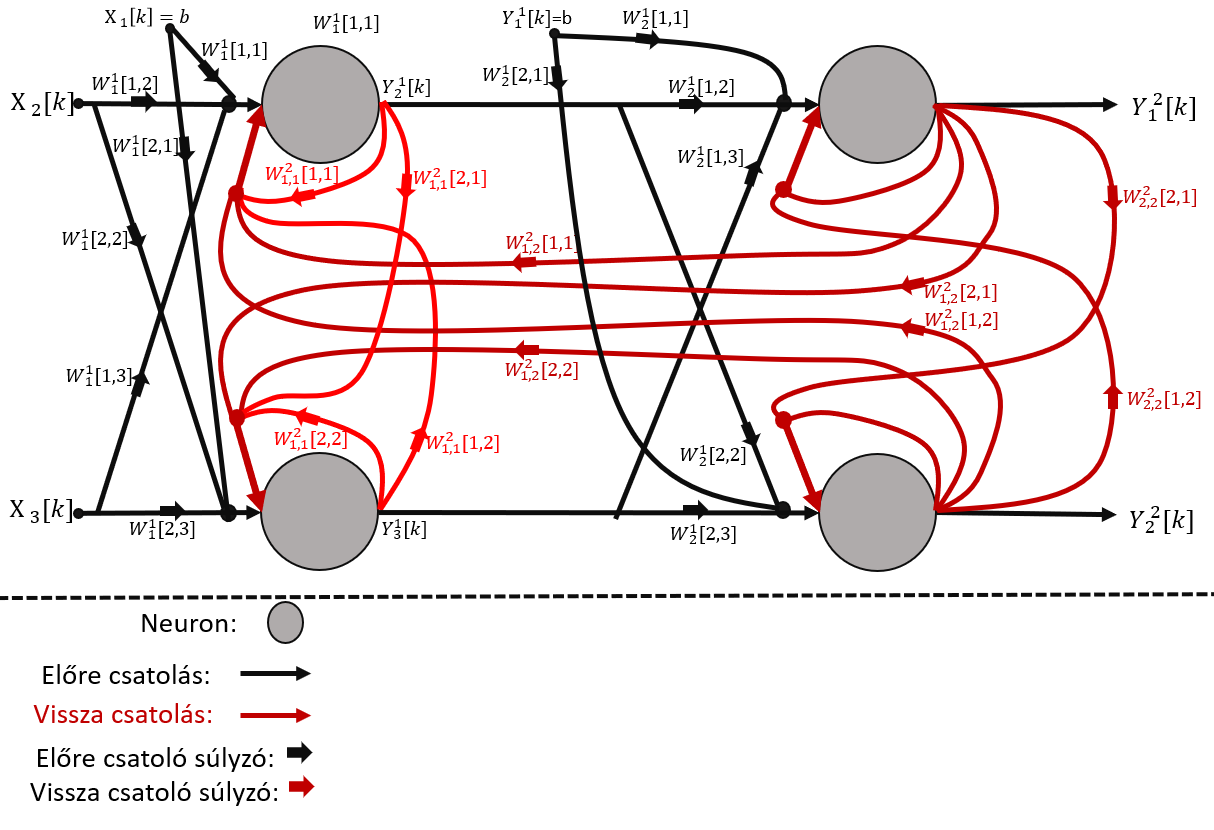
Minden j=1->NL

Vége minden

Vége minden

Jelölések és magyarázat:

* NL: rétegek száma.
* NI: a háló bemeneteinek a száma (nem tartalmazza a biast, ezért ennél eggyel mindig több súlyzóra van szükség)
* NNL(i): az i. rétegben levő neuronok száma (mindig eggyel több súlyzóra van szükség az előre csatolásoknál alkalmazott biasok miatt)
* random (n, m): létrehoz egy random értékekkel feltöltöttöt mátrixot, aminek n sora és m oszlopa van. Az értékek egy és nulla között vannak, kivonva 0.5-t az értékeket [0 1] közötti intervallumra képezzük le.



20. Ábra Kibővített visszacsatolt preceptron hálón található visszacsatolások és súlyzók

Az [20. ábrán](#ábra20) található jelölések:

: A háló n. bemenete a k. mintavételben (az első eleme a bias)

: Az m. réteg az n. kimenete a k. mintavételben (az első eleme a bias)

: Az n. réteg i. bemenetét és n-1. réteg j. kimenetét (a háló j. bemenetét, amennyiben az első rétegben vagyunk) összekötő előre csatoló súlyzó.

*:* Az n. réteg i. bemenetét és m. réteg j. kimenetét összekötő előre csatoló súlyzó.

: A bias, az ide kapcsolódó bemenet értéke mindig egy.

### **7.1.2 A háló szimulálása:**

A háló szimulálása szinkron módon történik. Ez azt jelenti, hogy egy adott mintavételben a háló minden neuronjának az értéke egyszerre számolóik az adott bemenetre, és a neuronok előző mintavételben levő kimeneteire. Ez azt eredményezi, hogy ahhoz, hogy egy bemenet hatása annyi mintavétel után jelenhet meg a kimeneten, ahány rétege van a neuronhálónak.

Fontos megjegyezni, hogy a bemenetekhez ( ) már hozzá van csatolva az egyes a biasok miatt.

A háló kimentének a számítása a következő algoritmus alapján történik:

1. Minden i: 1->NL
3. Vége minden
4. Minden i: 1->DL
5. //sorvektor, oszlopvektor
6. Minden j: 2->NL+1
7. // Az egy a biasok miatt kerül oda
8. Vége minden
9. Minden j: 1->NL
10. // oszlopvektor
12. Minden k: j->NL
14. Vége minden
16. Vége minden
17. Minden j: 1->NL
19. Vége minden
20. // oszlopvektor
21. Vége minden

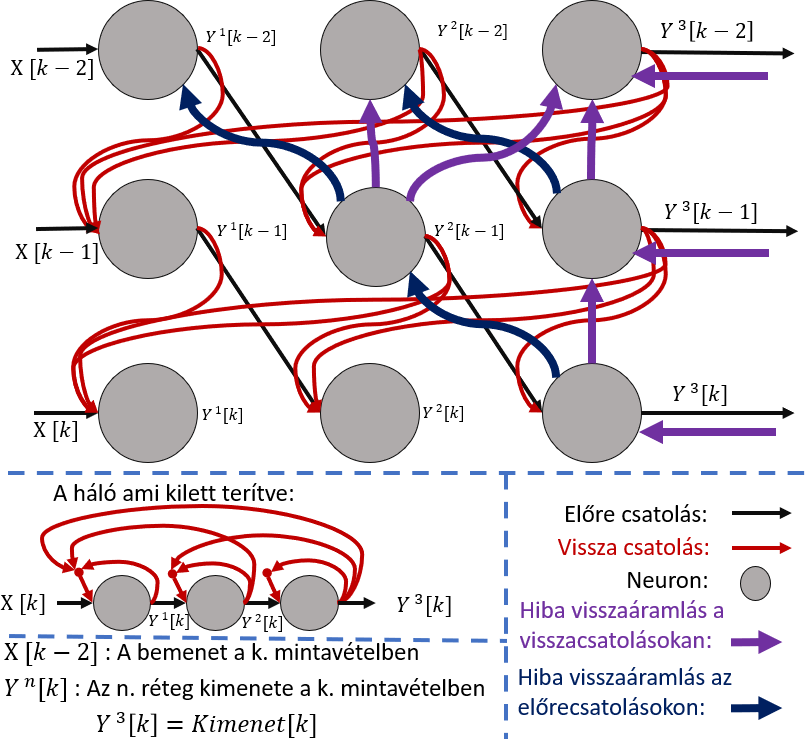
Jelölések és magyarázat:

* : az első iterációban az i. réteg kimenete, kezdetben az értékét nullára állítjuk.
* : létrehoz egy nullmátrixot, aminek n sora és m oszlopa van.
* : A neuronok kimeneteinek kezdeti értéke.
* DL: A tanítóhalmazban levő elemek száma.
* : oszlopvektor, a háló n. rétegének a neuronjainak a bemeneteit tartalmazza.
* sorvektor, a neuronháló bemenete az i. iterációban.
* : oszlopvektor, az első eleme egyes, utána a háló neuronjainak a kimeneteit tartalmazza az i. iterációból, a j-1. rétegből
* : oszlopvektor, a háló j. rétegében levő neuronokban számolt előrecsatolásokból származó összegeket tartalmazza a j. iterációból
* : oszlopvektor, a háló j. rétegében levő neuronokban számolt visszacsatolásokból származó összegeket tartalmazza a j. iterációból
* : megmondja, hogy -nek hány sora és oszlopa van.
* : megmondja, hogy -nek hány eleme van.
* : -nek az első elem kivételével minden eleme (az első elem azért marad ki, mert az egy bias, nem egy neuron kimenete)
* : a j. rétegben levő neuronokban számolt összeg, tartalmazza az előre és visszacsatolásokból számolt összeget is.
* : az aktivációs függvény szórása.
* : kiszámolja az aktivációs függvény értékét a neuronokban számolt összegekre, szórással.
* : a neuronháló kimenete.

A kimenetek számítására ugyanez az algoritmus van alkalmazva majd a tanítási algoritmusban is

### **7.1.3 A háló tanítása:**

A háló a kiterítés módszerével van tanítva. Ez azt jelenti, hogy kiszámítjuk a háló kimenetét több, időben eltolt bemenetre, majd időben kiterítjük a hálót és mindenik hibájával tanítjuk. Itt is több megközelítés létezik. Például, ha tanítjuk egy mintavételt, utána pedig visszaterítjük több iterációra, vagy ha többször tanítjuk mielőtt elkezdjük kiteríteni. Az itt használt algoritmus minden iterációban kiteríti a hálót, miután a hálónak számoltunk annyi kimenetet, hogy vissza lehessen teríteni. A kiterítés az [21. ábrán](#ábra21) tekinthető meg.



21. Ábra Kibővített visszacsatolt preceptron háló időbeli kiterítése

A kiterítés legnagyobb hibái az, hogy ahhoz, hogy több réteg esetén minden neuront tanítsak, előbb ki kell terítenem legalább annyiszor ahány rétegem van. A másik két nagy probléma, a gradiens felrobbanás és eltűnés, de ezek csak nagy kitérítés szám vagy nagyon sok réteg esetén jelentkeznek.

Az így kiterített hálót már taníthatom az MLP neuronhálóknál is alkalmazott backpropagation módszerrel. Itt fontos megjegyezni, hogy a hibát nemfeltétlen kell a háló végétől az elejéig visszaterjeszteni, hiszen ahol a kimenetek vannak, ott kiszámíthatom az elvárt és a kapott kimenetből. Ez segíthet abban, hogy a hiba pontossága ne romoljon, viszont rontom a háló memóriáját, ugyanis így csak a bensőbb rétegekbe propagálódik a hiba időben nagyobb mértékben. Viszont ez adhat egyfajta strukturáltságot a hálónak ugyanis a kimeneti réteg memóriája rövid, de ahogy haladunk a bensőbb rétegek fele az időben távolabb eső értékek hatása egyre nagyobb, így a memória is hosszabbtávú.

A tanításnál használt algoritmus a következő:

1. Minden o: 1->MaxIter
3. // Kimenet számítása:
4. Minden i: 1->NL
6. Vége minden
7. Minden i: 1->DL
8. //sorvektor, oszlopvektor
9. Minden j: 2->NL+1
10. // Az egy a biasok miatt kerül oda
11. Vége minden
12. Minden j: 1->NL
13. // oszlopvektor
15. Minden k: j->NL
17. Vége minden
19. Vége minden
20. Minden j: 1->NL

23. Vége minden
24. // oszlopvektor
25. //oszlopvektor, sorvektor
27. //Tanítás:
28. Ha i > R \* NL //Számítottunk e már elég kimenetet a visszaterítéshez
29. Minden j: 0->R-1

32. Ha Index < 1

35. Vége ha
37. Ha j > 0
38. ⊗

41. Vége ha
42. Minden k: NL-> (-1) ->Index
44. Minden g: NL-> (-1) ->k
46. Vége minden
47. Ha k > Index
48. ()
50. Ha j > 0
51. Minden g: Index+1->k-1
52. ()
53. Vége minden
54. Vége ha
56. Vége ha
57. Vége minden
58. Vége minden
59. Vége ha
60. Vége minden
61. Vége minden

Jelölések és magyarázat:

* MaxIter: a háló tanítását ennyiszer végzem el a teljes tanítóhalmazra.
* : a a háló kimeneteinek a négyzetes hibáinak az összege az o. tanítási ciklusban
* : az első iterációban az i. réteg kimenete, kezdetben az értékét nullára állítjuk.
* : létrehoz egy nullmátrixot, aminek n sora és m oszlopa van.
* : a neuronok kimeneteinek kezdeti értéke.
* DL: a tanítóhalmazban levő elemek száma.
* : oszlopvektor, a háló j. rétegének a neuronjainak a bemeneteit tartalmazza az i. bemenetre
* sorvektor, a neuronháló bemenete az i. bemenetre.
* : oszlopvektor, az első eleme egyes, utána a háló neuronjainak a kimeneteit tartalmazza az i. bemenetre, a j-1. rétegből
* : oszlopvektor, a háló j. rétegében levő neuronokban számolt előrecsatolásokból származó összegeket tartalmazza a j. iterációból
* : oszlopvektor, a háló j. rétegében levő neuronokban számolt visszacsatolásokból származó összegeket tartalmazza a j. iterációból
* : megmondja, hogy -nek hány sora és oszlopa van.
* : megmondja, hogy -nek hány eleme van.
* : -nek az első elem kivételével minden eleme (az első elem azért marad ki, mert az egy bias, nem egy neuron kimenete), az i. bemenetre.
* : a j. rétegben levő neuronokban számolt összeg, benne van az előre és visszacsatolásokból számolt összeg is.
* : az aktivációs függvény szórása.
* : kiszámolja az aktivációs függvény értékét a neuronokban számolt összegekre, szórással.
* : a j. rétegben levő neuronokban számolt összeg az i. bemenetre
* : a neuronháló kimenete.
* : a háló utolsó rétegében számolt hiba a neuronok kimenetén az i. bemenetre
* a háló elvárt válasza az i. bemenetre
* ennyiszer terítődik ki időben a háló.
* : segéd változók, mivel a hiba mindig csak egy réteget tud visszapropagálódni minden kiterítési fázisban, ezért ezek tartják számon, hogy hány réteg neuront lehet tanítani az egyes kiterítési időpillanatokban. Csak azokat lehet tanítani, ahova a hiba már visszapropagálódott. A index1 legkisebb értéke egy, ami azt jelenti, hogy az első réteget is taníthatom.
* : a k. rétegbe visszapropagált teljes hiba az i-j. bemenetre.
* : a k. rétegbe a visszacsatolásokon keresztül visszapropagált hiba az i-j. bemenetre.
* ⊗: Hadamard szorzat.
* : kiszámolja az aktivációs függvény deriváltjának értékét a k. rétegben levő neuronokban számolt összegekre, az i-j. bemenetre, szórással.
* : az előrecsatoló súlyzók tanításánál használt lépéshossz
* : az visszacsatoló súlyzók tanításánál használt lépéshossz

A tanítás kezdetén indul egy ciklus, ami MaxIter-szer fogja elvégezni a tanítást.

Először az tömb aktuális elemét állítjuk, ebbe a tömbbe vannak eltárolva az o. tanítási ciklusban létrejött hibák négyzetes értékeinek összege (o lesz a ciklusváltozó).

Ezután a háló minden rétegének kimenetét lenullázuk, hiszen az első bemenet számításánál ezek még nem ismertek, majd indul egy ciklus, aminek a változója i, és ez annyiszor fog végrehajtódni, ahány bemeneti minta van. Ez tartalmazza a kimenetszámítást és a tanítást is.

Ezt követően a kimenetszámító kódrészlet következik. Ez ki fogja számolni a kimenetet annyi alkalommal, ahányszor ki kell teríteni a hálót, hogy meg legyenek a kimenetek, amikből a hiba értéke megkapható.

Minden számítási ciklus végén mentődik a kimenet az mátrixba, melyben egy oszlop, egy számítási ciklus alatt számolt kimeneteket fogja tartalmazni. Ez a háló végső kimenete.

Emellett az -hoz adódik hozzá az adott számítási ciklus végén fellépő hiba négyzetes értékét.

Itt létrejön még egy nevű cella tömb. Ez nagyon fontos lesz, hiszen ez fogja tárolni a hiba értékét minden bemenetre történő számítás esetén, ebből lehet a visszaterjesztett hibákat meghatározni, és majd azok is ebbe a cella tömbbe mentődnek.

az i. bemenetre számolt hiba a j. rétegben. ( az i. ciklusban fellépő hiba az utolsó rétegben, ez az, mint elvárt kimenet - számított kimenet).

A hibaszámítás:

(Ahogy fentebb is említve van, az elvárt kimenet az i. bemenetre)

Ezt követi a tanítási rész.

Első lépésben el kell dönteni, hogy van-e legalább annyi kimenet számolva ahányszor ki kell teríteni a hálót, szorozva a réteg számmal.

Ezt követően egy ciklus indul, ami annyiszor fog lefutni, ahányszor kiterítjük a hálót. A ciklus változó j lesz, és ez 0-tól lépked R-1-ig (R a kiterítések száma).

Kiterített szekvenciának lesz a neve hálónak, a bemeneti rétegétől a kimenetiig. Egy kiterített szekvencia a teljes hálót fogja jelenteni a kiterítés során egy időszeletben. Például az [21. ábrán](#ábra21) 3 kiterített szekvencia látható.

Ebben a ciklusban először létrejön két segédváltozó (Index, Index2) amik segítenek nyomon követni, hogy a hiba hányadik rétegig propagálódott vissza egy adott kiterítési időpillanatban. Amennyiben Index értéke egy, akkor a hiba visszajutott a bemeneti rétegig.

Utána kimentődik a hiba az aktuális kiterített szekvencia utolsó rétegéből. Az első hibakimentés az utolsó kiterítési szekvenciánál történik, a legutolsó számolt kimenetnél, és onnan megyünk visszafele.

Az cellatömbbe mentődnek majd el visszapropagált hibák is.

azt jelenti, hogy az adott kimenetszámításhoz képest (i) visszamenve időben j vel, és onnan a háló utolsó rétege.

Itt abban az esetben, ha már egyszer kiterítettem, akkor a visszacsatolásokból érkező hibát is hozzáadom a kimeneten levő hibához. Így segítve azt, hogy a kimeneti réteg is rendelkezzen memóriával, de az adott kimeneten levő hiba is számításbavevődjön.

Ezt követően indul egy másik ciklus, ami annyiszor hajtóidk végre ahány rétege van a hálónak mínusz a fentebb leírt index, és az adott kiterítési szekvencián haladunk végig visszafele. Ennek a ciklusváltozója k lesz, és NL-től (rétegek száma, utolsó réteg indexe) megy vissza egyig.

Ebben először az adott réteg hibájára tanítjuk a réteg előre csatolt súlyzóit, a következő összefüggéssel:

ahol a k. rétegben levő előre csatoló súlyzómátrix, az előre csatoló súlyzók tanításánál használt lépéshossz, a k. réteg bemenete az i-j. iterációban, az aktivációs függvény deriváltja, a k. rétegben, az i-j. iterációban számolt összeg a neuronok bemenetén, oszlopvektor, a az aktivációs függvény szórása, pedig a k. réteg i-j. iterációban levő hibája.

Az is fontos, hogy a mátrixok ezen módon történő szorzása (a transzponálásokat tekintve) csak akkor aktuális, ha ilyen módon vannak létrehozva a mátrixok.

Ezt követően egy újabb ciklus indul, de ebben csak a visszacsatolási súlyzók tanítása található meg.

A ciklusváltozó g, és az utolsó réteg indexétől jön visszafele az aktuálisan tanított rétegig. Ezen rétegekből érkeznek be visszacsatolások az éppen tanított rétegbe. A tanítás a következő összefüggés alapján történik:

az a második elemétől, az utolsóig, első elemét nem kell venni, mivel az egy egyes, ami a bias szerepét látja el, de az nem csatolódik vissza.

azok a súlyzók, amik a g-ből csatolnak vissza k rétegbe, a visszacsatoló súlyzók tanításánál használt lépéshossz.

A tanítást követően el kell dönteni, hogy lehet e visszapropagálni hibát. Ez attól függ, hogy hányadszor terítődik ki a háló, hogy a hiba eljutott e már a rétegbe, ahonnan vissza kell propagálódjon.

Ha ez a feltétel teljesült akkor kiszámítom az előző rétegre (k-1. réteg) visszapropagált hibát, az előre csatolásokon keresztül, a következő összefüggéssel:

()

Itt abban különbözik a normál előre csatoló mátrixoktól, hogy le van vágva az első oszlopa, amiben az egyesek lennének a biasok miatt. Azokon keresztül nem áramlik vissza hiba.

Itt a hiba az i-j+1. kimenet számítási ciklus k. rétegéből propagálódik vissza a k-1.-be.

Ezután ellenőrzésre kerül, hogy nem e tart a tanítás az első kiterítési szekvenciában, hiszen a legutóbb számolt kimenetre nincs honnan visszaterjedjen hiba.

Itt indul egy újabb ciklus, amiben kiszámítódik a visszacsatolásokon keresztül visszapropagált hiba, a következő összefüggéssel:

()

Legvégül, az előrecsatolásokból számolt hibához hozzáadódik a visszacsatolásokból számolt hiba, így megkapjuk a teljes visszapropagált hibát, és kezdődhet a következő tanítási ciklus:

### **7.1.4 Mérések és elért eredmények:**

A dolgozat során a Matlab alatt létrehozott hálók építésére és tesztelésére létrehozott környezet használata rendkívül egyszerű.

Ahogy az [22. ábra](#ábra22) is mutatja, először be kell tölteni a Simulinkben vagy máshol elvégzett mérési eredményeket. Ezt követően egy függvény (ReadScopeData) kiolvassa az eredményeket a betöltött állományból. A kiolvasott adatokat egy második függvény formázza (GetReadyToUse). Ez a függvény az egyeseket is beteszi a bemenetek közé a biasok miatt.

Ezt követően el kell végezni a paraméterezést, meg kell adni, hogy hány bemenet és kimenet van, hány rétegű legyen a háló és a rétegekben hány neuron legyen, valamint hogy a neuronok kezdeti értéke mennyi legyen.

Meg kell adni, hogy mekkora legyen az aktivációs függvény szórása, mekkorák legyenek a tanítás során használt lépéshosszok, hány ciklust tanuljon, és hányszor legyen időben kiterítve a háló.

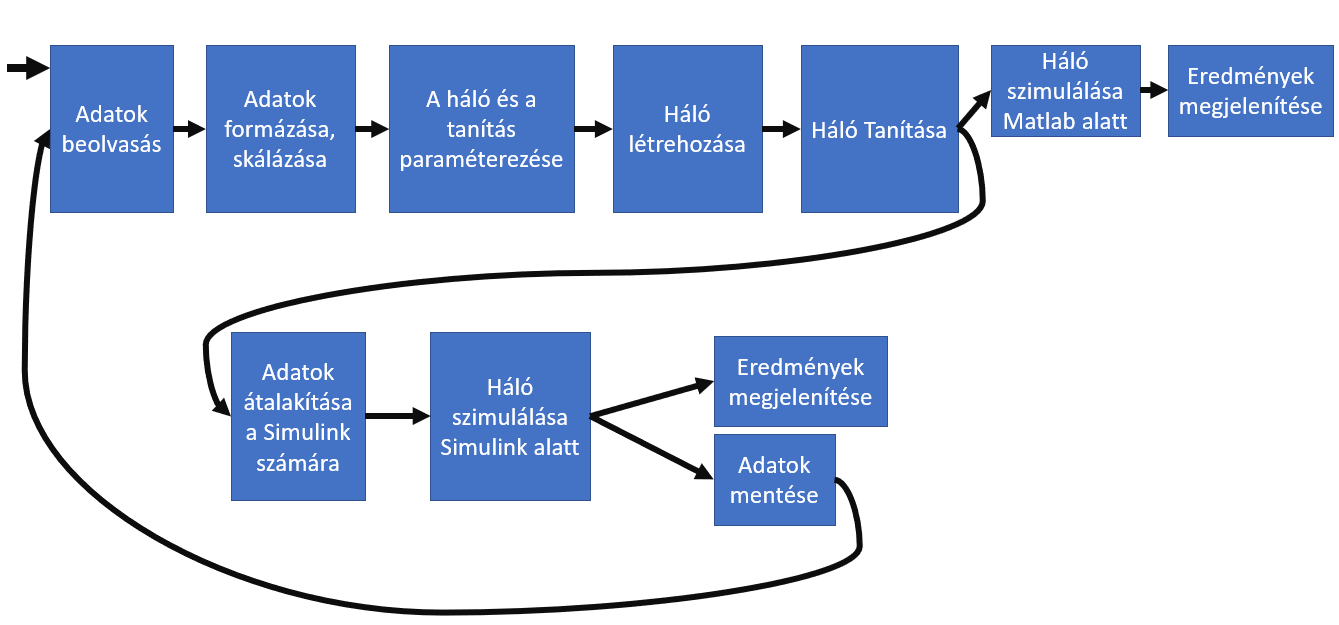
Ezután egy függvény (CreateNN) legenerálja a hálót, egy másik (TeachNN) tanítja és egy harmadik szimulálja (SimNN).

Amennyiben Simulinkben szimulálni szeretnénk a hálót egy függvény (CellToArray) átkonvertálja egy vektor formájába, amit majd a létrehozott Simulink függvény értelmez, és újra hálót épít belőle.

Végül a beépített Matlab függvények kirajzolják a tanítás eredményeit.

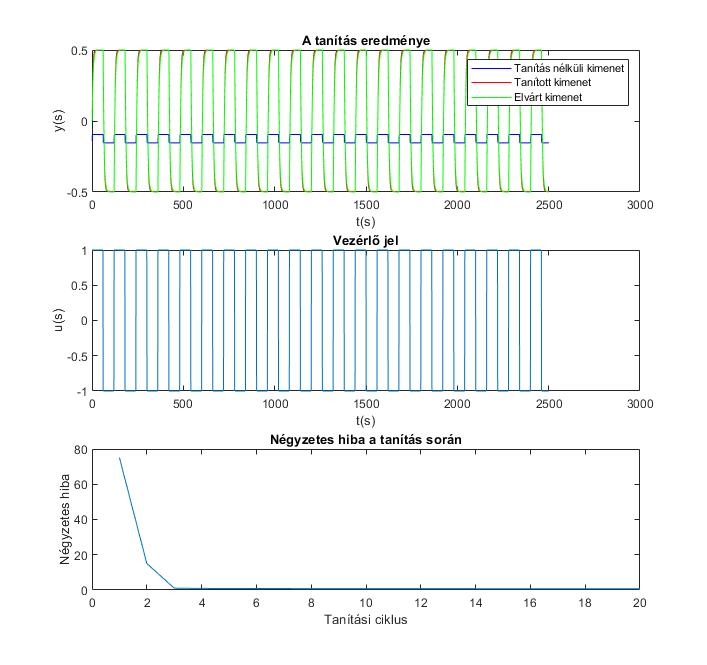
A Simulink-beli szimulációhoz annyi szükséges, hogy bekössük a dolgozat során létrehozott Neuronhálót beolvasó és futtató blokkot, ami az adatok skálázását is elvégzi. Amennyiben A Matlab-ban létrehoztuk a hálót akkor Simulink-ben már tesztelhető is.

A Simulink a Matlab munkalapjára kimenti a szimulált adatokat, onnan elmenthetőek és beolvashatóak, hogy azokból ismét taníthassuk a hálót és kezdhessük a folyamatot elölről.



22. Ábra Háló használatára folyamatábra

#### **7.1.4.1 Elsőfokú rendszer átvitelifüggvénye:**



23. Ábra 1. tanítás eredmény: elsőfokú rendszer átvitelifüggvény

A hálót tanulási képességét először egy elsőfokú rendszer átvitelifüggvényének a megtanulására volt tesztelve.

A tanult rendszer a következő:

Kezdetben a négyszögjel bemenetre 2500 mintavétel volt rögzítve, egy másodperces mintavételezési periódussal.

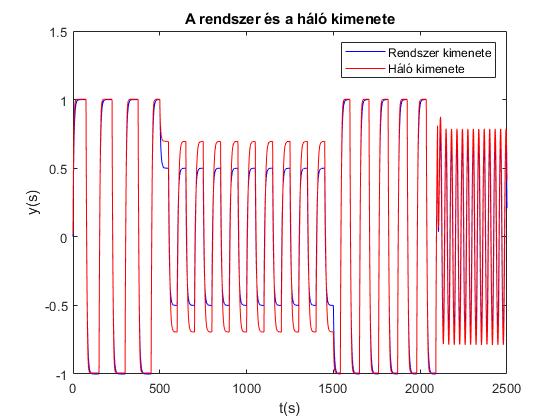
A neuronhálónak két rétege volt, két neuronnal az első és egy neuronnal a második rétegben.

A tanításnál az előrecsatolások tanításának lépeshossza és a visszacsatolások tanításának lépéshossza . Az aktivációs függvény tangens hiperbolikus és a szórása . A hálót időbeni kiterítéseinek a száma R=4 és MaxIter=20 alkalommal volt tanítva.

A tanulás eredménye az [23. ábrán](#ábra23) látható.

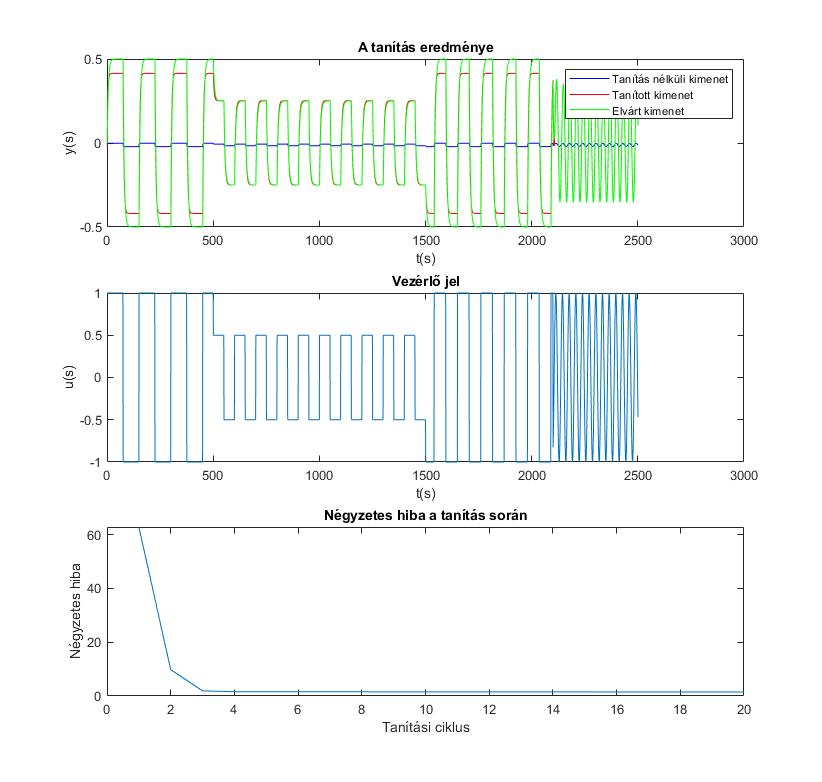
Az ábrán jól látható, hogy a háló nagyon jól megtanulja a rendszer szimulálását a tanítóhalmazra. A tanulás utolsó ciklusában a négyzetes hiba a 2500 mintára összesen mindössze 0.7782.

Azonban a Simulink szimuláció egy más bemenő jelre már rosszabb eredményeket szolgáltatott, mint ahogy az a [24. ábrán](#ábra24) is látható. A háló bár képesvolt megtanulni a jellegét a változásnak, de sajnos a kimenet amplitúdója határozottan rossz.



24. Ábra Háló kimenete: elsőfokú rendszer átvitelifüggvénye

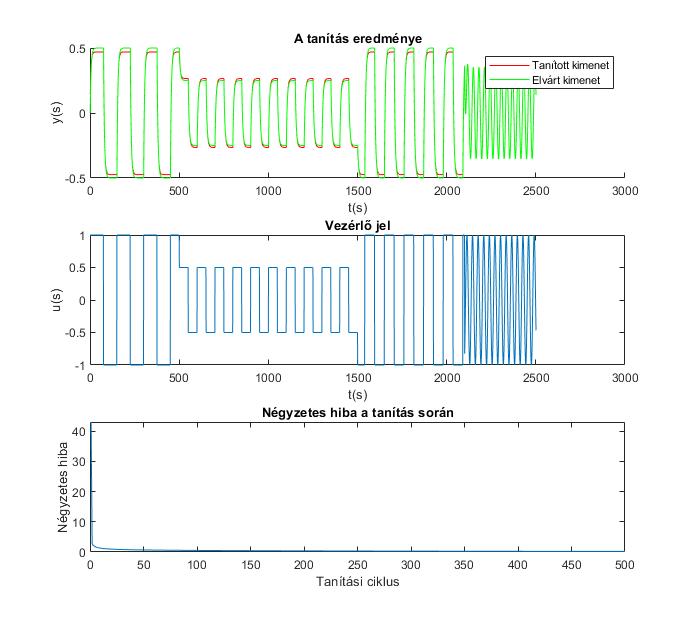
Mivel a rendszer a tanítóhalmazt kiválóan megtanulta, ezért valószínűsíthetően nem a háló paraméterekkel kellett, hogy legyen az elsődleges probléma, hanem a tanítóhalmazzal. Nem volt elég változatos. Az [25. ábrán](#ábra25) a háló ugyanazon paraméterekkel tanult változatosabb tanítóhalmazra.



25. Ábra 2. tanítás eredménye: elsőfokú rendszer átvitelifüggvénye

Mint az ábra is mutatja, a háló nem volt képes megfelelőmódon megtanulni a rendszert, tehát a paramétereken is változtatni kell.

A tanuláson végül az segített, hogy megnöveltem a neuronok számát tízre az első rétegben, az aktivációs függvény szórását egyre, és ötszáz iterációt tanítottam a hálót. Az így végrehajtott tanítás eredménye az [26. ábrán](#ábra26) látható.



26. Ábra 3. tanítás eredménye: elsőfokú rendszer átvitelifüggvénye

#### **7.1.4.2 Fékezett inga:**

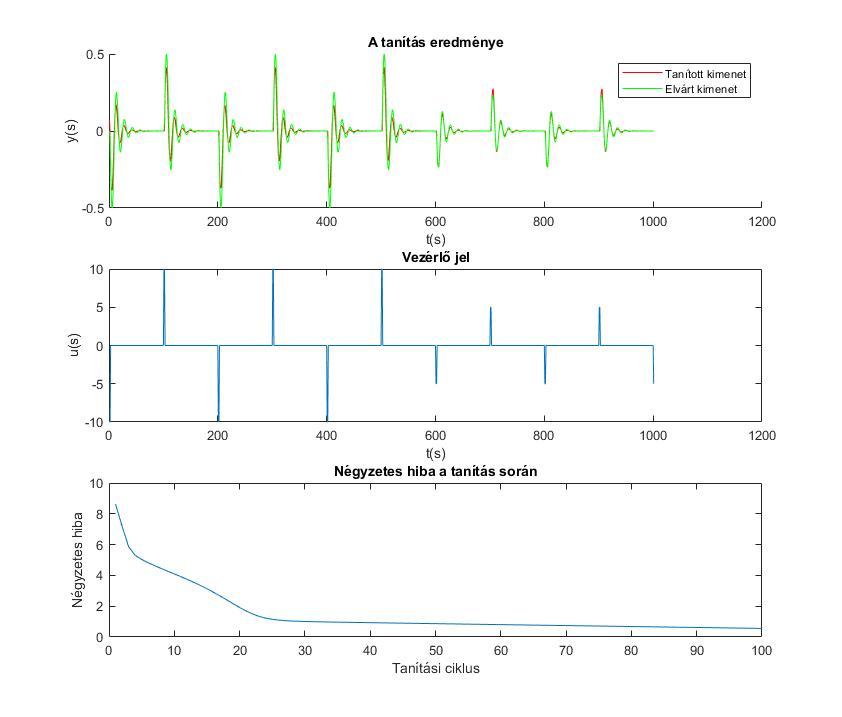
Az elsőfokú rendszert követően fontos volt letesztelni, hogy a háló képes e nemlineáris rendszerek megtanulására.

A nemlineáris rendszer egy inga ezesetben.

Az ingánál elért legjobb tanulás esetében egy háromrétegű neuronháló volt használva, húsz neuron a bemeneti rétegben, nyolc a rejtett rétegben és egy a kimeneti rétegben.

A tanításnál az előrecsatolások tanításának lépeshossza és a visszacsatolások tanításának lépéshossza . Az aktivációs függvény tangens hiperbolikus és a szórása . A hálót időbeni kiterítéseinek a száma R=4 és MaxIter=100 alkalommal volt tanítva.

A tanítás eredménye az [27. ábrán](#ábra27) látható. Ahogy az ábra is mutatja, a háló képes nemlineáris rendszerek viselkedésének a megtanulására.



27. Ábra 1. tanítás eredménye: fékezett inga

#### **7.1.4.3 Szinuszjel generátor:**

A fékezett inga után egy állandó amplitúdójú szinuszjel volt megtanítva a hálónak.

Itt a hálónak nincs bemenete, így azokat létre kell hozni valahogy. Akkor volt eredményes a tanítás, ha a háló kimenete időben késleltetve visszavezetőik a bemenetre. A kimenet háromszor volt késleltetve, egy, kettő majd három mintavétellel.

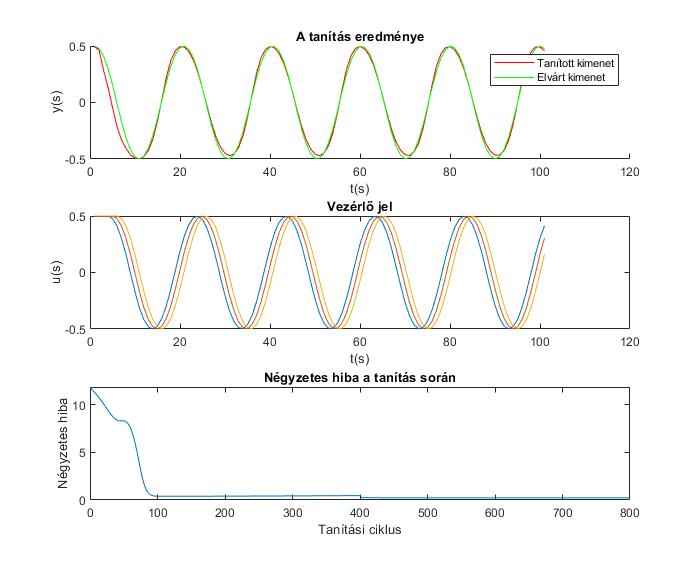
A tanítás során a háló bemenete egyszerűen a három, az elvárt kimenethez képest időben késleltetett szinuszjel értékei voltak. A tanítási algoritmus annyiban változott, hogy a kezdeti hibák elkerülése végett a háló állapota ki lett mentve. Így a háló szimulálása során egy stabil, jól meghatározott állapotból indul a háló, elkerülve a nemkívánt lengéseket a szimuláció kezdetén.

Az így megtanított hálóba kell majd visszavezetni a kimeneteit időben eltolva.

A hálónak három rétege van, a bemenetiben tíz neuron, a rejtett rétegben két neuron, és a kimenetiben egy neuron.

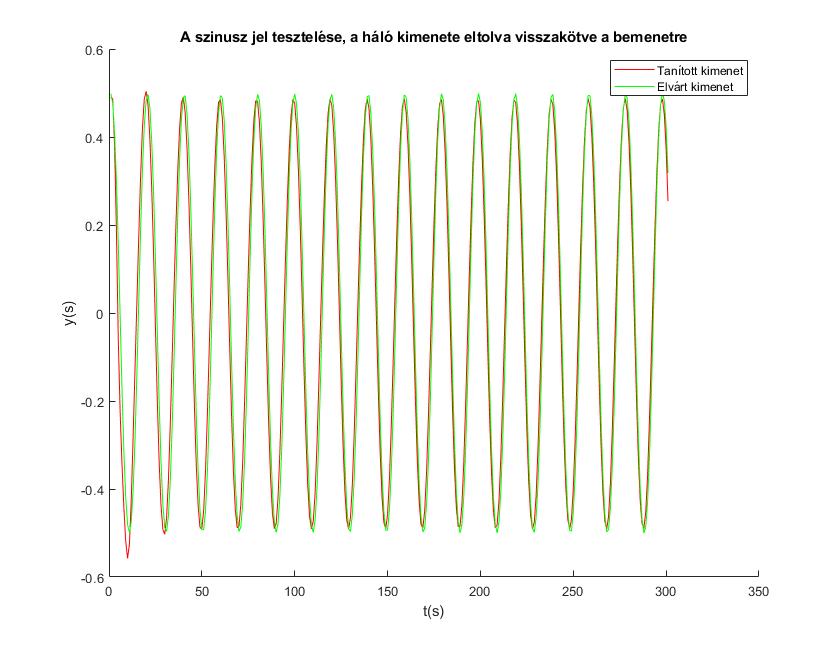
A tanításnál az előrecsatolások tanításának lépeshossza és a visszacsatolások tanításának lépéshossza . Az aktivációs függvény tangens hiperbolikus és a szórása . A hálót időbeni kiterítéseinek a száma R=4 és MaxIter=800 alkalommal volt tanítva.

A tanítás eredménye az [28. ábrán](#ábra28) látható.



28. Ábra 1. tanítás eredménye: szinuszjel generátor

Az így megtanított háló tesztelése az [29. ábrán](#ábra29) tekinthető meg.



29. Ábra Teszteredmény: szinuszjel generátor

#### **7.1.4.4 PI szabályzó:**

Az utolsó elvégzett mérés egy másodfokú rendszert irányító PI szabályzón történt. A háló a szabályozó viselkedését tanulta meg.

A másodfokú rendszer a következő:

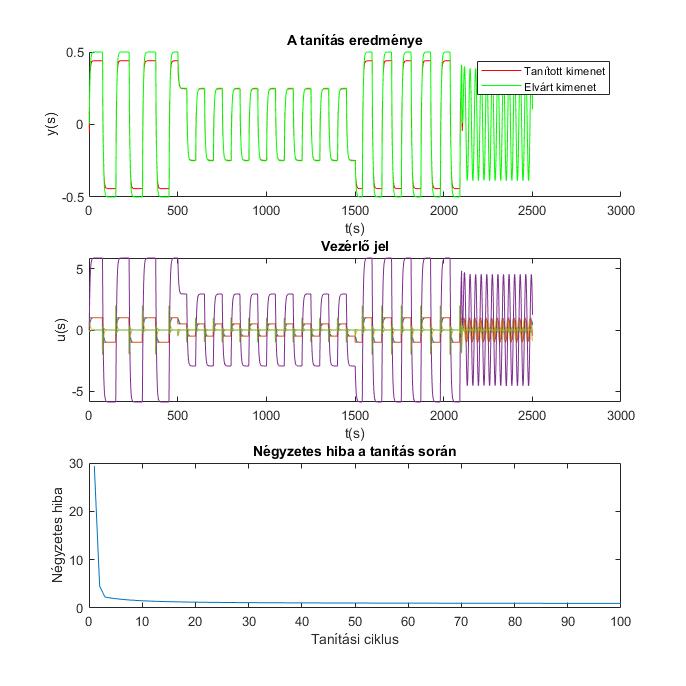
A PI szabályozó paraméterei: P= 0.00020349, I= 1673.72664341839.

Fontos megjegyezni a háló nem csak a hibát kapja meg, hanem a hiba integráltját és deriváltját, valamint a referenciát, és a rendszer kimenetét is.

A háló három rétegből áll, az első rétegben három, a másodikban kettő és a harmadik rétegben pedig 1 neuron található.

A tanításnál az előrecsatolások tanítási együtthatója és a visszacsatolások tanítási együttható . Az aktivációs függvény tangens hiperbolikus és a szórása . A hálót időbeni kiterítéseinek a száma R=4 és MaxIter=100 alkalommal volt tanítva a tanítóhalmazra.

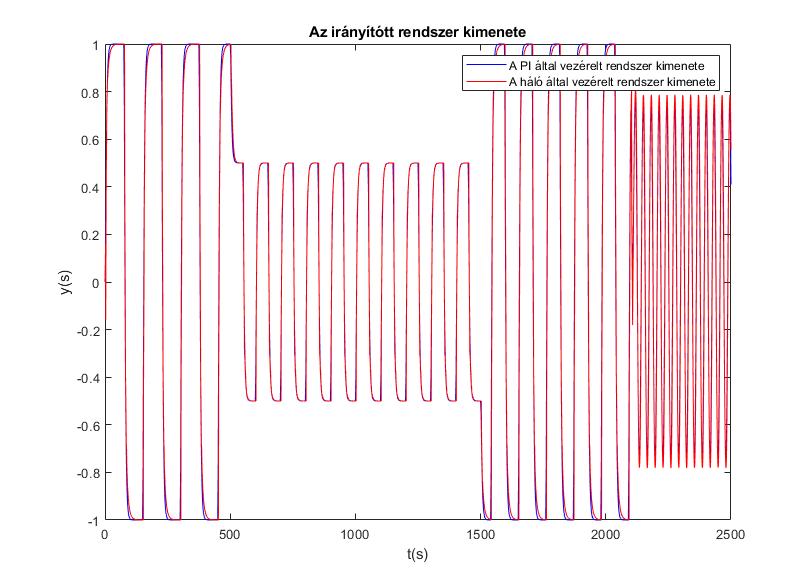
A tanítás eredménye az [32. ábrán](#ábra30) látható.



30. Ábra 1. tanítás eredménye: PI szabályzó

Itt jól látható, hogy a bemeneten levő ugrások még nem okoznak nagy ampitudójú tüskéket a kimeneten, amik a teljesen visszacsatolt neuronhálónál megfigyelhető. Mivel itt a szerkezet rétegeltebb, a benső rétegek kiszűrik a nagy ugrásokat.

Az így tanított hálóval irányított másodfokú rendszer Simulink szimulációja az [31. ábrán](#ábra31) tekinthető meg.



31. Ábra Teszteredmény: PI szabályzó, irányított rendszer kimenete

Az ábra jól mutatja, hogy a háló kiválóan megtanulta a PI viselkedését. Úgy sem romlott le a szabályzás minősége jelentős mértékben, hogy a hálónak három mintavételre van szüksége a három réteg miatt, hogy a bemeneten levő változás megjelenjen a kimeneten, míg a PI esetén ehhez mindössze egy mintavétel szükséges. Gyorsabb szabályzásoknál ez viszont problémát okozhat.

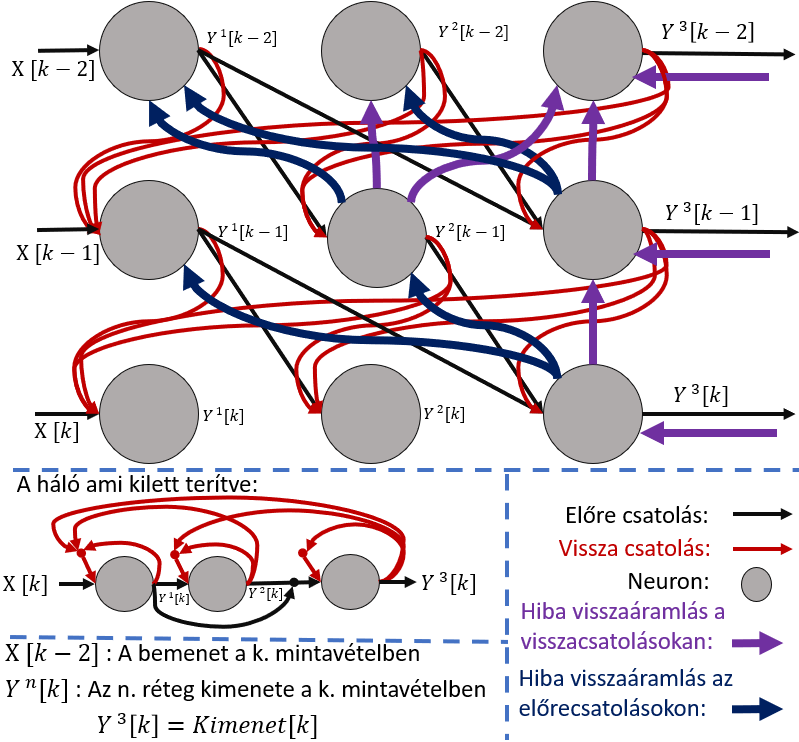
## **7.2 Egyszerűsített teljesen visszacsatolt neuronháló:**

### **7.2.1 A háló létrehozása:**

Az egyszerűsített teljesen visszacsatolt neuronháló az [19. ábrán](#ábra19) tekinthető meg. Nagyon kis mértékben tér el a kibővített visszacsatolt preceptron hálótól. A szerkezete fentebb ismertetve volt.

A kiterítés kis mértékben módosul, ahogy az [32.ábrán](#ábra32) látható.

Az ábrán jól látható, hogy az első rétegekbe már az első visszaáramoltatás után visszajutott a hiba, bár ez a hiba egyelőre csak az előrecsatolásokon keresztül jutott oda. A visszacsatolásokon ugyanúgy több kiterítés szükséges.



32. Ábra Egyszerűsített teljesen visszacsatolt neuronháló időbeli kiterítése

Az algoritmus csak minimális mértékben kell módosuljon ennek a hálónak a megvalósításához. Az előrecsatolásokon tartalmazó itt már csak a bemenetek bekötéséért felel a bemeneti rétegbe, és a többi előrecsatolás súlyzója is a -ben kap helyet.

Így a kimeneteket számító algoritmus a következő lesz:

1. Minden i: 1->NL
3. Vége minden
4. Minden i: 1->DL
5. //sorvektor, oszlopvektor
6. Minden j: 2->NL+1
7. // Az egy a biasok miatt kerül oda
8. Vége minden
9. Minden j: 1->NL
10. Ha j=1
11. // oszlopvektor
12. Máskülönben
14. Vége ha
16. Minden k: 1->NL
18. Vége minden
20. Vége minden
21. Minden j: 1->NL
23. Vége minden
24. // oszlopvektor
25. Vége minden

Az algoritmus csak a 10,11,12,13 sorokban módosult, valamint a 16. sorban a ciklust egytől kezdődik (az első rétegtől kezdve számolok az eddig csak a visszacsatolásokat tartalmazó - vel).

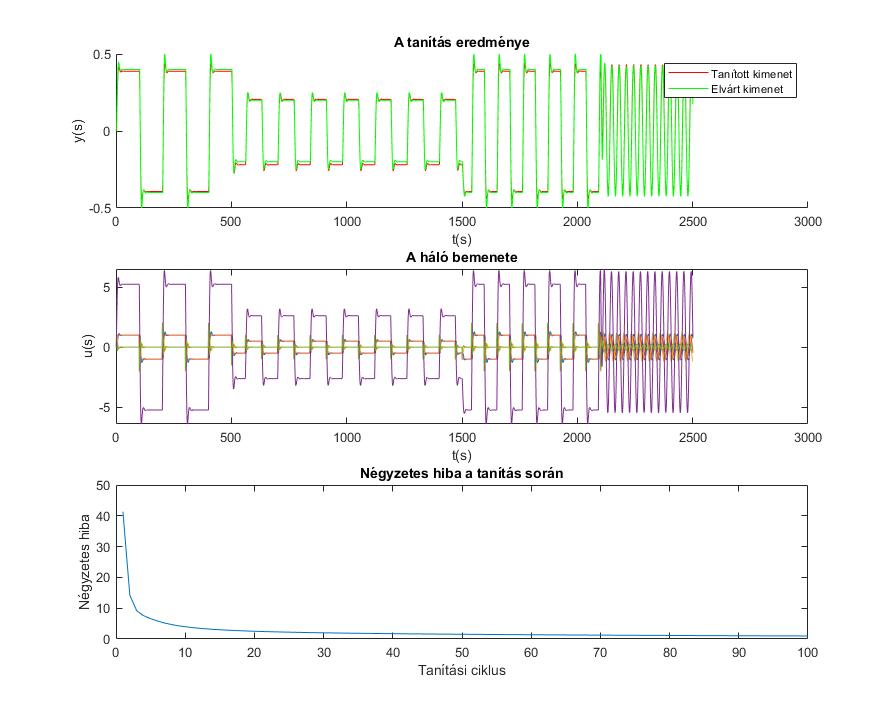
A tanítás az első módszerhez hasonlóan időben való kiterítéssel és gradiens módszerrel, hiba visszaáramoltatással történik.

A tanítási algoritmus is ugyanezen helyeken módosul, valamint már az első visszaáramoltatás során számolódik hiba minden rétegben, az előrecsatolásokon keresztül.

### **7.2.2 Mérések, elért eredmények:**

A létrehozott környezet használata ugyanúgy történik, mint a kibővített visszacsatolt preceptron háló esetén.

#### **7.2.2.1 PI szabályzó:**



33. Ábra Tanítás eredménye: PI szabályzó

Az előző mérések során is használt PI szabályzó viselkedését tanulta a háló. Itt azonban a PI kimenete és a rendszer bemenete közé volt téve egy három mintavételes eltolás (A mintavételi periódus itt is egy másodperc), hogy imitálva legyen a háló rétegek miatti késleltetése.

Az itt használt PI paraméterei: P= 0.300434374912646, I= 0.579433661057263.

A vezérelt rendszer itt a következő:

A háló ugyanúgy megkapja a rendszerkimenet és a referencia közti hibát, ennek e deriváltját és az integrálját, valamint a referenciát és a rendszer kimenetét is.

A háló háromrétegű háló volt, öt neuronnal az első rétegben, három neuronnal a másodikban és egy neuronnal az utolsóban.

A tanításnál az előrecsatolások tanításának lépeshossza és a visszacsatolások tanításának lépéshossza . Az aktivációs függvény tangens hiperbolikus és a szórása . A hálót időbeni kiterítéseinek a száma R=4 és MaxIter=100 alkalommal volt tanítva a tanítóhalmazra.

A tanítás eredménye az [33. ábrán](#ábra33) látható.

## **7.3 Teljesen visszacsatolt neuronháló:**

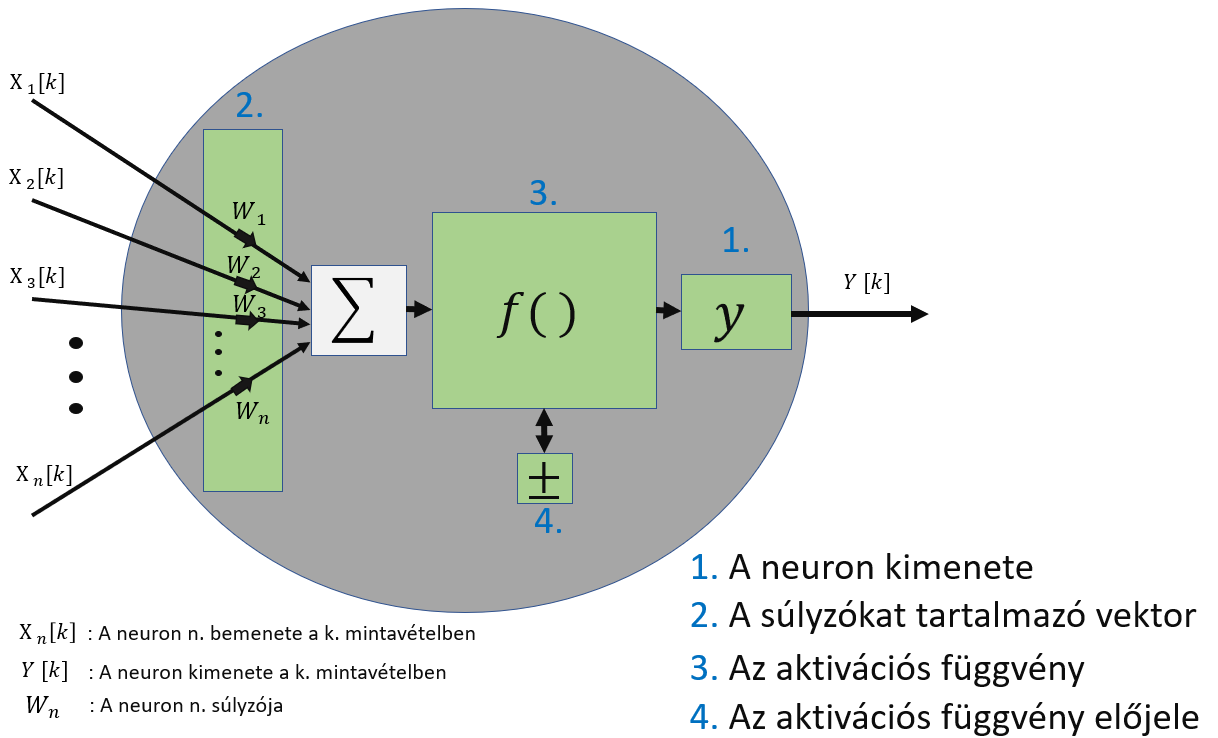
### **7.3.1 A háló létrehozása:**

A teljesen visszacsatolt neuronháló már csak egy lépésre van az egyszerűsített teljesen visszacsatolt neuronhálól. Annyi változik benne, hogy itt minden réteget köt a bemenetekre, nem csak az első réteget. Ez ismét csak minimális változásokat jelent az algoritmusban és a hiba visszaterjesztésben.

Azonban itt létre van hozva egy másik ábrázolásmód, amiatt, hogy ezt a struktúrát már genetikus algoritmussal építi és kezdi el a tanítását.

Ott minden neuron egy struktúra, a saját súlyzóival, és aktivációs függvényével, valamint a neuronban van lehetőség az aktivációs függvény előjelét is módosítani, hogy a mutáció során létrejött változások minél változatosabbak lehessenek. Amiatt, hogy kompatibilis legyen az előző ábrázolással, az esetek többségében a háló csak egy aktivációs függvényt tartalmaz, aminek az előjele nem változhat. Egy neuron felépítése az [34. ábrán](#ábra34) látható.

Itt mivel teljesen visszacsatolt hálóról van szó a neuronok bemenetként megkapják a háló összes bemenetét, valamint a hálóban levő összes neuron kimenetét is.



34. Ábra Teljesen visszacsatolt neuronháló második neuronstruktúra

Az így létrehozott neuronok egy cella tömbben tárolódnak el. A háló kimenete úgy számolódik, hogy ahány kimenetet adunk meg a hálónak, a neuronokat tartalmazó cellatömb elejéről annyi neuron lesz kimenet. Ez megadja a háló minimális méretét, legalább annyi neuron kell legyen a hálóban, ahány kimenete van a hálónak.

### **7.3.2 A genetikus algoritmus megvalósítása:**

Az megvalósított algoritmus úgy dolgozik, hogy először legenerál egy n neuronhálóból álló populációt. Kezdetben a populáció minden neuronhálóját a minimális neuronszámmal hozza létre. Ezt meg lehet adni, vagy ha nem akkor annyival ahány kimenete van a hálónak.

Ezt követően minden háló kimenetét kiszámolja a tanítóhalmazra és minden hálónál számol egy összesített négyzetes hibát.

Az így számolt hiba alapján, a populáció azon felét, ami a legrosszabbul teljesített törli. Ezt követően a populáció megmaradt feléből újra felkell tölteni a populációt hálókkal.

A feltöltés úgy történik, hogy az algoritmus végig megy a megmaradt neuronhálókon, és páronként létrehoz belőlük egy utódod. Ez azt jeleni, hogy az első és a második hálóból létrejön egy utód, majd a másodikból és a harmadikból, aztán a harmadikból és a negyedikből, és így tovább.

Ezt követően ismét szimuláljuk a populációban levő neuronokat, és a ciklus kezdődik elölről.

Az algoritmusba bele van építve hogyha n ciklusig nem javul a hiba értéke a populációban akkor a populációban a legjobb háló kivételével mindenki törlődik és új populáció generálódik a legjobb háló mellé.

### **7.3.3 Az utódok létrehozása:**

Az utódok létrehozása úgy történik, hogy az algoritmus végigmegy a szülők neuronjain, és minden neuron esetén 50% esély lesz, hogy az egyik és 50% esély, hogy a másik szülőtől öröklődik. Ha valamelyik szülőnek több neuronja van akkor minden fennmaradt neuronnak 50% esélye lesz, hogy az utód megkapja.

Ha létrejött az utód, utána a mutációk következnek.

### **7.3.4 A mutációk:**

A mutáció random valószínűségek alapján történik. Az algoritmus végigmegy a háló minden neuronján. Először van egy valószínűsége, hogy a neuron minden súlyzója újra generálódik, ha ez nem következett be, akkor egy másik valószínűség alapján megváltozhat egy adott magnitúdóval az az egyik random választott súlyzó.

Ezek mellett van egy valószínűség arra, hogy az aktivációs függvény vagy az előjele lecserélődik.

A két legfontosabb mutáció a két utolsó, ugyanis egy adott valószínűséggel a háló kaphat egy új neuront, vagy egy másik valószínűséggel elveszíthet egyet. Ez a két mutáció, ahogy az [35. ábra](#ábra35) is mutatja, csak egyszer hajtódhat végre egy neuronháló esetén, amíg az előzőek a hálóban levő minden neuronra végrehajtódhattak.

Fontos, hogy a háló itt nem kaphat kimeneti neuront, de elveszíthet egyet. Ha elveszíti akkor a hálóban eltárolt következő neuron lép a kimeneti neuron helyére. Csak akkor veszíthet neuront, ha legalább egyel több neuronja van, mint a háló kimeneteinek a száma.

A képen asztal látható

Automatikusan generált leírás

35. Ábra Mutáció

### **7.3.5 Mérések, elért eredmények:**

A létrehozott környezetben a háló építés és szimulálás kezdete ugyanúgy történik, mint az előző esetekben, beolvassuk majd formázzuk az adatokat. Ugyanúgy páraméterezni kell, de a kimenetek és a bemenetek száma mellett meg kell adni, hogy hány aktivációs függvény van, és hogy hány neuron legyen pluszba a minimális neuronszámon felül.

Ugyanúgy megadjuk az aktivációs függvények szórását, valamint, hogy mennyi legyen a neuronok kezdeti állapota.

Megadjuk, hogy hány generációnk legyen és mekkora populáció, valamint amennyiben utólag gradiens módszerrel is tanítjuk a hálót, hogy hány iterációt tanuljon ott és hányszoros kiterítéssel.

Ezt követően a későbbiekben részletezett valószínűségek paraméterezése következik.

Ha a paraméterezés megvan, akkor egy függvény (BuildAndTeach) elvégzi a háló építését és tanítását genetikus algoritmus alapján. Az így létrehozott hálót egy második függvény szimulálja (SimSBNN).

Ez a hálóábrázolás eltér attól, amit a létrehozott gradiens módszer tanítani tud, így amennyiben Gradiens módszerrel is tanítunk egy függvénnyel (GetReadyForBackpropagation) át kell alakítani mátrixos formába a hálót.

Innen már minden úgy történik, mint ahogy az előzőleg be lett mutatva, tanítjuk a gradiens módszerrel, ábrázoljuk, átalakítjuk, hogy a Simulink is értelmezni tudja, ott szimulálhatjuk és menthetjük a szimulált adatokat.

#### **7.3.5.1 PI szabályzó:**

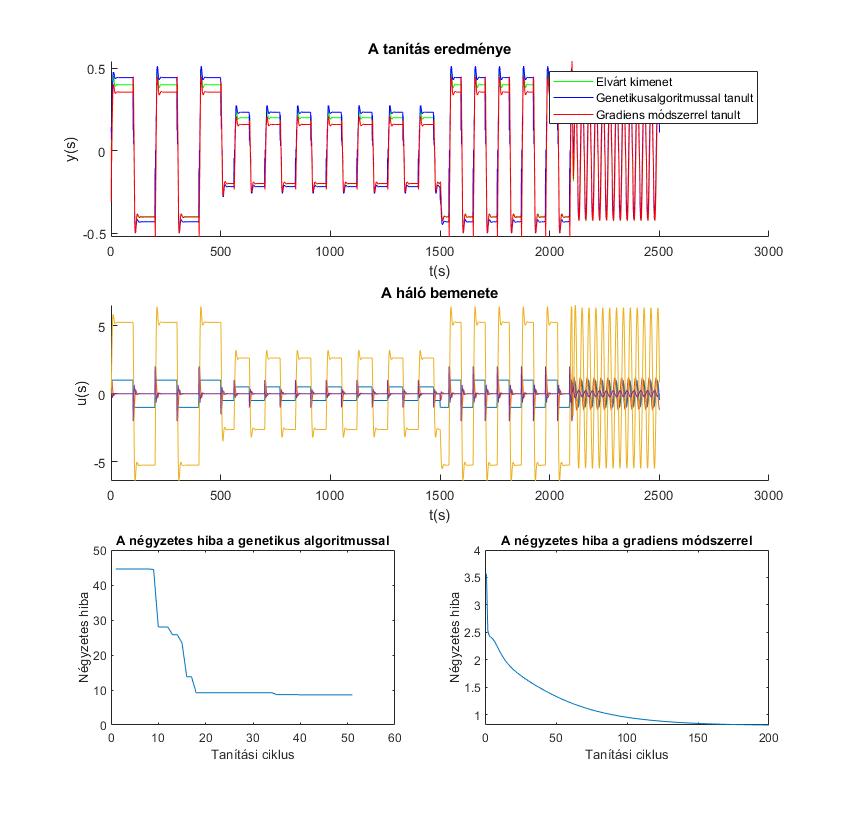
Ez a háló úgyszintén PI szabályzó viselkedésének a megtanulására volt tesztelve. A háló kezdetben mindössze egy neuront tartalmaz, onnan növekszik majd a tanítás során.

Az itt használt PI paraméterei úgyszintén: P= 0.300434374912646, I= 0.579433661057263 és a vezérelt rendszer itt átvitelifüggvénye a következő: .

A használt aktivációs függvény tangens hiperbolikus és a szórása . A gradiens módszerrel történő tanítás során pedig a tanítás paraméterei a következőek: az előrecsatolások tanításának lépeshossza , a visszacsatolások tanításának lépéshossza . A háló időbeni kiterítéseinek a száma R=2 és MaxIter=200 alkalommal volt tanítva a tanítóhalmazra.

* A genetikus algoritmussal való tanítás során a háló paraméterei a következőek:
* A mutáció valószínűsége 80%
* A teljes súlyzómátrix mutációjának a valószínűsége 50%
* Egyetlenegy súlyzó mutációjának a valószínűsége 80% és a mutáció maximális értéke egy
* Az aktivációs függvény és az előjele nem mutálódhat ezesetben
* Annak a valószínűsége, hogy új neuron jön be a hálóba 95%
* Egy neuron elvesztésének a valószínűsége 30%
* A populáció ötven neuronból áll és ötven generációt tanítok.

A tanítás eredménye az [36. ábrán](#ábra36) látható.

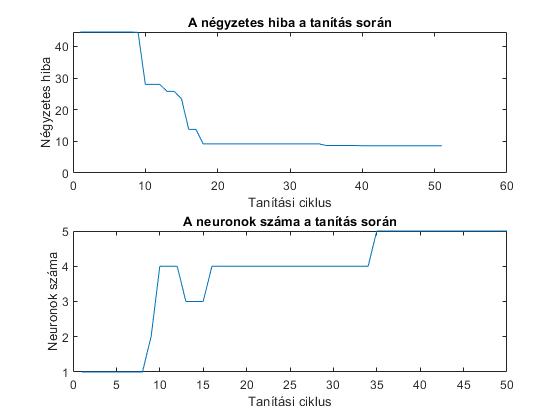


36. Ábra Tanítás eredménye: PI szabályozó

Ahogy az ábra is mutatja a tanítás mindkét szakaszban eredményes volt a maga módján. A genetikus algoritmus felépítette a hálót és a hibát lecsökkentette a hibát. Onnan a gradiens módszer átvette a tanítást és javított a háló működésén.

A háló kimenetén láthatóak a tüskék, amik említve voltak. Ezen tüskék is eltüntethetőek elegendő tanítással, de az előző hálóstruktúrákra sokkal kevésbé voltak jellemzőek a rétegeltség miatt.

Az [37. ábrán](#ábra37) látható ahogy a háló genetikus algoritmussal való tanítása során hogyan változott a hiba, valamint hogyan kerültek be a neuronok a hálóba. A legnagyobb ugrások a hibában mindig akkor láthatóak, amikor egy új neuron kerül be. Ahogy az ábra is mutatja, végül öt neuron alkotja a létrehozott hálót. Ebből egy a kimeneti neuron, a többi az első rétegben foglal helyet.



37. Ábra Hiba a neuronszám függvényében

## **8.1 Neuronháló megvalósítása FPGA lapon:**

A háló magvalósítása FPGA lapon egyelőre csak egy próbálkozásból áll. A Matlab lehetőséget biztosít egy megfelelő módon megírt Matlab függvény VHDL nyelvre való lefordítására.

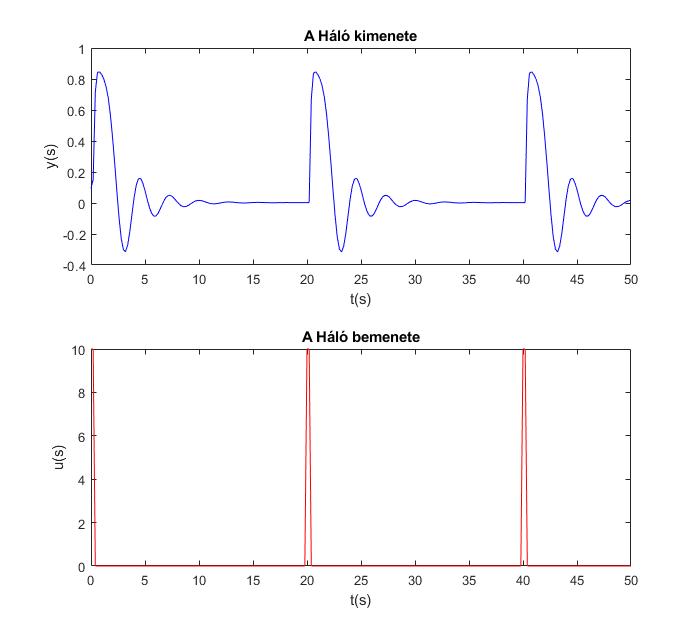
A kipróbált háló nagyon egyszerű, három rétege van, a bemeneti rétegben három neuron található, a rejtett rétegben kettő, a kimenetiben pedig egy. Az aktivációs függvény tangens hiperbolikus, 0.88 szórással.

A háló egy inga mozgását szimulálta, és így egy bemenete és egy kimenete volt. A bemenet az erő, amivel meglökjük az ingát, a kimenet pedig az inga mozgása.

Mivel itt nem a tanításon van a lényeg, [fentebb](#_7.1.4.2_Fékezett_inga:) egy ingára való tanítás már részletezve van, ezért azt itt nem részletezem.

A háló **kibővített visszacsatolt preceptron,** Matlab alatt tanul, és az így létrejött súlyzómátrixok bele vannak írva a Matlab függvénybe, amit a beépített HDL coder lefordít VHDL nyelvre.

A háló Simulink alatti szimulációjának az eredménye az [38.ábrán](#ábra38) látható.



38. Ábra VHDL-re fordítandó háló Simulink teszteredménye

Az ábrán látható, hogy a háló jól követi egy lengő inga mozgását.

Az itt megírt háló, már a fordításra előkészített algoritmus alapján fut. Az algoritmus nem tartalmaz ciklusokat, a háló működését teljesen szekvenciálisan kellett leírni és a súlyzók értékei is bele vannak írva a kódba.

A fordítás után a háló Vivado 2019.2 programban lett szintetizálva és szimulálva, de már, mint VHDL kód.

A lapka amire le lett fordítva a VHDL kód egy Artix-7 (xc7a100tcsg324-1). A kihasználtsága az [39. ábrán](#ábra39) látható.

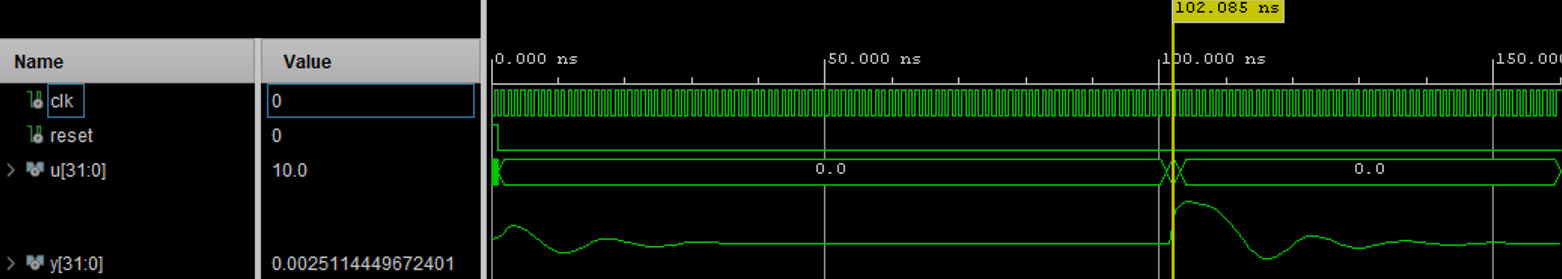
Sajnos az ábra mutatja, hogy a fordítás egyáltalán nem optimális, ezen még dolgozni kell, hiszen a mindössze 6 neuront tartalmazó háló erőforrásigénye óriási.

A képen asztal látható

Automatikusan generált leírás

39. Ábra Lapka kihasználtsága

A fordítás után a Vivado beépített szimulációs környezetével történő szimuláció eredménye az [40. ábrán](#ábra40) látható.



40. Ábra VHDL re fordított háló Vivado teszteredménye

A fenti ábrán az első lengés, ami 100ns után következik be (az első csak a háló stabilizálódása a szimulációjának kezdetén), maximális értéke 0.846647, míg a Simulink szimuláció alatt az első lengésnél történő maximális érték 0.846649 volt. Az eltérés a hatodik tizedesnél következik be, ez jól mutatja, hogy a két rendszer megközelítőleg identikus módon működik.

# **9. Következtetések:**

Bár a dolgozat még távol áll a célok elérésétől, ugyanis az újra konfigurálható digitális áramkörökön való futtatás még csak egy próbálozás volt, és a fordítás eredményeként létrejött háló erőforrás igénye jóval nagyobb volt, mint az várható volt így még a fordításon és annak paraméterezésén dolgozni kell, de az bizonyos, hogy megoldható.

A szimulációk eredményei ígéretesek, hiszen bebizonyosodott, hogy a dolgozatban létrehozott hálóstruktúra képes a nemlineáris rendszerek szimulációjára, és ha erre képes akkor képes kell, hogy legyen az irányításukra is. Jelen állás szerint a hálót anélkül, hogy egy létező szabályzó utánzására tanítanánk, a genetikus algoritmussal lehetne szabályzásra tanítani, úgy, hogy a mért a populáció tesztelése során nem állunk meg a háló szimulációjánál, hanem a hálókat rákötjük a vezérlendő rendszerre és a rendszer kimenetén számítjuk a hibát, ami alapján a populáció tagjainak rangsorolása történik. Ehhez még sokat kell finomítani az evolúciós algoritmust, hiszen az jelenleg még rendkívül lassan tanul, nagyon egyszerű rendszereket is.

A tanítás jelenlegi elképzelés szerint marad számítógépen, a nagy memóriaigény miatt, de amennyiben sikerül megfelelő módon optimalizálni a tanító algoritmusokat az is átültethető lenne újra konfigurálható digitális áramkörökre.

A gradiens módszer alapú tanítás mindig jóval gyorsabb lesz a genetikus algoritmusnál, így amennyiben rendelkezésünkre állnak mért adatok, még mindig hatékonyabb annak a használata. A hiba visszaáramoltatás és gradiens módszerrel történő tanítási módszer eredményesnek tűnik, és a tapasztalatok alapján robusztus. Ez alapján azt értem, hogy a hiba visszaterjsztő algoritmus megírása során rengeteg változat jött létre, amikben különböző logikai hibák voltak. Például a visszacsatolásokat rossz sorrendben, vagy nem megfelelő időszeletben tanítottuk, de a háló ugyanúgy képes volt a tanulásra.

Egyelőre a kettőnek a kombinációja tűnik a leg célravezetőbbnek általános esetekben, ugyanis a genetikus algoritmus képes a háló felépítésére, de a gradiens módszer nem, ezzel szemben a gradiens módszere messzemenően kevesebb idő alatt képes a hálót tanítani, mint a genetikus algoritmus. Így azzal, hogy csak a kezdeti „durva” tanítást bízzuk a genetikus algoritmusra és a háló építését, majd a finomhangolást, ami a legidőigényesebb a gradiens módszerre bízzuk, amennyiben ez lehetséges, mindkét módszer nyújtotta előnyöket ki lehet használni.

A tapasztalatok alapján úgy tűnik, hogy a rendszerek működésének tanításához nincs szükség nagyon mély visszaterjesztésre és hálóra, így az időfüggő neuronhálók két nagyobb problémája, a gradiens eltűnés és felrobbanás egyelőre nem jelentett problémát.

A gradiens módszerrel való tanítás során a tanított háló egyszersem vált instabillá, és a tanítás is csak nagyon kevés és szélsőségesen paraméterezett tanítások során vált instabillá.

A genetikus algoritmus során egyszer sem fordult elő instabilitás, hiszen a tanítás során az a háló, ami instabil állapotba kerül a nagy hiba miatt egyszerűen kikerül a rendszerből.

Ezek viszont nem jelentik azt, hogy az instabilitás nem okoz problémát, hiszen, ha kis mértékben is, de jelen volt a genetikus algoritmussal való tanítások során, és mivel a hálót vezérlésre is alkalmazni szeretnénk, bizonyosnak kell lenni a stabilitásában, így a hálók stabilitása további kutatást igényel.

A végkövetkeztetés az, hogy a dolgozat még távol áll a célok elérésétől, de az elért eredmények ígéretesek és mindenképpen abba az irányba mutatnak, hogy érdemes foglalkozni a témával.

# **10. Irodalomjegyzék:**

[1] YEGNANARAYANA, B. Artificial neural networks for pattern recognition. *Sadhana*, 1994, 19.2: 189-238.

[2] PEDRERO-SÁNCHEZ, José Francisco, et al. Classification of healthy, Alzheimer and Parkinson populations with a multi-branch neural network. *Biomedical Signal Processing and Control*, 2022, 75: 103617.

[3] UNCINI, Aurelio. Audio signal processing by neural networks. *Neurocomputing*, 2003, 55.3-4: 593-625.

[4] NGUYEN, V. T.; PASHCHENKO, F. F. Development of an object recognition algorithm based on neural networks With using a hierarchical classifier. *Procedia Computer Science*, 2021, 184: 438-444.

[5] YANG, Gang; DING, Fei. Associative memory optimized method on deep neural networks for image classification. *Information Sciences*, 2020, 533: 108-119.

[6] GONZALEZ, Jesús; YU, Wen. Non-linear system modeling using LSTM neural networks. *IFAC-PapersOnLine*, 2018, 51.13: 485-489.

[7] SHARMA, Sanjay K.; SUTTON, Robert. An optimised nonlinear model predictive control based autopilot for an uninhabited surface vehicle. *IFAC Proceedings Volumes*, 2013, 46.10: 73-78.

[8] REN, Guanghua, et al. A modified Elman neural network with a new learning rate scheme. *Neurocomputing*, 2018, 286: 11-18.

[9] <https://www.youtube.com/watch?v=fkIJwYArKpw>

[10] <https://en.wikipedia.org/wiki/Bidirectional_associative_memory>

[11] <https://www.youtube.com/watch?v=vlRwUV_sGcs>

[12] <http://www.scholarpedia.org/article/Echo_state_network>

[13] <https://en.wikipedia.org/wiki/Echo_state_network>

[14] <https://programmathically.com/understanding-the-exploding-and-vanishing-gradients-problem/>

[15] LI, Shuai, et al. Independently recurrent neural network (indrnn): Building a longer and deeper rnn. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018. p. 5457-5466.

[16] <https://en.wikipedia.org/wiki/Recursive_neural_network>

[17] <https://machinelearningmastery.com/recurrent-neural-network-algorithms-for-deep-learning/>

[18] SOLTANI, Rohollah; JIANG, Hui. Higher order recurrent neural networks. *arXiv preprint arXiv:1605.00064*, 2016.

[19] HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. Neural computation9 (8): 1735-1780. 1997.

[20] <http://dprogrammer.org/rnn-lstm-gru>

[21] CAMPO, Ángel; REYES, José Santos. Evolution of adaptive center-crossing continuous time recurrent neural networks for biped robot control. In: *ESANN*. 2010.

[22] TUTSCHKU, Kurt. Recurrent multilayer perceptrons for identification and control: The road to applications. *Univ. Würzburg, Germany, ser. Research Report Series*, 1995, 7.

[23] ZHAO, Bin; LI, Xuelong; LU, Xiaoqiang. Hierarchical recurrent neural network for video summarization. In: *Proceedings of the 25th ACM international conference on Multimedia*. 2017. p. 863-871.

[24] ANTUNES, Alexandre; LAFLAQUIERE, Alban; CANGELOSI, Angelo. Solving bidirectional tasks using MTRNN. In: *2018 Joint IEEE 8th International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*. IEEE, 2018. p. 19-25.

[25] GRAVES, Alex, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 2016, 538.7626: 471-476.

[26] SUN, Guo-Zheng, et al. The neural network pushdown automaton: Model, stack and learning simulations. *arXiv preprint arXiv:1711.05738*, 2017.