Catedra de Structura sistemelor de calcul

Recunoasterea cifrelor scrise de mana folosind o retea artificiala neuronala

Dunca Lucian - Marian
Grupa 302219
Profesor Laborator: Vlad Ratiu
Data: 20.11.2020

Content

1. Rezumat

O retea neuronala artificiala este o colectie de unitati numite neuroni, acestia au ca model neuronii creierului uman. Printr-o retea neuronala putem rezolva diferite probleme din lumea reala prin aplicarea in practica a modului in care creierul uman functioneaza.

S-a folosit aceasta tehnologie pentru a modela un program care va avea capabilitatea sa recunoasca si sa clasifice niste imagini cu cifre scrise de mana. Obiectivele principale ale proiectului sunt conturarea unei arhitecturi capabile de a rula pe o placa FPGA si de a prezice cifra din imagine.

Pentru a rezolva aceasta problema am utilizat mai intai un program scris in limbajul python care va antrena un model de recunoastere a cifrelor dintr-o imagine, setul de date este cel oferit de MNIST. Pentru scrierea propriu zisa a aplicatiei care va rula pe placuta FPGA am folost limbajul de descriere hardware VHDL. In acesta am implementat o retea neuronala cu propagare inainte, deoarece antrenarea modelului se face software si nu este necesara implementarea propagarii inapoi.

Rezultatele dezvoltarii RAN au fost unele asteptate, precizia retelei nu este una foarte mare datorita utilizarii reprezentarii in virgula mobile dar nu este nici foarte mica. Asadar aceasta implementare este una fiabila care duce la cresterea vitezei calculelor necesare in reteaua neuronala.

2. Introducere

Proiectul urmareste implementarea unei retele artificiale neurale (RAN) cu propagare inainte in limbajul "Very high speed integrated circuit hardware description language" (VHDL) pentru o placa "Field programmable gate array" (FPGA.) RAN este o tehnica puternica si flexibila de invatarea automata care simuleaza creierul uman in sensul in care proceseaza semnale de intrare si le transforma in semnale de iesire [1]. Limbajul VHDL este un limbaj de descriere hardware folosit pentru a descrie sisteme digitale si mixte cum ar fi FPGA. Placutele FPGA sunt niste circuite integrate foarte rapide care sunt create pentru a fi configurate de catre client. Datorita acestei proprietati aceste circuite integrate sunt des folosite in domeniul RAN.

RAN fac parte din domeniul invatarii profunde (deep learning), care face parte la randul ei dintr-un domeniu mai larg, cel al invatarii automate (machine learning). Invatarea automata este o ramura a inteligentei artificiale, al carui obiectiv este de a dezvolta tehnici care dau posibilitatea calculatoarelor de a invata. Mai precis, se urmareste sa creeze programe capabile de generalizare pe baza unor exemple. RAN sunt niste structuri de procesare a informatiei prin simularea artificiala al fiziologiei si structurii creierului uman. Sistemul neural uman functioneaza pe baza legaturilor intre neuroni. Semnalul de intrare al unui neuron este compus din nenumarate semnale ale altor neuroni care sunt conectati la el, atunci cand suma tuturor semnalelor depaseste o anumita limita, neuronal va genera un semnal electric care se va propaga prin sinapse la alti neuroni.[2]

RAN sunt formate din unitati elementare numiti neuroni combinati dupa o anumita arhitectura. De exemplu, acestia pot sa fie aranjati in multiple straturi. Arhitecturile multistrat au urmatoarele componente: [2]

1. Stratul cu datele de intrare, format din n neuroni (un neuron pentru fiecare semnal de intrare)
2. Stratul ascuns, format din unul sau mai multe straturi formate din m neuroni.
3. Stratul de iesire, format din p neuroni (un neuron pentru fiecare semnal de iesire)

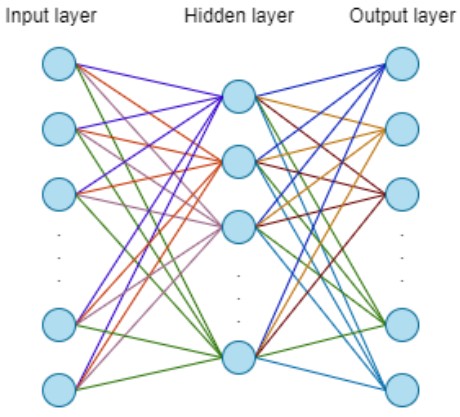


Fig 1. Straturi RAN

Proiectului are ca scop implementarea unei RAN de recunoastere a cifrelor scrise de mana, in limbajul de descriere hardware VHDL. Mai apoi rularea pe o placa de dezvoltare FPGA pentru generarea rezultatelor. Pentru realizarea acestuia se urmaresc urmatoarele obiective:

1. Dezvoltarea modelului RAN in python
2. Importarea setului de imagini pe care se va antrena modelul
3. Antrenarea modelului si generarea greutatilor fiecarui strat al RAN in python
4. Implementarea RAN in limbajul VHDL
5. Testarea RAN cu ajutorul greutatilor obtinute din modelul retelei din python
6. Rularea pe placa de dezvoltare BASYS3

Implementarea aleas pentru RAN in limbajul VHDL presupune multiple compromisuri. Antrenarea modelului unei RAN este un lucru de o dificultate ridicata pentru placutele FPGA doarece acestea nu au capabilitatea de a face calcule in virgula mobila. Asadar, se abordeaza implementarea unei RAN strict cu propagare inainte care nu presupune antrenarea modelului ci doar utilizarea unui model gata antrenat. Rolul acesteia va fi doar de a ne da rezultate in urma aplicarii unor imagini la intrarea retelei.

Pentru obtinerea modelului care va fi folosit pe placuta, se va folosi o implementare software in limbajul python care va creea si antrena modelul. Modelul va fi antrenat pe imagini care se vor importa din baza de date MNIST. Imaginile importate sunt sub forma unei matrici cu marimea de 28 pe 28 de pixeli. Pentru a reduce numarul de neuroni si greutati generate, modelul va fi antrenat cu imagini de 20 pe 20 de pixeli. Imaginile sunt reduse cu ajutorul unei interpolari cubice, astfel imaginea isi mentine forma. Fiecare pixel din imagine va avea o valoare cuprinsa intre 0 si 255 (scala gri). Se va seta fiecare pixel cu valoarea mai mare decat 0 pe 1 pentru a reduce complexitatea calculelor.

Deoarece greutatile generate vor fi reprezentate in virgula mobila si aceasta reprezentare nu este suportata de limbajul VHDL, ne vom folosi de reprezentare in punct fix, prin scalarea tuturor greutatilor cu 10000. Aceasta reprezentare insa va reduce din precizia rezultatelor.

In sectiunile urmatoare se vor prezenta diferite aspecte ale proiectului: fundamentarea teoretica se prezinta baza teoretica pe care se bazeaza proiectul precum si tehnologiile care pot sa fie utilizate, proiectare si implementare reprezinta partea principala a proiectului care contine descrierea fiecarei etape parcurse prentru realizarea obiectivelor proiectului, rezultate experimentale unde se demonstreaza ca proiectul implementat a fost implementat cu success si rezultatele obtinute sunt valide.

3. Fundamentare teoretica

Pentru a intelege cum functioneaza o RAN cu propagare inainte vom porni de la unitatea de baza a acesteia, perceptronul (Fig 2). Perceptronul este precursorul neuronului din RAN cu propagare inapoi, acesta este format doar dintr-un strat de intrare, unul de iesire, o functie de insumare si o functie de activare.

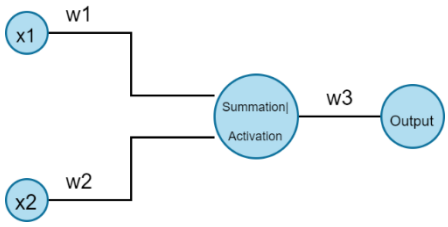


Fig 2. Perceptron

Imaginea de mai sus reprezinta un singur perceptron, dar daca avem multe intrari si multe date, un singur perceptron nu este de ajuns. Se va mari numarul de perceptroni formandu-se modelul de baza a unei RAN cu propagare inainte (Fig 3). Fiecare linie intre perceptroni este reprezentata de o greutate, valoare perceptronilor dintr-un strat poate sa fie scris ca o inmultire matriceala intre o matrice a greutatilor si o matrice coloanal a intrarilor (Fig 4).

Fig 3

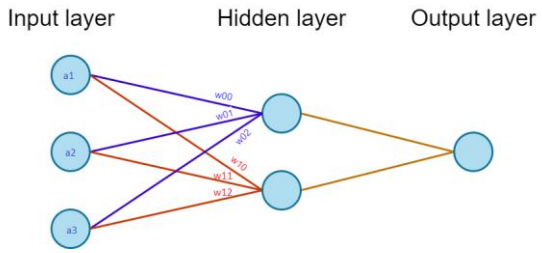


Fig 4

$$\begin{pmatrix} w^0 & w^1 & w^2 \\ w^{10} & w^{11} & w^{12} \end{pmatrix} \cdot \begin{pmatrix} a1 \\ a2 \\ a3 \end{pmatrix}$$

Procesul de antrenare al unui model al unei RAN implica direct aceste greutati. Prin antrenarea unei RAN se intelege modificarea greutatilor cu scopul ca predictia retelei va fi aceeasi cu predictia asteptata in etapa de antrenare, astfel reducandu-se valoarea erorii din retea. Aceasta “antrenare” se realizeaza prin mici modificari ale greutatilor care sunt reprezentate in general de numere in virgula mobile.

Pentru implementarea acestor greutati in limbajul VHDL se va folosi reprezentarea in punct fix pentru a exprima numeral in virgula mobile obtinute prin antrenarea modelului RAN. Prin folosirea reprezentarii in puct fix se va reduce precizia RAN si ii va creste complexitatea calculelor necesare datorita inmultirii unor numere cu valori mari, insa este necesara abordarea acesteia deoarece reprezentarea in virgula mobile nu este acceptata in limbajul de descriere VHDL fara a fi creat de catre noi.

O alta componenta vitala intr-o RAN este functia de activare care are ca rol convertirea sumei ponderate a semnalelor de intrare a unui neuron intr-un semnal de iesire. Exista nenumarate functii de activare care se folosesc in functie de ce avem nevoie. Cea mai simpla functie de activare din punct de vedere al nevoii computationale este functia RELU : f(x) = max(0,x) (Fig 5). Alte functii folositoare sunt functia sigmoida, folosita frecvent deoarece simplifica calculele ulterioare prin maparea semnalelor de iesire in intervalul (0,1). La fel si functia tanh care mapeaza valorile in intervalul (-1,1). Functiile prezentate precedent sunt folosite maim ult in straturile ascunse datorita proprietatilor lor. O alta

functie de activare importanta, folosita pentru obtinerea rezultatelor in ultimul strat este functia SoftMax. Aceasta functie este o normalizare exponentiala care mapeaza rezultatele intr-o distributie de probabilitati, avand suma componentelor 1.

Fig 5

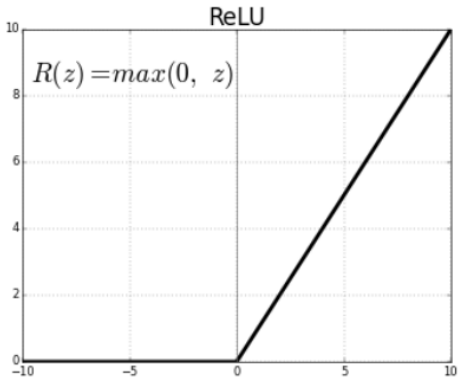
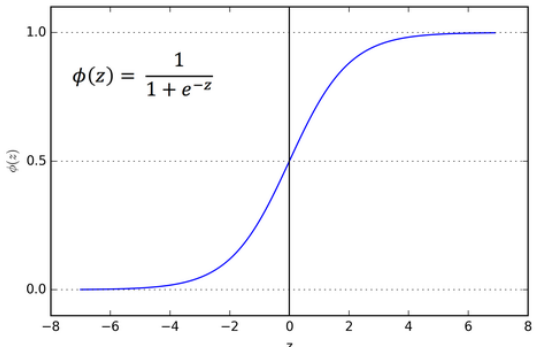


Fig 6



RAN sunt capabile de a invata si din acest motive le trebuie antrenate mai intai. Exista cateva metode de invatare:

- **Invatarea supravegheta:** implica un professor si RAN insasi, aceasta va ghici raspunsul dorit iar profesorul ii va spune ulterior raspunsul corect. Mai apoi reteaua va compara raspunsul sau cu cel oferit de catre professor si va face ajustarile necesare in functie de eroarea existenta.

- **Invatarea nesupravegheta:** este necesara atunci cand nu exista un set de date cu raspuns cunoscut. In acest caz, se vor imparti datele in grupuri in functie de un criteriu bazat pe datele prezente.

- **Invatare consolidate:** aceast tip de invatare se bazeaza pe observatie, RAN face o decizie prin observarea mediului. Daca decizia este falsa, reteaua isi va ajusta greutatile pentru a fi capabila sa faca o decizie diferita data viitoare.

Crearea unei RAN pentru recunoasterea cifrelor de mana nu este un subiect nou abordat. Exista nenumarate lucrari despre aceasta tema si multe abordari diferite [1][4]. O abordare posibila este folosind retele neurale convolutionale, abordare prezentata in lucrarile [5] si [6] care au una dintre cele mai bune performante in procesarea imaginilor. Aceasta abordare presupune introducerea unor straturi suplimentare numite straturi de convolutie si pooling care au rolul de a extrage caracteristici dintr-o imagine astfel crescand precizia si viteza.

Creare RAN poate avea o abordare software folosind librariile keras/tensorFlow in limbajul de programare python, se pot gasi numeroase articole pe aceasta tema. Pe de alta parte, contruirea unei RAN intr-un limbaj de descriere hardware nu este un subiect asa de des intalnit comparat cu varianta software. O analiza a metodei de implementare a une RAN pentru o placa FPGA se poate gasi in articolul [7].

4. Proiectare şi implementare

Pentru implementarea RAN in limbajul de descriere hardware VHDL s-a dezvoltat mai intai modelul acesteia printr-o metoda software in limbajul python cu ajutorul framework-ului “keras”. A fost ales sa se dezvolte o RAN cu propagare inapoi care are 3 straturi, o alta alternativa prezentata in ‘Fundamentarea teoretica’ ar fi fost o RAN convolutiva. S-a ales aceasta metoda deoarece complexitatea imaginilor nu este ridicata, fiind posibila procesarea acestora folosind o RAN simpla, fara tehnicile prezente in retelele convolutive.

4.1 Implementarea software a RAN

Pentru crearea RAN s-a folosit framework-ul ‘keras’ care ne ajuta sa construim straturi cu un numar de neuroni variabili si in acelasi timp sa alegem functia de activare care va fi aplicat pe fiecare strat in parte. Mai intai s-au important imaginile folosite pentru a antrena si testa reteaua. Imaginile folosite sun cele din baza de date MNIST, acestea au fost inarcate sub forma de matrici cu marimea de 28 pe 28 de pixeli. Setul de date a fost impartit in 2 parti, 60.000 de poze pentru antrenarea sistemului si 10.000 de poze pentru testarea acestuia.

Initial s-a ales ca stratul de intrare sa aiba 36 de neuroni, stratul ascuns un numar de 5 neuroni iar implicit stratul de iesire un numar de 10 neuroni. Datorita reprezentarii greutatilor in reprezentarea in punct fix, rezultatele acestei retele au fost scazute. Asadar, s-a ales cresterea numarului de neuroni chiar daca complexitatea calculelor va fi mai ridicata.Prin urmare, RAN are in stratul de intrare un numar de 400 de neuroni (indicate de o imagine de 20 pe 20 de pixeli), in stratul ascuns un numar de 10 de neuroni, iar in stratul de iesire tot 10 neuroni.

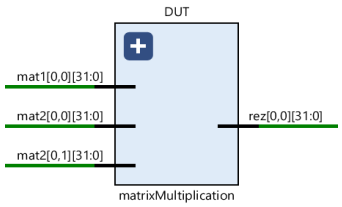
Imaginile importate din baza de date MNIST au o marime de 28 pe 28 de pixeli. Pentru a obtine imagini cu marimea dorita vom folosi o interpolare cubica care ne va reduce imaginea la marimea dorita, in acelasi timp nedegradand imaginile. In continuare vom prelucra imaginile pentru a putea fi direct trimise catre modelul RAN. In primul rand se vor transforma matricile de marime 20 pe 20 intr-o matrice cu o singura linie cu marimea de 1 pe 400. Mai apoi se vor reprezenta variabile de iesire cu ajutorul “one hot encoding” pentru a fi posibila procesarea acestora de catre RAN. Crearea straturilor se va realiza utilizand metode din framework-ul ‘keras’. Stratul ascuns va avea 10 neuroni si functia de activare va fi reLU. Stratul de iesire va avea tot 10 neuroni (reprezentand cele 10 cifre posibile) si functia de activare va fi o normalizare simpla. La final se va active antrenarea modelul retelei prin apelarea unei functii din keras.

4.2 Implementarea hardware in limbajul VHDL

Pentru modelarea RAN in limbajul de descriere hardware VHDL este nevoie de a proiecta mai multe module cu scopuri diferite. Pentru a stoca informatiile intr-un mod eficient si de a ne ajuta in momentul in care trebuie facute calcule, se va declara o structura noua sub forma de matrice. Aceasta structura are o marime generica care se poate schimba si va avea rolul de a memora toate datele RAN care sunt reprezentate de catre semnalele de intrare, imagini, greutati si semnale de iesire intermediare. Se va folosi un pachet pentru a declara aceasta structura de date impreuna cu toate constantele aferente necesare reprezentate de catre greutatile si imaginile RAN.

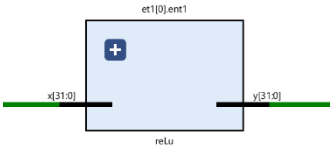
4.2.1. Implementarea modulului de inmultire

Pentru a simplifica inmultirea greutatilor cu cea a semnalelor de intrare, ne vom ajuta de catre structura folosita in stocarea datelor si se va implementa un modul de inmultire matriceala generic. Acest modul va putea primi ca intrari 2 matrici de marime m x n respectiv n x p si va crea o matrice ca semnal de iesire de marimea m x p



4.2.2. Implementarea modulului functiei de activare reLU si normalizare

Se vor implementa 2 module pentru definirea celor 2 functii de activare. Definirea functiei reLU este triviala, necesitand alegerea maximului intre 0 si valoarea de input. Pe de alta parte, pentru stratul de iesire se va folosi o functie de normalizare simpla deoarece in urma deciziilor de implementare alese nu putem aplica functia softmax asupra valorilor obtinute in ultimul strat.



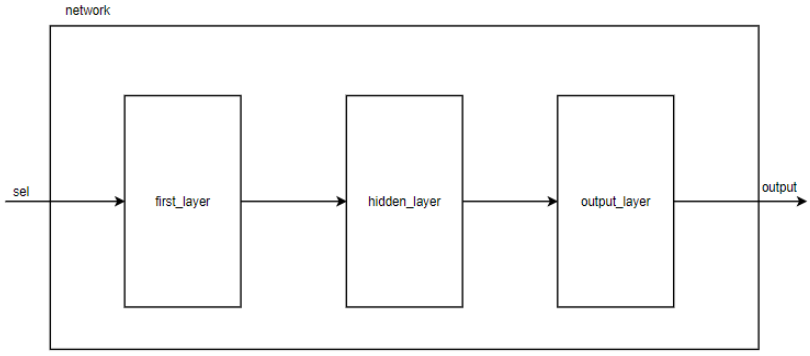


Fig 7 - Schema bloc a sistemului

4.2.3. Implementarea primului strat

Un prim modul necesar este primul strat al retelei care are rolul de a procesa imaginile sub forma de matrice si de a le transforma intr-o matrice cu un singur rand. Aceasta noua matrice obtinuta va fi folosita ca si semnal de intrare de catre urmatorul strat al RAN (stratul ascuns).

4.2.4. Implementarea stratului ascuns

Stratul ascuns va primi ca input imaginile procesate de catre stratul de intrare. Acesta va folosi modului de multiplicare pentru a genera iesirile prin inmultirea greutatilor cu cea a input-ului. Inainte de a trimite spre iesire valorile obtinute din multiplicare, se va aplica functia de activare reLU asupra fiecarei valori.

4.2.5. Implementarea stratului de iesire

Stratul de iesire va primi ca input valorile trimise de catre stratul ascuns. Acesta va folosi modului de multiplicare pentru a genera iesirile prin inmultirea greutatilor cu cea a input-ului. Inainte de a trimite spre iesire valorile obtinute din multiplicare, se va aplica functia de activare pentru a normaliza rezultatele si de a le aduce in intervalul (0,100), aceste valori reprezentand sansa ca o cifra specifica sa fie in imagine.

4.2.6 Implementarea modului principal

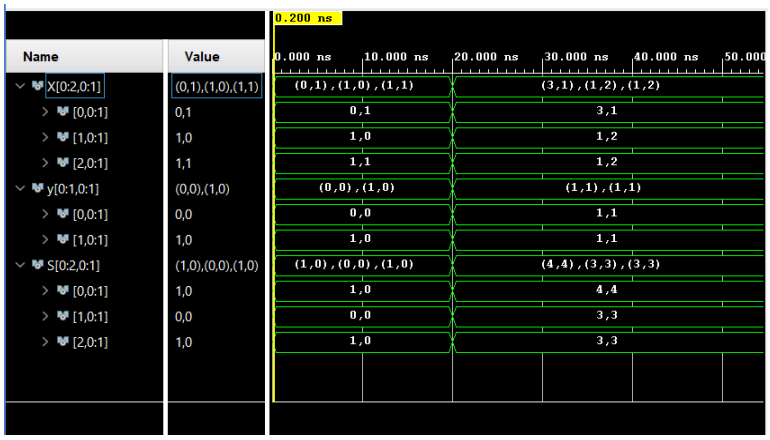
Rolul modului principal este de a instantia restul entitatilor RAN si de a le lega intre ele cu ajutorul unor semnale intermediare. Se foloseste un semnal generic pentru declararea marimii imaginii (se considera imaginea ca fiind o matrice patratrica cu latura de lungime data de semnalul generic), iar ca semnal de intrare avem un stimul care selecteaza imaginea care va trece prin RAN. Semnalul de iesire va fi o matrice cu 10 coloane in care vom avea o distributie de valori care reprezinta probabilitatea ca imaginea sa reprezinte cifra de pe care pozitie se afla valoarea respective.
Imaginile sunt niste constante declarate intr-un pachet care continue toate constantele necesare sistemului. O data cu selectia imaginii, primul strat primeste aceasta imagine pe care o aplatizeaza si o trimite stratului ascuns printr-un semnal intermediar. Stratul ascuns primeste aceasta matrice pentru procesare,totodata se declara semnalele generice necesare acesteia (numarul de neuroni si marimea matricii aplatizate). Rezultatul este trimis catre ultimul strat. Stratul de iesire va face operatiile necesare si va aplica functia de normalizare, ulterior transmitand matricea finala catre iesire. Aceasta matrice este conectata la semnalul de iesire al modului principal.

Rezultate experimentale

Pentru implementare si testarea s-a folosit limbajul VHDL, mediul de dezvoltare utilizat pentru implementare este 'Vivado Design Suite 2019.2'. Pentru a testa implementarea proiectului s-au folosit numeroare banci de test pentru fiecare modul in parte. Un banc de test este un cod scris in VHDL care ne permite sa avem un set de stimuli repetitive si portabili pentru nenumartate simulari. Acesta poate sa fie un simplu fisier cu un semnal de ceas si date de intrare sau un fisier mai complex cu verificari de erori si testare conditionala. Pentru testare s-a ales simpla introducere a unor stimuli si verificarea corectitudinea datelor de iesire

Testarea modului de inmultire matriceala

Pentru a testa modulul de inmultire matriceala s-a create un banc de test separate in care s-au insantiat toate semnalele necesare instantirii modulului de inmultire. S-au declarant 2 semnale care vor fi folosite ca stimuli de intrare al modulului, X si Y. Se poate observa ca in urma inmultirii celor 2 matrici X si Y, rezultatul este corect din punct de vedere matematic.

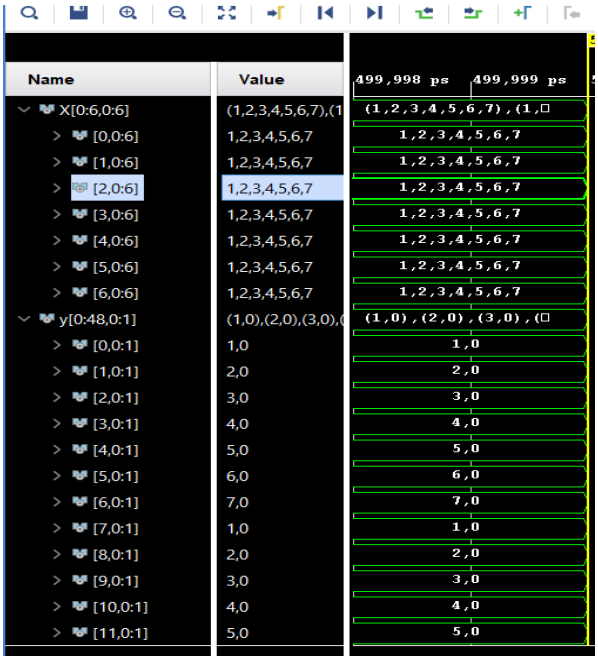


Testarea modului de normalizare

Pentru a testa modulul de inmultire matriceala s-a create un banc de test separate in care s-au insantiat toate semnalele necesare instantirii modulului de inmultire. S-au declarant un semnal care va fi folosit ca stimul de intrare al modulului. Ca iesire se va putea observa un alt semnal care va avea un procent care reprezinta rata de acoperire a valorii de pe pozitia respective in suma totala.

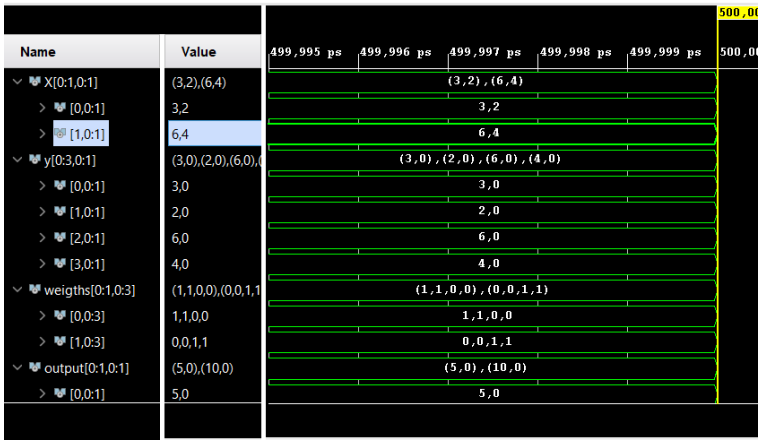
Testarea modulului primului strat

Modulul primului strat s-a testat intr-un mod similar cu modulele precedente. S-a create un banc de test si s-au declarant toate semnalele necesare instantierii entitatii primului strat. Stimulul de intrare al modulului este o matrice cu marimea de 7 pe 7. Iesirea obtinuta este o matrice cu marimea de 49 pe 1 care este obtinuta din aplatizarea matricii de la intrare.



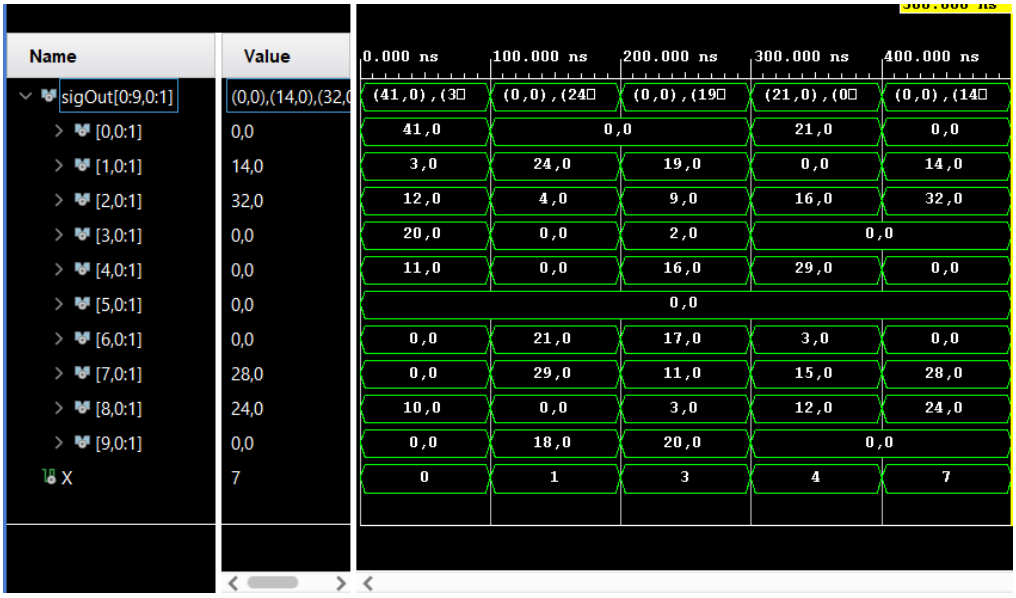
Testarea modului stratului ascuns

Pentru testarea stratului ascuns se vor instantia entitatile primului strat si al celui ascuns. Ca stimuli de intrare pentru primul strat se va folosi o matrice de 2 pe 2 care va fi trimisa catre primul strat. Acesta va procesa matricea si o va aplatiza pentru a putea sa fie folosita de catre stratul ascuns. Stimulii de intrare pentru stratul ascuns va fi matricea aplatizata impreuna cu matricea greutatilor. Rolul modulului celui de-al 2 lea strat este de a inmulti matricea greutatilor cu cea a valorilor din matricea aplatizata, mai apoi sa aplice functia de activare reLU asupra valorilor obtinute inainte sa le trimita spre iesire.



Testarea retelei neurale

Pentru testarea retelei neuronale a fost create un banc de test unde s-a instantiate modului principal al retelei ‘network’. S-a ales sa se testeze reteaua cu ajutorul unor imagini de 20 pe 20 de pixeli. A fost declarant un stimul pentru selectarea imaginii transmise RAN, acest stimul va fi schimbat pe parcursul simularii pentru a vedea comportamentul RAN pentru mai multe imagini. Se vor folosi 5 imagini cu 5 cifre diferite si anume cifrele: zero, unu, trei, patru si sapte.



Valoarea lui X reprezinta o imagine care continue cifra indicata de numarul semnalului. Se poate observa din imagine procentele obtinute pentru fiecare imagine in parte. Precizia sistemului nu este una ridicata, avand o precizie de aproximativ 40% datorita reprezentarii greutatilor in punct fix. Totodata distributia acestora este una obtinuta prin normalizare simpla si nu printr-o functie care ne ofera o distributie logaritmica.

Concluzii

Scopul proiectului a fost de a implementa o RAN capabila sa recunoasca intr-o imagine o cifra scrisa de mana si sa ne spuna sub forma unui procent care este sansa ca in imagine sa fie cifra respective. Obiectivele principale fiind dezvoltarea modelului RAN printr-o implementarea software pentru a obtine greautatile unui model antrenat si a le putea utiliza pentru dezvoltarea variantei hardware a RAN. Utilizarea unnei retele neurale pe o placuta FPGA duce la o imbunatatire a vitezei de calcul. Din acest motiv RAN sunt de obicei preantrenate si apoi puse sa ruleze pe placutele de dezvoltare deoarece antrenarea sistemului se face o singura data si nu are rost sa se dezvolte o implementare hardware pentru a antrena reteaua direct pe placa de dezvoltare.

Utilizarea reprezentarii in punct fix a greutatilor a dus la o scadere a performantei retelei, acesta fiind un compromise datorita lipsei reprezentarii numerelor in virgula mobile in limbajul VHDL. Asadar reteaua lucreaza doar cu numere intregi reprezentate pe 4 octeti si nu este necesara folosirea numerelor in virgula mobile.

Exista multe posibilitati de dezvoltare ulterioara a retelei: o mare imbunatatire ar fi dezvoltarea unei reprezentari in virgula mobile in limbajul VHDL si utilizarea acesteia pentru greutati. O data cu posibilitatea reprezentarii in virgula mobile se poate utiliza si functia softMax care nu a putut fi implementat datorita reprezentarii in punct fix. O alta posibila dezvoltare ar putea fi schimbarea tipului retelei neuronale in una convolutionala care ar putea sa inbunatateasca viteza si sa scada complexitatea calculelor necesare.

Bibliografie

[1] Z. Zhang, " A gentle introduction to artificial neural networks," *Annals of Translational Medicine,* p. 7, October 2016 .

[2] C. Gallo, "Artificial Neural Network Tutorial," *Encyclopedia of Information and Technology, Third Edition,* p. 13, january 2015 .

[3] J. Brownlee, "How to Develop a CNN for MNIST Handwritten Digit Classification," 24 August 2020. [Online]. Available: https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-from-scratch-for-mnist-handwritten-digit-classification/.

[4] S. Ahlawat, "Improved Handwritten Digit Recognition Using Convolutional Neural Networks (CNN)," 25 May 2020 .

[5] I.-J. K. a. X. Xie, "Handwritten Hangul recognition using deep convolutional," p. 29, 2016 .

[6] "Image Classification in 10 Minutes with MNIST Dataset," [Online]. Available : https://towardsdatascience.com/image-classification-in-10-minutes-with-mnist-dataset-54c35b77a38d.

[7] F. W. Wibowo, "An Analysis of FPGA Hardware Platform Based Artificial Neural Network," 2019

[8] M. Nielsen, "Using neural nets to recognize handwritten digits," December 2019. [Online]. Available: http://neuralnetworksanddeeplearning.com/chap1.html .

[9] "7 Types of Neural Network Activation Functions: How to Choose?," [Online]. Available: https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/

[10] "Deep Learning Project – Handwritten Digit Recognition using Python," [Online]. Available: https://data-flair.training/blogs/python-deep-learning-project-handwritten-digit-recognition/ .

[11] "Perceptrons and Multi-Layer Perceptrons: The Artificial Neuron at the Core of Deep Learning," [Online]. Available: https://missinglink.ai/guides/neural-network-concepts/perceptrons-and-multi-layer-perceptrons-the-artificial-neuron-at-the-core-of-deep-learning/ .

[12] Udemy, "Machine learning from A to Z," [Online]. Available: https://www.udemy.com/course/machinelearning/.

[13] 3Blue1Brown, "But what is a Neural Network? | Deep learning, chapter 1," 2017. [Online]. Available: https://www.youtube.com/watch?v=aircAruvnKk .

[14] "The Complete Guide to Artificial Neural Networks: Concepts and Models," 2018. [Online]. Available: https://missinglink.ai/guides/neural-network-concepts/complete-guide-artificial-neural-networks/ .

[15] 3. SERIES, "Gradient descent, how neural networks learn | Deep learning, chapter 2," [Online]. Available: https://www.youtube.com/watch?v=IHZwWFHWa-w&t=590s .

[16] "Neural Networks for Image Recognition: Methods, Best Practices, Applications," [Online]. Available: https://missinglink.ai/guides/computer-vision/neural-networks-image-recognition-methods-best-practices-applications/ .