Implementarea unei rețele neuronale pe un circuit FPGA de tip PYNQ-Z1

Andrei Sântoma
grupa 30236

Universitatea Tehnică din Cluj-Napoca

Structura Sistemelor de Calcul
prof. Dragoș Lișman

Ianuarie 2020

Cuprins

1 Rezumat

Domeniul învățării automate a acaparat lumea tehnologiei în ultimii ani, iar o subarie din acesta care s-a remarcat în mod special prin rezultatele extraordinare obținute în rezolvarea problemelor este aceea a învațării profunde. Problema pe care acest proiect își propune să o rezolve este implementarea pe un circuit FPGA a unei rețele neuronale care să fie capabilă să clasifice, cu o precizie mai mare de 85%, imagini ce conțin numere scrise de mână. Modelarea structurii a fost efectuată cu succes pe circuitul PYNQ-Z1, utilizând limbajul de programare Python, iar rezultatul obținut la finalul testării a fost o precizie de 89.11%.

2 Introducere

Învățarea profundă este bazată pe rețele neuronale artificiale, care sunt structuri ce, într-o oarecare măsură, reproduc modul în care funcționează neuronii și conexiunile dintre aceștia în cadrul organismelor biologice.

Rețelele neuronale artificiale sunt colecții de unități interconectate (denumite *neuroni*) organizate în straturi. Asemănător cu sinapsele biologice, conexiunile dintre neuronii artificiali reprezintă căi prin care circulă informație, iar fiecare neuron poate primi, prelucra și transmite mai departe către un altul informația respectivă. Conexiunilor dintre neuroni le este atribuită o pondere, astfel încât unele au importanță mai mare decât altele. Domeniul rețelelor artificiale este unul de mare importanță la momentul curent datorită aplicabilității în probleme variate și care ar fi dificil (sau chiar imposibil) de rezolvat prin utilizarea algoritmilor clasici, cum ar fi: procesare de imagini, diagnoză medicală, conducere automată, data mining etc.

În cele ce urmează este prezentată implementarea unei rețele neuronale artificiale pe un circuit FPGA (mai precis, *PYNQ-Z1*) care să se antreneze în vederea recunoașterii unor numere de o cifră scrise de mână, stocate ca imagini de dimensiune 28×28 pixeli, cunoscute sub denumirea de *Setul de date MNIST* [1].

Soluția propusă în cazul acestei probleme este o rețea neuronală simplă de tip *feedforward* formată din trei straturi de neuroni (perceptroni) și implementată utilizând limbajul de programare Python (de a cărui suport nativ beneficează circuitul PYNQ-Z1 susmențional), acesta fiind cel mai folosit în rândul dezvoltatorilor de sisteme de învățare automată.

3 Fundamentare teoretică

Perceptronul [2] este un algoritm simplu de clasificare binară (decide dacă o intrare furnizată sub forma unui vector aparține sau nu unei anumite clase). Clasificarea efectuată de către un perceptron este de tip liniar, ceea ce înseamnă că acesta folosește o *funcție de activare* [3] liniară ce combină intrarea vectorială cu un set de ponderi asociate fiecărei componente. Pentru intrarea x, funcția arată astfel:

$$f(x) = \begin{cases} 1, & \text{dacă } w \cdot x + b > 0 \\ 0, & \text{altfel} \end{cases}$$

unde x reprezintă intrarea vectorială, w reprezintă vectorul de ponderi, $w \cdot x$ este produsul lor scalar, iar b este *biasul inductiv* [4].

Rețelele neuronale de tip *feedforward* [5] sunt acele rețele în care conexiunile dintre neuroni nu formează cicluri (informația se propagă într-o singură direcție, de la primul strat al rețelei până la ultimul, sau invers).

O rețea de perceptroni organizați pe mai multe straturi [6], denumită de aici înainte MLP (*Multilayer perceptron*), este o rețea neuronală artificială de tip feedforward în care neuronii sunt reprezentați de perceptroni. În cadrul fiecărui strat, fiecare neuron este conectat la toți neuronii din stratul următor, deci rețeaua este complet conexă. În mod tradițional (și în cazul acestui proiect), rețeaua este alcătuită din trei straturi de neuroni: unul de intrare, unul ascuns și unul de ieșire. Neuronii din stratul de intrare folosesc o funcție de activare liniară, iar ceilalți folosesc funcții de activare non-liniare, precum *funcția sigmoid* [7]. Acest aspect din urmă permite rețelelor MLP să clasifice date ce nu sunt liniar separabile [8], spre deosebire de perceptronul simplu, care poate să clasifice doar seturi de date liniar separabile.

Pentru procesul propriu-zis de învățare, rețeaua MLP procesează fiecare intrare din setul de date din care trebuie să extragă caracteristici și își adaptează în mod dinamic ponderile conexiunilor dintre neuroni printr-un algoritm numit *backpropagation* [9]. Această adaptare se face în urma evaluării unei funcții de eroare după trecerea unei intrări prin straturile rețelei neuronale, care efectuează compararea rezultatului obținut la ieșire cu rezultatul

așteptat. Pentru optimizarea adaptării ponderilor se folosește algoritmul *Adam* [10].

În vederea manipulării setului de date MNIST se folosesc două librării Python, mai precis *numpy* [11] și *pandas* [12]. Numpy este o librărie ce oferă o implementare eficientă pentru stocarea de obiecte N-dimensionale în Python și funcții variate și avansate pentru prelucrarea acestora. Pandas oferă utilități ce permit stocarea eficientă în memorie și analiza datelor tabelare numerice, totodată fiind folosită pentru a modela datele în așa fel încât acestea să poată fi transmise către diverse module de învățare automată.

4 Proiectare şi implementare

4.1 Metoda experimentală utilizată

Pentru a atinge scopul proiectului a fost aleasă o abordare ce se bazează pe o implementare pur software, datorită suportului nativ de Python de care beneficiază circuitul PYNQ-Z1. Principala motivație pentru această alegere a fost ușurința considerabil mai ridicată față de un limbaj de descriere hardware (precum VHDL) cu care se poate face implementarea unei arhitecturi relativ complexe (în cazul de față, un MLP).

4.2 Soluția aleasă

Precum este menționat anterior, s-a decis asupra implementării unei rețele neuronale artificiale de tip MLP de complexitate mai redusă, ce însumează trei straturi de neuroni:

- stratul de intrare: conține 784 neuroni, câte unul pentru fiecare pixel din imagine;
- stratul ascuns: conține 100 neuroni;
- stratul de ieșire: conține 10 neuroni corespunzători celor 10 clase (cifrele de la 0 la 9) în care rețeaua trebuie să încadreze imaginile.

Printre principalele motive pentru care s-a ales un model mai simplu se enumeră dimensiunile memoriei RAM și viteza de procesare reduse [1] ale circuitului (512MB RAM DDR3, respectiv două nuclee de procesare cu frecvența de 650MHz), dar și faptul că sistemul de operare care rulează pe circuit nu permite instalarea unor pachete mai complexe ce implementează rețele neuronale, cum ar fi *Tensorflow* [13] sau *Keras* [14].

Neuronii din straturile rețelei folosesc funcția de activare sigmoid, iar pentru optimizarea procesului de actualizare a ponderilor conexiunilor dintre

[1] în comparație cu un calculator de tip Desktop

neuroni se folosește Adam, care, în acest caz, oferă o viteză de execuție a procesului de învățare mai ridicată față de alte metode, precum *SGD* [15].

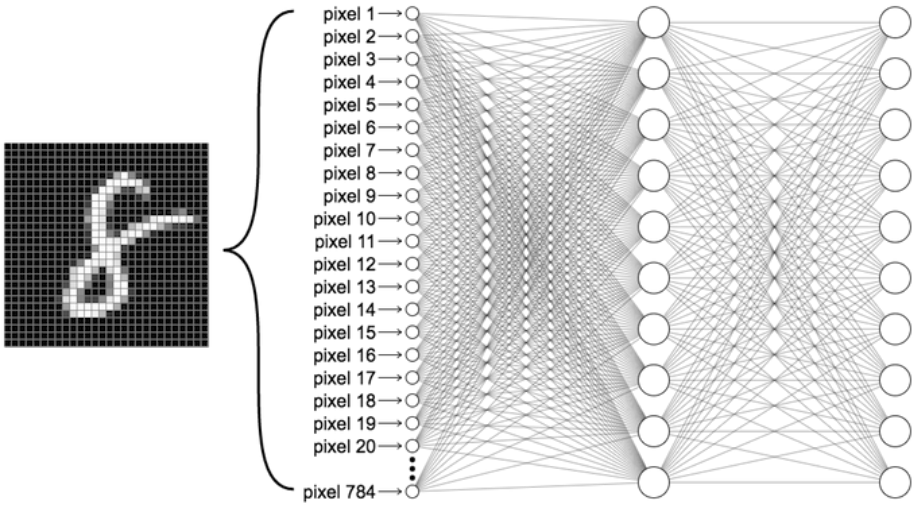

Fig. 1: MLP cu trei straturi (sursa: `https://ml4a.github.io/`)

4.3 Arhitectura generală a sistemului

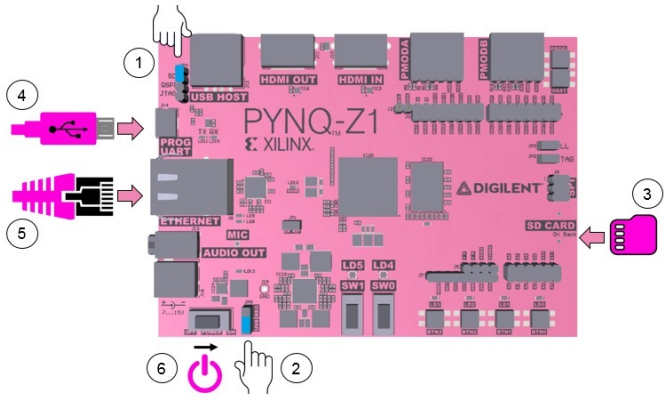

Fig. 2: circuitul PYNQ-Z1 (sursa: `https://pynq.readthedocs.io/`)

Sistemul este alcătuit din două componente, circuitul PYNQ-Z1 și un terminal care trebuie să fie capabil să comunice prin intermediul SSH (în acest

caz se folosește o mașină Linux), iar cele două sunt conectate printr-un cablu UTP și unul micro-USB. Terminalul este folosit pentru rularea comenzilor de instalare a diferitelor pachete necesare rulării proiectului, dar și pentru a putea folosi mediul Jupyter pus la dispoziție de către circuitul PYNQ-Z1.

4.4 Algoritmii implementați

Dintre algoritmii implementați se remarcă:

- procedura de feedforwarding;
- procedura de backpropagation;
- procedura de actualizare a ponderilor.

4.5 Manual de utilizare

4.5.1 Instalarea sistemului de operare al plăcuței PYNQ-Z1

Se descarcă imaginea sistemului de operare [16] apoi se scrie imaginea respectivă pe o cartelă SD de minim 8GB. Imaginea este bootabilă și sistemul ar trebui să pornească cu succes atunci când cartela este introdusă în slotul corespunzător de pe plăcuță.

4.5.2 Asigurarea conexiunii plăcuței la internet

Se asignează terminalului adresa statică `192.168.2.1`, apoi se conectează plăcuța la terminal prin cablul Ethernet. Se activează pe terminal opțiunea de *bridging* [17] pentru rețeaua wired a acestuia.

4.5.3 Conectarea terminalului la plăcuță

Se efectuează conexiunea dintre terminal și plăcuță prin SSH, la adresa `192.168.2.99`, cu numele de utilizator `xilinx` și parola `xilinx`.

4.5.4 Instalarea pachetelor Python

Se transmit prin SSH următoarele comenzi către circuitul PYNQ-Z1:

```
pip3 install numpy
pip3 install pandas
```

4.5.5 Antrenarea și evaluarea rețelei neuronale

Se copiază seturile de date și notebook-ul Jupyter pe plăcuță, utilizând protocolul *Samba* [18] (se accesează sistemul de fișiere al plăcii la adresa `smb:pynq/xilinx`, cu numele de utilizator `xilinx` și parola `xilinx`). Se accesează mediul Jupyter al plăcii navigând la adresa `192.168.2.99:9090` într-un browser de internet deschis pe terminal și se introduce parola `xilinx`, mai apoi se deschide din acest mediu notebook-ul Jupyter copiat anterior și se rulează codul celulă cu celulă (pentru a rula o celulă de cod se selectează aceasta, apoi se apasă `Ctrl+Enter`).

Se pot varia parametrii rețelei neuronale schimbând valorile variabilelor `epochs` (numărul de iterații de antrenare), `learning_rate` (rata de învățare) și `hidden_nodes` (numărul de noduri din stratul ascuns). Progresul intermediar și rezultatul final al procesului vor apărea pe ecran.

5 Rezultate experimentale

Proiectul a fost realizat utilizând limbajul de programare Python3 (versiunea 3.4.3 este instalată nativ pe circuitul PYNQ-Z1) în mediul de dezvoltare Jupyter. Antrenarea rețelei neuronale pe 16000 din cele 60000 de date de antrenare și 90 de iterații a rezultat într-o acuratețe a predicției de 89.11% pe setul de date de test. Antrenarea completă a însumat în jur de 30 de minute, cu fiecare iterație având o durată de aproximativ 20 de secunde. Rezultatul poate fi îmbunătățit dacă este crescut numărul de iterații de antrenare, respectiv numărul de intrări din setul de antrenare.

```
Epoch 70. Time elapsed: 1418.95s.
Epoch 71. Time elapsed: 1438.92s.
Epoch 72. Time elapsed: 1458.98s.
Epoch 73. Time elapsed: 1478.95s.
Epoch 74. Time elapsed: 1498.93s.
Epoch 75. Time elapsed: 1518.92s.
Epoch 76. Time elapsed: 1538.92s.
Epoch 77. Time elapsed: 1558.90s.
Epoch 78. Time elapsed: 1578.78s.
Epoch 79. Time elapsed: 1598.73s.
Epoch 80. Time elapsed: 1618.71s.
Epoch 81. Time elapsed: 1638.69s.
Epoch 82. Time elapsed: 1658.51s.
Epoch 83. Time elapsed: 1678.47s.
Epoch 84. Time elapsed: 1698.43s.
Epoch 85. Time elapsed: 1718.26s.
Epoch 86. Time elapsed: 1738.21s.
Epoch 87. Time elapsed: 1758.20s.
Epoch 88. Time elapsed: 1778.17s.
Epoch 89. Time elapsed: 1798.13s.
Total time elapsed:1798.1325s
```

Fig. 3: Durata procesului de învățare

```
Out[33]: 0.8911
```

Fig. 4: Proporția de imagini clasificate corect

6 Concluzii

Acest proiect a realizat cu succes antrenarea unei rețele neuronale de tip MLP pentru a fi capabilă să recunoască cifre scrise de mână stocate sub formă de imagini digitale. Rezultatele sunt mai slabe decât cele obținute în cazul antrenării folosind biblioteci specializate, dar diferențele nu sunt majore. Proiectul are scop pur didactic, dar o aplicație utilă a antrenării de rețele neuronale pe un astfel de dispozitiv ar putea fi, spre exemplu, recunoașterea de obiecte în imaginile ce sunt furnizate de un robot care se deplasează și acționează automat. Avantajul principal al unei astfel de implementări ar fi portabilitatea circuitului, iar un dezavantaj major este puterea redusă de calcul și memorie a acestuia. Ca și dezvoltări viitoare ar putea fi optimizat modul în care sunt încărcate datele în rețea pentru a economisi memorie RAM și introducerea de straturi suplimentare de neuroni.

7 Bibliografie

[1] Setul de date MNIST
http://yann.lecun.com/exdb/mnist/

[2] Perceptron
https://en.wikipedia.org/wiki/Perceptron

[3] Funcție de activare
https://en.wikipedia.org/wiki/Activation_function

[4] Bias inductiv
https://en.wikipedia.org/wiki/Inductive_bias

[5] Rețea neuronală de tip *feedforward*
https://en.wikipedia.org/wiki/Feedforward_neural_network

[6] MLP
https://en.wikipedia.org/wiki/Multilayer_perceptron

[7] Funcția sigmoid
https://en.wikipedia.org/wiki/Sigmoid_function

[8] Liniar separabilitate
https://en.wikipedia.org/wiki/Linear_separability

[9] Backpropagation
https://en.wikipedia.org/wiki/Backpropagation

[10] Diederik P. Kingma, Jimmy Lei Ba. *Adam: A Method for Stochastic Optimization*. ICLR, 2015
https://arxiv.org/abs/1412.6980

[11] Numpy
https://numpy.org/

[12] Pandas
https://pandas.pydata.org/

[13] Tensorflow
https://www.tensorflow.org/

[14] Keras
https://keras.io/

[15] SGD
https://en.wikipedia.org/wiki/Stochastic_gradient_descent

[16] Sistemul de operare pentru PYNQ-Z1
files.digilent.com/Products/PYNQ/pynq_z1_v2.1.img.zip

[17] Bridging
https://en.wikipedia.org/wiki/Bridging_(networking)

[18] Samba
https://en.wikipedia.org/wiki/Samba_(software)