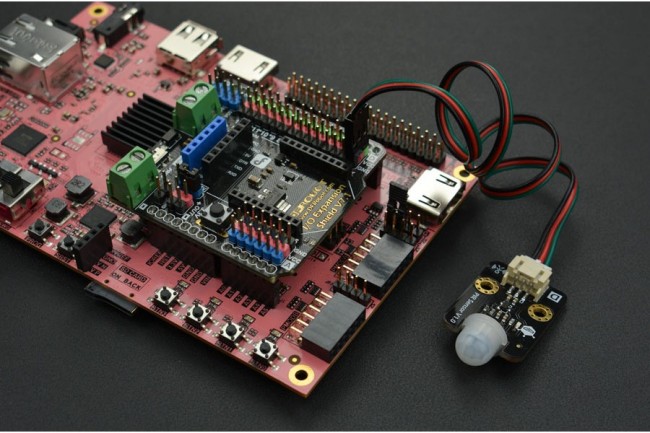
Implementacija neuronske mreze na FPGA plocama

  
  
  
  
  
  
  
  
  
  
  
  
  
  
Pavle Oprić 69/2020

Vuk Lazović 51/2020

# Sadrzaj

Contents

[Sadrzaj 2](#_Toc144926785)

[Uvod 3](#_Toc144926786)

[Potrebne/koriscene tehnologije 3](#_Toc144926787)

[Treniranje neuronske mreze 4](#_Toc144926788)

[Priprema slika I testiranje neuronske mreze 6](#_Toc144926789)

[Verilog implementacija neuronske mreze na FPGA ploci 11](#_Toc144926790)

# Uvod

Projekat se zasniva na implementaciji neuronskih mreža koje prepoznaju rukopis na Zynq 2 FPGA ploču. Cilj projekta je da se razvije sistem koji može da prepozna i klasifikuje rukopisne karaktere(cifre) u realnom vremenu koristeći Zynq 2 FPGA ploču kao hardversku platformu. Neuronske mreže će biti obučene na velikom skupu podataka zvanom “MINST”, koji sadrži veliku kolicinu rucno napisanih cifara, a zatim će biti implementirane na FPGA ploči kako bi se omogućila binarna interpretacija klasifikovanih brojeva pomocu 4 led diode koje ova ploca poseduje . Ovaj projekat je radjen u okviru predmeta “Projektovanje VLSI Sistema” na Prirodno matematickom fakultetu u Kragujevcu.

# Potrebne/koriscene tehnologije

- Xilinx

- Vivado Design Suite

- Python programski jezik/Python Notebook/pip

- Verilog programski jezik(System Verilog)

- Git

-FPGA Ploca(PYNQ-Z2 u nasem slucaju)

# Treniranje neuronske mreze

U direktorijumu python, skripta “data.py”(slika 1) učitava MNIST skup podataka rukom pisanih cifara iz datoteke pod nazivom “mnist.npz” koja se nalazi u poddirektorijumu “data” direktorijuma koji sadrži skriptu. Funkcija koristi biblioteku numpy za učitavanje podataka iz datoteke i zatim ih obrađuje kako bi ih pripremila za upotrebu u obuci mašinskog učenja.

A screenshot of a computer

Description automatically generated

Slika

U istom direktorijumu, python notebook skripta “project.ipynb” koristi biblioteke data, numpy, matplotlib.pyplot, sys, numpy i PIL za učitavanje i obradu podataka, kao i za prikaz rezultata.

Prvi deo skripte učitava MNIST skup podataka koristeći funkciju get\_mnist iz modula data, koju smo malopre pomenuli, I inicijalizuje težine(weights) i pristrasnosti(bias) neuronske mreže nasumičnim vrednostima iz uniformne raspodele. Zatim se postavlja stopa učenja i broj epoha za obuku.

Sledeći deo skripte sadrži glavnu petlju za obuku neuronske mreže. (slika 2) Petlja se izvršava za svaku epohu i prolazi kroz sve slike i oznake iz skupa podataka za obuku. Za svaku sliku, ulazni vektor se preoblikuje kako bi odgovarao dimenzijama ulaznog sloja neuronske mreže, a zatim se izračunavaju aktivacije skrivenog i izlaznog sloja koristeći backpropagation algoritam.

Nakon toga, skripta izračunava grešku na izlazu mreže kao srednju kvadratnu grešku između predviđanja mreže i stvarne oznake. Zatim se ažuriraju težine(weights) i pristrasnosti(bias) mreže koristeći gradijentni spust kako bi se minimizovala greška. Vidimo da se nasa neuronska mreza sastoji iz 784 ulaznih neurona(po jedan za svaki piksel), kao I da se skriveni sloj sastoji od 20, a izlaz od 10 vrednosti.

A computer screen shot of a program code

Description automatically generated

Slika

Na kraju svake epohe, skripta ispisuje tačnost mreže na skupu podataka za obuku kao procenat tačno klasifikovanih slika. Vidimo da nasa mreza ima sasvim zadovoljavajucu tacnost, gde nakon trece epohe, tacnost mreze dostize 93.35%.(slika 3)

A number of numbers on a white background

Description automatically generated

Slika

# Priprema slika I testiranje neuronske mreze

Da bi nasa neuronska mreza funkcionisala optimalno, potrebno je prvo pripremiti slike za upotrebu u Verilog jeziku. U istom direktorijumu, skripta “prepare\_image.py”(slika 4) koristi biblioteke sys, PIL i numpy za obradu slika i manipulaciju podacima.

Funkcija prepare\_real\_photo uzima putanju do slike kao ulazni parametar i vrši nekoliko koraka obrade kako bi pripremila sliku za upotrebu. Prvo, slika se otvara koristeći Image.open funkciju iz PIL biblioteke i zatim se konvertuje u crno-belu sliku koristeći convert metodu sa parametrom "L". Slika se zatim skalira na dimenzije 28x28 piksela koristeći resize metodu sa parametrom Image.BOX.

Nakon toga, skripta prolazi kroz sve piksele slike i postavlja njihove vrednosti na 255 (belo) ako je vrednost piksela veća od 158, ili na 0 (crno) u suprotnom. Ovo se radi kako bi se slika binarizovala i dobila jasniju razliku između pozadine i rukopisa.

Zatim se slika pretvara u 1D niz vrednosti koristeći dve ugnježđene petlje. Ako je vrednost piksela 0 (crno), odgovarajući element niza se postavlja na 1, a u suprotnom na 0. Ovaj niz se zatim konvertuje u numpy niz i vraća kao rezultat funkcije.

Funkcija convert\_to\_list\_1D uzima 2D numpy niz kao ulazni parametar i konvertuje ga u 1D listu. Ovo se radi tako što se niz prvo pretvori u listu lista koristeći tolist metodu, a zatim se elementi unutrašnjih lista dodaju u rezultujuću listu.

Funkcija prep uzima putanju do slike kao ulazni parametar i poziva funkciju prepare\_real\_photo da pripremi sliku. Zatim se dobijeni niz preoblikuje u 2D niz oblika (784, 1) i množi sa parametrom param kako bi se vrednosti skalirale. Na kraju, niz se konvertuje u 1D listu koristeći funkciju convert\_to\_list\_1D i vraća kao rezultat.

Glavna funkcija skripte je prepare\_img\_for\_verilog koja ne uzima ulazne parametre. Ova funkcija poziva funkciju prep sa putanjom do slike koja je prosleđena skripti kao prvi argument komandne linije (sys.argv[1]). Zatim prolazi kroz dobijenu listu vrednosti i ispisuje ih u formatu koji je pogodan za upotrebu u Verilog programu kao inicijalizaciju registra.

A screen shot of a computer program

Description automatically generated

Slika

Ovim smo uspesno pripremili sliku za upotrebu u Verilog jeziku/programu. Pre nego sto implementiramo nasu neuronsku mrezu, provericemo njene peformanse na jednom od test primera. Aktiviramo neuronsku mrezu na nacin koji smo prethodno pomenuli u sekciji “Treniranje neuronske mreze”. (slika 5)

Testiranje u pythonu odradjeno je u pretposlednjoj celiji project.ipynb python notebook skripte koju smo ranije pomenuli.

Nekoliko napomeni:

Sigmoidna funkcija vraca 0 za svaku vrednost manju od -6 I 1 za svaku vrednost vecu od 37. Vrednosti su zatim pomnozene sa 100 kako bi rad sa celobrojnim promenljivama bio omogucen.

A screen shot of a computer

Description automatically generated

Slika

Na kraju, mozemo videti da je nasa neuronska mreza uspesno klasifikovala cifru koju smo joj zadali. (slika 6)

A black and white image of a number

Description automatically generated

Slika

Kada smo se uverili u performanse nase neuronske mreze, mozemo izgenerisati Verilog kod koristeci prepare\_img\_for\_verilog funkciju koju smo prethodno objasnili. U nastavku je generisan Verilog kod za sliku klasifikovanu kao broj 9.(slika 7) Svaki od brojeva u nizu predstavlja jedan piksel na slici.

A screenshot of a computer code

Description automatically generated

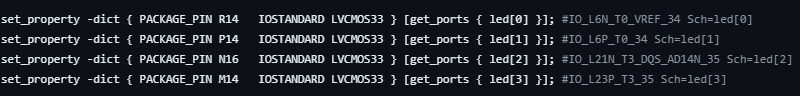
Slika 7

# Verilog implementacija neuronske mreze na FPGA ploci

## Ogranicenja(Constraints)

U direktorijumu projekta “verilog/constraints” nalazi se  Xilinx Design Constraints (XDC) fajl pod imenom “PYNQ-Z2 v1.0.xdc”. Ovaj fajl se koristi za definisanje ograničenja dizajna u Xilinx Vivado softveru. Ograničenja dizajna se koriste za specificiranje fizičkih i vremenskih karakteristika dizajna, kao što su pin lokacije, standardi ulazno/izlaznih signala i vremenska ograničenja.

U fajlu je otkomentarisan kod koji je potreban za naš konkretan slučaj. Konkretno, govorimo o 4 LED diode. Portove led[0], led[1], led[2] I led[3] mapiramo na pinove R14, P14, N16 I M14, respektivno.(slika 8)



Slika 8

## Izvor(Source)

## U direktorijumu projekta “verilog/project/gotov proij.srcs/sources\_1/new ” nalazi se “source.v”, izvorni fajl projekta  koji sadrži Verilog kod koji opisuje dizajn digitalnog sistema.

## Ovaj kod se sastoji od niza modula koji opisuju funkcionalnost i ponašanje pojedinih delova dizajna.

## Upravo unutar ovog fajla je izvrsena implementacija neuronske mreze.

Na vrhu vidimo modul source(slika 9). On nam govori da ovaj Verilog modul za ulaz ima sistemski takt, a za izlaz 4-bitni vektor koji predstavlja gore-pomenute LED diode.

A screenshot of a computer

Description automatically generated

Slika 9

U nastavku je kod(I vrednosti) koje smo generisali u python skriptama koje smo pre pominjali.(slika 10)

Registar img je niz od 784 elemenata I predstavlja pripremljenu sliku iz python skripte, gde svaki element predstavlja jedan piksel.

Vrednosti registara za pristrasnost I tezine za svaki od slojeva smo takodje generisali prilikom treniranja neuronske mreze u python-u.

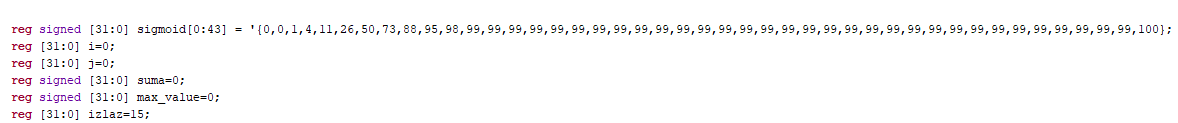
Sve vrednosti registara koji oznacavaju skriveni I output sloj su inicijalizovana nulom.

A close-up of a text

Description automatically generated

Slika 10

Registar sigmoid je niz koji sadrzi 44 elementa I predstavlja “look up” tablicu.(slika 11) Prisetimo se da smo prilikom treniranja neuronske mreze zakljucili da sigmoidna funkcija vraca 0 za vrednosti ispod -6, I 1(ili nakon mnozenja, 100) za vrednosti iznad 37. Ovaj registar predstavlja vrednosti sigmoidne funkcije za brojeve izmedju -6 I 37, redno. Ove vrednosti su takodje prethodno izracunate. Ostale registre koji predstavljaju sume, brojace I ostale vrednosti relevantne za neuronsku mrezu, takodje smo inicijalizovali nulom, dok je izlaz namesten na vrednost 15(sve LED diode svetle).



Slika 11

Zatim, u bloku koda koji se aktivira svakim uzlaznim signalom sistemskog takta, prvo racunamo vrednosti skrivenog sloja.(slika 12)

Sumu za svaki od neurona skrivenog sloja racunamo kao zbir proizvoda svih tezina(weightova) I vrednosti ulaznih neurona nase pripremljene slike. Na kraju dodamo odgovarajucu pristrasnost(bias) I cuvamo vrednost.

A white screen with black text

Description automatically generated

Slika 12

Nakon odredjivanja vrednosti neurona skrivenog sloja, primenjujemo sigmoid aktivacione funkcije, po principu koji smo prethodno objasnili.(slika 13) Izmenjene vrednosti cuvamo za svaku vrednost neurona skrivenog sloja.

A computer code with black text

Description automatically generated

Slika 13

Vreme je da izracunamo izlaze, I njih racunamo po istom principu kao kod racunanja vrednosti skrivenog sloja. Vazno je uociti da koristimo odgovarajuce tezine I pristrasnosti.(slika 14)

A white background with black text

Description automatically generated

Slika 14

Po istom principu, takodje, primenjujemo sigmoid aktivacionu funkciju izlazima.(slika 15)

A computer code with black text

Description automatically generated

Slika 15

I sve sto je ostalo, je da nadjemo indeks izlaza(cifru) koji ima najvecu verovatnocu. To radimo jednostavnom for petljom(slika 16). Cifru sa najvecom verovatnocom pamtimo, I upravo je to cifra koju cemo binarno predstaviti na nasoj ploci.

A screenshot of a computer program

Description automatically generated

Slika 16

# Testiranje neuronske mreze na FPGA ploci

U nastavku cemo testirati nasu neuronsku mrezu. Podrazumevamo da je ploca vec povezana na odgovarajuci nacin. Imamo sav potreban kod, samo je potrebno da ga pokrenemo I to radimo na sledeci nacin(slika 17): Pokrecemo sintezu, a zatim implementaciju.

A screenshot of a computer

Description automatically generated

Zatim generisemo Bitstream, idemo klikom na “Open Hardware Manager” zatim “Open Target ” I “Auto Connect ” (slika 18). Ovo ce nam otvoriti Hardware manager, I opcija “Program Device” trebalo bi postati vidljiva(Slika 19).

A screenshot of a computer

Description automatically generated

Slika 18

A screenshot of a computer

Description automatically generated

Slika 19

I klikom na “Program Device”, odabirom ploce I klikom na dugme “Program”, nasa ploca bi trebalo biti isprogramirana(slika 20), a binarna reprezentacija klasifikovane cifre, vidljiva 😉.(slika 22 I slika 21 za svrhe uporedjivanja)

A screenshot of a computer

Description automatically generated

Slika 20

A black and white pixelated number

Description automatically generated

Slika 21

A close up of a circuit board

Description automatically generated

Slika 22