**SVEUČILIŠTE U ZAGREBU**

**PRIRODOSLOVNO-MATEMATIČKI FAKULTET**

**MATEMATIČKI ODSJEK**

Daniel Jelušić

Andrea Perčinlić

Jure Šiljeg

Validacija 4 algoritma o rukom pisanim brojevima

Programerska dokumentacija

Zagreb, veljača 2014.

Konvolucijske neuronske mreže:

Cijeli program je opisan u funkciji cnnTrain koja sadrži sljedeće komponenete :

**0. korak:**

Ovdje inicijaliziramo neke parametre koji su korišteni pri gradnji programa:

imageDim - odnosi se na dimenziju slike (28x28)

numClasses - broj klasa u koju svrstavamo znamenke (10 njih)

filterDim - dimenzija konvolucijske maske

numFilters - broj maski za konvolucijski sloj

poolDim - dimenzija bazena

Potom povučemo podatke iz MNIST-ove baze, napravimo odgovarajući reshape slika da postanu operativne, spremimo ih u images, kategoriziramo oznaku 0 kao 10 i proslijedimo gore navedene parametre u skriptu cnnInitParams.

**1. korak :**

Ovdje želimo implementirati cost funkciju i pronaći način kako računamo gradijente. Sve to je opisano u cnnCost funkciji.

**2. korak :**

Želimo istrenirati naš model da ga možemo potom testirati. Uzet ćemo da će se ponavljati sve kroz 3 epohe, uz , momentum = 0.95 i minibatch = 256. Pokušavamo naći optimalan theta (opttheta) gdje nam je potrebna implementacija funkcije minFunctSGD.

**3. korak :**

Konačno, testiranje. Ovaj korak izvršavamo tako da testiramo iz MNIST-ove baze podataka 10k slika. Najprije ih učitamo i odznačimo nultu kategoriju tako da joj pridružimo labelu 10. Za kraj treba samo pokrenuti funkciju cnnCost koja prima ulazne parametre koje smo do sada računali i relevantni su za naše rješenje i to su, redom: opttheta, testImages, testLabels, numClasses, filterDim, numFilters, poolDim, true. Za kraj je još dovoljno ispisati postotak uspješnosti.

Koristili smo sljedeće funkcije:

*cnnConvole\_1 :*

Prima kao argumente sljedeće parametre:

filterDim - dimenzija konvolucijske maske

numFilters - broj maski za konvolucijski sloj

images - velike slike s kojima radimo konvoluciju

W,b - matrice koje služe za konvoluiranje sa slikama, kao neke maske

Vraća:

convolvedFeatures - matrica konvoluiranih vrijednosti slika i maski

Radi:

Izvuče iz images informacije o broju slika (numImages) i dimenziji (imageDim), te izračuna dimenzije nove, konvoluirane slike (convDim). Sama funkcija konvoluira svaku masku sa svakom slikom (54. linija koda gdje to radi MATLAB-ova funkcija conv2), doda pomak i računa sigmoidu u tim novonastalim vrijednostima (convolvedImage). Kod računanja filtera koristimo funkciju rot90 (jer conv2 rotira matricu pa je trebamo vratiti u početno stanje).

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*cnnPool\_1* *:*

Prima kao argumente sljedeće parametre:

poolDim - dimenzije bazena

convolvedFeatures - povratna vrijednost funkcije *cnnConvolve\_1*

Vraća:

pooledFeatures - matrica koja sadrži konvoluirane i downsamplane (ukloni određene retke/stupce) slike

Radi:

Smanjuje veličinu konvoluiranih slika kako bi se ubrzao cijeli algoritam.

Radit će se konvolucije svih slika sa srednjim vrijednostima matrice ones kojoj je dužina/širina jednaka poolDim (najvažnija je linija 41. gdje se opet javlja MATLAB-ova funkcija conv2).

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*cnnInitParams* :

Prima kao argumente sljedeće parametre:

imageDim - visina/širina slike

numClasses - broj klasa u koju svrstavamo znamenke (10 njih)

filterDim - dimenzija konvolucijske maske

numFilters - broj maski za konvolucijski sloj

poolDim - dimenzija bazena

Vraća : theta - vektor s inicijaliziranim vrijednostima Wc, Wd, bc, bd

Radi : inicijalizira težine Wc i Wd (te respektivne pomake bc i bd) na slučajno odabrane vrijednosti obzirom na proporcije ulaznih parametara.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*cnnCost :*

Prima kao argumente sljedeće parametre:

theta, numClasses, filterDim, numFilters, poolDim

images - pohranjene slike iz MNIST-ove baze

niz gdje je numImages ukupan broj slika

pred - boolean vrijednost koja služi za propagaciju unaprijed i povrat predviđanja

Vraća:

cost - cross entropy cost

grad - gradient with respect to theta (if pred==False)

preds - lista predviđanja za svaki primjer (if pred==True)

Radi:

U 5 dijelova. Najprije, u prvom dijelu, iščita podatke iz images i spremi visinu/širinu u imageDim, a broj slika spremi u numImages. Potom se treba napraviti reshape parametara Wc, Wd, bc i bd (to napravi funkcija cnnparamsToStack), te inicijalizirati njihove odgovarajuće gradijentne matrice.

Drugi dio je propagacija unaprijed. U početku se propagira ulaz kroz konvolucijski i pooling sloj, pa se potom ti podaci koriste kao ulaz u drugi (pravi) softmax sloj. U konvolucijskom sloju se događa konvolucija između svake maske (Wc) i svake slike, pribraja se pomak i primijeni se sigmoidna funkcija na dobivene podatke (funkcija cnnConvolve\_1(filterDim, numFilters, images, Wc, bc)) i pohrani se rezultat konvolucija u activations. Potom se uzorkuju konvoluirane aktivacije sa središnjim bazenom (funkcija cnnPool\_1(poolDim, activations)), a rezultat toga se spremi u activationsPooled. Također se tu izračuna i dimenzija konvoluiranog izlaza koji ćemo koristiti u activations (convDim), kao i dimenzija uzorkovanog izlaza koji se koristi u activationsPooled (outputDim). Na koncu je potrebno napraviti resize od activationsPooled kojeg ćemo koristiti u sljedećem softmax sloju. U njemu se propagiraju vrijednosti izračunate u activationsPooled u standardni softmax sloj, a same rezultate spremamo u probs. Potom izračunamo izlaz softmax sloja tako da pomnožimo težine WD i activationsPooled i proširimo s repmat(bd,1,M) gdje je bd pomak, a M=size(images, 3), oduzmemo po točkama maksimalnu vrijednost, te po točkama opet ubacimo umjesto npr. vrijednosti vrijednost od Za kraj još normaliziramo vrijednosti od aux3 i spremimo sve u probs. Spremni smo za posljednje korake gdje računamo cijenu koju ćemo koristiti u softmax regresiji. U ovom koraku koristimo labele koje smo dobili kao ulazne podatke zajedno s probs za izračun cost funkcije. Nakon toga se naprave predviđanja obzirom na dane probs i vraćamo se natrag backpropagation algoritmom kako bi izračunali gradijente za SGD. U tu svrhu rabimo formulu[[1]](#footnote-1) koja nam daje način kako izračunati Wd\_grad. Nakon toga analogno računamo po formuli[[2]](#footnote-2) i deriv\_2\_pooled\_sh. Ostali podaci se dobiju analogno koristeći upravo spomenutu formulu. Propagacija unatrag u softmax sloju se lako uradi[[3]](#footnote-3), potom se greška treba propagirati kroz *pooling* sloj gdje trebamo upsampleati grješku u odnosu na *pooling* sloj obzirom na svaku sliku i svaku masku (koristimo *kron[[4]](#footnote-4)* funkciju).

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*minFunctSGD :*

Prima kao argumente sljedeće parametre:

funObj - funkcijski hande koji prima kao parametre thetu, data, labels i vraća cost i gradijent obzirom na thetu

theta

data - pohranjuje podatke u m x n x numExamples tenzor

labels - odgovarajuće oznake u numExamples x 1 vektoru

options - struktura za spremanje specifičnih opcija potrebnih za optimizaciju:

1) epochs - broj epoha kroz koje testiramo podatke

2) alpha - inicijalni koeficjent učenja

3) minibatch - veličina minibatcha

4) momentum - konstanta (po defaultu 0.9)

Vraća:

opttheta - optimirani vektor theta

Radi:

Za početak u m spremi veličinu trening skupa. Početne uvjete i postavke za momentum postavimo u varijable mom=0.5, momIncrease=20. Sljedeće što treba uraditi jest implementirati SGD (Stohastic Gradient Descent - stohastički gradijentni spust). Tu, također, koristimo kaljenje koeficjenta učenja (da dobijemo bolju konvergenciju prilikom kada učenje počinje usporavati (na kraju svake epohe). Isto tako ako slučajno permutiramo podatke prilikom ulaza u svaku epohu, eksperimentalno se može utvrditi da ćemo dobivati bolje rezultate. Kad dobivamo vrijednosti varijable velocity, želimo pomoću nje optimirati naš theta i to tako da grad pokazuje nagib negdje u okolini otpimuma i radi slično kao i Newtonova metoda bisekcije.

Višeslojni perceptron:

Raspoznavanje rukom pisanih brojeva pomoću višeslojnog perceptrona MLP (*MultiLayer* *Perceptron*)

Opis zadatka:

Opisati programsku izvedbu MLP – a za raspoznavanje rukom pisanih brojeva.

Programska izvedba zadatka realizirana je u MATLAB – u.

*model* je struktura koja predstavlja MLP, a sastoji se od brzine učenja *learning\_rate*, momenta *momentum*, matrica težina *weights{i}*, matrica pragova *biases{i}* i matrica *lastdelta{i}* koje služe za korekciju težina. Učenje MLP – a ostvareno je pomoću funkcije *train\_mlpNN*, a validacija tako trenirane mreže provedena je pomoću funkcije *test\_mlpNN*. Učenje mreže i korekcija težina vrši se pomoću *update\_mlpNN*.

U nastavku je dan detaljniji opis navedenih funkcija.

***train\_mlpNN***:

*train\_mlpNN* inicijalizira *model*, mreži predstavlja uzorke iz skupa za učenje određeni broj puta (*iterations*) te vraća istrenirani *model* i točnost raspoznavanja ispravno klasificiranih uzoraka za učenje *cc*. Ulazni argumenti funkcije su skup za učenje *input*, ciljni odziv mreže *target*, skriveni slojevi mreže *hidden*, broj epoha *iterations*, brzina učenja *learning\_rate* i moment *momentum*.

*input* je matrica koja ima R redaka, gdje je R broj uzoraka u skupu za učenje, i C stupaca, gdje je C broj neurona u ulaznom sloju mreže (784). Dakle, svakom uzorku iz skupa za učenje pridružen je točno jedan redak matrice *input*.

*target* je matrica koja ima R redaka, gdje je R broj uzoraka u skupu za učenje, i S stupaca, gdje je S broj neurona u izlaznom sloju mreže (10). *target(i, :)* je željeni odziv mreže za i – ti uzorak iz skupa za učenje. Na primjer, ako je i – ti uzorak iz skupa za učenje klasificiran kao znamenka j, tada je *target(i, :)* vektor – redak koji se sastoji od 1 jedinice na mjestu *j + 1* i 9 nula na preostalim mjestima.

*hidden* je vektor – redak koji predstavlja skrivene slojeve u mreži. Na primjer, *hidden* = [5 5] ukazuje na 2 skrivena sloja u mreži koji se sastoje od 5 neurona. U ovom zadatku *hidden* je oblika [10 0] (1 skriveni sloj s 10 neurona).

Implementacija funkcije *train\_mlpNN* detaljno je komentirana u datoteci u privitku *train\_mlpNN.m*.

***update\_mlpNN***:

*update\_mlpNN* je funkcija koju (*iterations* ∙ *broj uzoraka u skupu za učenje*) puta poziva funkcija *train\_mlpNN*. Ulazni argumenti funkcije su *model*, *input* i *target* koji su već prije opisani. Prvo se propagira funkcijski signal neurona u svakom sloju unaprijed kroz mrežu, zatim se signal greške definira na izlazu mreže kao razlika ciljnog i dobivenog odziva neurona te naposljetku, algoritmom učenja propagacije greške unatrag kroz mrežu korigiraju se težine i pragovi ulaznog *modela*. Funkcija vraća istrenirani *model*.

Detalji korekcija težina i pragova na osnovu kojih je implementirana funkcija jasno su objašnjeni u radu navedenom u korisničkoj dokumentaciji u korištenoj literaturi (3).

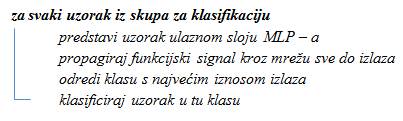
Implementacija funkcije *update\_mlpNN* detaljno je komentirana u datoteci u privitku *update\_mlpNN.m*.

***test\_mlpNN***:

*test\_mlpNN* obavlja postupak klasifikacije rukom pisanih brojeva iz ispitne baze podataka. Ulazni argumenti funkcije su istrenirani *model*, skup uzoraka za ispitivanje *input* te pripadni ciljni odziv mreže *target*.

*input* je matrica koja ima P redaka, gdje je P broj uzoraka u skupu za ispitivanje, i C stupaca, gdje je C broj neurona u ulaznom sloju mreže (784). *target* je matrica koja ima P redaka, gdje je P broj uzoraka u skupu za ispitivanje, i S stupaca, gdje je S broj neurona u izlaznom sloju mreže (10). *input* i *target* imaju ista svojstva kao i već opisani *input* i *target* (razlikuju se jedino u broju redaka matrica)*.*

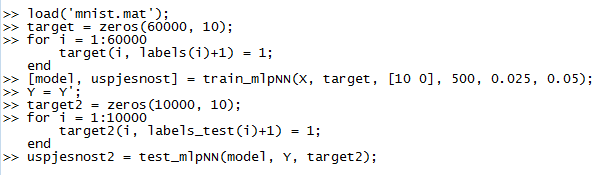
Funkciju test\_mlpNN najjednostavnije je opisati pseudokodom u nastavku:

**

Povratna vrijednost funkcije je točnost raspoznavanja ispravno klasificiranih uzoraka za ispitivanje *cc.*

Implementacija funkcije *test\_mlpNN* detaljno je komentirana u datoteci u privitku *test\_mlpNN.m*.

Nakon što su opisane sve korištene funkcije, u nastavku slijedi primjer jednostavne upute za pokretanje postupka raspoznavanja rukom pisanih brojeva pomoću MLP – a u MATLAB – u.



Napomena:

Prilikom pokretanja bitno je osigurati da su sve pozivajuće funkcije i datoteka *mnist.mat* iz *MLP.rar*.

SVD dekompozicija:

Cijeli program se sastoji od jedne MATLAB skripte po imenu svdscript. Za početak je potrebno povući sve podatke iz MNIST-ove baze podataka s informacijama o rukom pisanim znamenkama i to radi prva linija u skripti. Nakon toga je potrebno bilo transponirati matricu X da se dimenzije poklapaju onako kako želimo. Potom izdvajamo skupove T{i} i vršimo SVD dekompoziciju istih da dobijemo singularne vektore od T{i} koje potom respektivno nazovemo U{i}. Varijabla m iz 11. linije je opcionalna i testirana je na nekoliko primjera (m=10, m=30 i m=50). Potom je bilo potrebno klasificirati testne znamenke (njih 10k) i to tako da izračunamo reziduale i potom vršimo usporedbu s pravim oznakama za znamenke i mjerimo postotak uspješnosti.

Naivni Bayesovski klasifikator:

Priložene su datoteke mnist.mat i bayes.m.

Sama implementacija je dosta jednostavna. U početku je potrebno pretvoriti sve numeričke varijable u logičke (da imamo samo 0 i 1).

U prvom dijelu se provodi trening, koji se zapravo svodi na izračunavanje potrebnih aproksimacija vjerojatnosti.

Kod klasifikacije, važno je napomenuti da vjerojatnosti logaritmiramo kako bismo unaprijedili numeričku stabilnost. Naime, uzastopnim množenjem vjerojatnosti često se dobivaju vrlo maleni brojevi koje MATLAB može interpretirati kao 0. Posljednji dio koda (velika for petlja) se odnosi na provjeravanje uspješnosti.

1. http://ufldl.stanford.edu/wiki/index.php/Softmax\_Regression [↑](#footnote-ref-1)
2. Ista se može pročitati ovdje : http://ufldl.stanford.edu/wiki/index.php/Backpropagation\_Algorithm [↑](#footnote-ref-2)
3. Pogledati za pojedinosti na ovom linku : http://ufldl.stanford.edu/wiki/index.php/Backpropagation\_Algorithm [↑](#footnote-ref-3)
4. Više se može pročitati na sljedećem linku : http://ufldl.stanford.edu/tutorial/index.php/Exercise:\_Convolutional\_Neural\_Network#Implement\_Convolutional\_Neural\_Network [↑](#footnote-ref-4)