Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Nac Stoklas

Globoke nevronske mreže v medicinski diagnostiki

Globoke nevronske mreže v medicinski diagnostiki

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor: izr. prof. dr. Matjaž Kukar

Ljubljana, 2017

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge: Globoke nevronske mreže v medicinski diagnostiki

Preučite področje uporabe strojnega učenja v medicinski diagnostiki s poudarkom na nevronskih mrežah. Oglejte si klasične primere uporabe in v zadnjih letih izjemno populare globoke nevronske mreže, predvsem na slikovnih preiskavah. Osredotočite se na uporabo globokih nevronskih mrež na s posplošenimi atributi opisanih problemih in implementirajte več različic v programskem okolju Python. Implementirane metode ovrednotite na zahtevnem velikem

Zahvaljujem se svojemu mentorju, družini in prijateljem za nenehno podporo, vodstvo in vzpodbudo.

Kazalo

Seznam uporabljenih kratic in tujk

kratica	angleško	slovensko
ReLu	Rectified linear unit	Aktivacijska funkcija s popravljeno linearno enoto
Softmax	Softmax function	Softmax funkcija (posplošitev logistične funkcije)
NN	Neural network	Nevronska mreža
CNN	Convolutional neural network	Konvolucijska nevronska mreža
Dense	Dense layer	Gosto povezana plast
MaxPool	Max pooling	Maksimalno združevanje
Conv2D	Convolutional layer	Konvolucijska plast
cuDNN	CUDA Deep Neural Network library	Knjižnica CUDA za delo z globokimi nevronskimi mrežami
CPU	Central processing unit	Centralna procesna enota
GPU	Graphics processing unit	Grafična procesna enota
CUDA	Compute Unified Device Architecture	NVIDIA arhitektura za visoko performančne aplikacije na GPU
BLAS	Basic Linear Algebra Subprograms	Knjižnica za osnovne operacije linearne algebre

Povzetek

Naslov: Globoke nevronske mreže v medicinski diagnostiki

Avtor: Nac Stoklas

V zadnjih letih količina medicinskih podatkov ter preiskav strmo narašča. S tem se področju strojnega učenja odpira še več možnosti za raziskovalno delo, ki je v medicini prisotno že dlje časa. Tradicionalni pristopi k analizi podatkov so se izkazali za učinkovite, zadnje čase pa lahko opazimo porast v uporabi globokih nevronskih in natančneje konvolucijskih nevronskih mrež. Te so uporabne predvsem za slikovne podatke, začenjajo pa se uporabljati tudi na klasičnih atributnih podatkih. V našem primeru smo skušali zgraditi napovedni model s pomočjo konvolucijskih nevronskih mrež, ki bi za vhod prejel navadne atributne podatke. Uporabili smo podatkovni set, ki ima veliko različnih atributov (rezultati preiskav), veliko razredov (različne bolezni) in mnogo nedefiniranih vrednosti (na različnih bolnikih se opravijo različne preiskave, zato je v podatkih mnogo praznih mest). Na teh realnih podatkih smo preizkusili različne zgradbe plitkih in globokih nevronskih mrež. Izkazalo se je, da so dobljeni rezultati primerljivi z najboljšimi doslej pridobljenimi na istih podatkih.

Ključne besede: globoke nevronske mreže, medicinska diagnostika, strojno učenje.

Abstract

Title: Deep neural networks in medical diagnostics

Author: Nac Stoklas

In recent years the amount of medical data and research procedures is rapidly increasing. With it there are even more possibilities for machine learning research, which has been present in the medical fields for a long while. Traditional approaches to data analysis have proven to be useful. The same goes for neural networks and more specifically for convolutional neural networks, which have seen heavily increased use in the latest years. They are mostly used on image data, although lately we have seen examples of them being used on conventional attribute data. We try to build a model based on CNN which would receive regular attribute data as input. We use a data set, which has many attributes (results), many classes (different diseases) and many undefined values (different patients require different procedures, which results in many undefined test values). It is on this real life data that we test different structures of deep and shallow CNN's. We conclude that results obtained in this manner are comparable to the best results on this data so far.

Keywords: deep neural networks, medicinal diagnostics, machine learning.

Poglavje 1

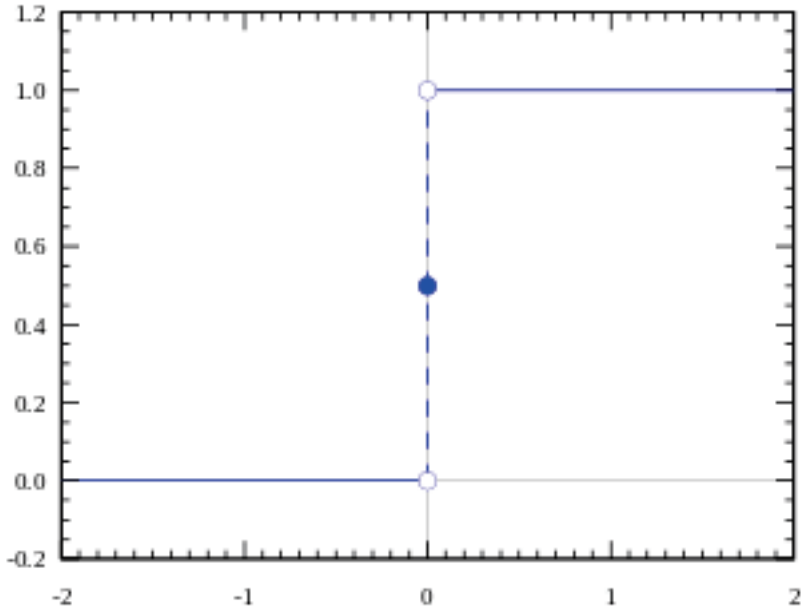
Uvod

Umetna inteligenca je del računalništva, ki poskuša narediti računalnike (vsaj na videz) bolj inteligentne. Ena izmed osnovnih zahtev za inteligenco je učenje. Večina raziskovalcev se strinja, da ni inteligence brez učenja. Tako lahko sklepamo, da je strojno učenje eno izmed najpomembnejših področij umetne inteligence - ki se tudi najhitreje razvija. Že od začetka se uporablja algoritme strojnega učenja za analizo medicinskih podatkov. Zdravstvene ustanove hranijo vedno večje količine podatkov in z njimi se razvija vedno več naprednih metod strojnega učenja [15]. Eden izmed pristopov, ki je v zadnjih letih v ospredju, so nevronske mreže. Najpreprostejšo definicijo je podal izumitelj enega izmed prvih računalnikov za delo z nevronskimi mrežami, Dr. Robert Hecht-Nielsen [5].
Nevronske mreže so računski sistem, ki jih sestavlja veliko preprostih, visoko prepletenih elementov za procesiranje, katerih stanje je dinamično odvisno od njihovega odziva na zunanji vhod.

Elementi so prepleteni s pomočjo uteženih povezav. Vsak izmed preprostih elementov izračuna uteženo vsoto vseh vhodov vanj ter prišteje pristranskost. Na ta način dobi element (ali nevron) Y (enačba 1.1) stanje iz zgornje definicije.

$$Y = \sum(utezi \times vhodi) + pristranskost \tag{1.1}$$

Nevron se nato glede na aktivacijsko funkcijo odloči, ali se aktivira in posreduje informacijo naslednji plasti. Poznamo mnogo različnih aktivacijskih funkcij. Za ilustracijo (slika 1.1) si lahko zamislimo takšno, ki aktivira nevron, ko ima vrednost večjo od neke arbitrarne vrednosti, npr. 0. Nasprotno se element ne aktivira, če je njegova vrednost manjša od 0.



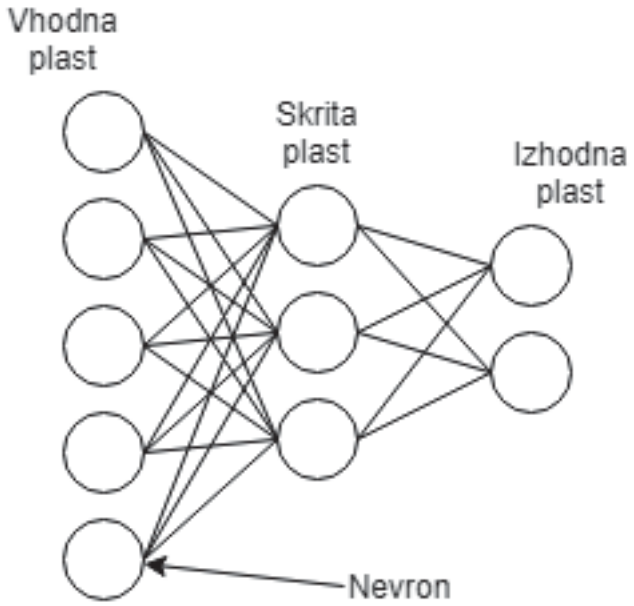
Slika 1.1: Primer preproste aktivacijske funkcije.

Takšno aktivacijsko funkcijo, ki vrača le 1 ali 0, imenujemo binarna aktivacijska funkcija. Primerne so za probleme, kjer imamo le dve možnosti. V tem diplomskem delu problem ne bo binarne narave. Zato se srečamo z aktivacijskimi funkcijami, ki lahko vrnejo več kot dve različni vrednosti. Uporabljali bomo softmax (posplošitev logistične funkcije) ter relu [2] (enačba 1.2). Slednja vrne 0, če je vrednost nevrona Y manjša od 0. V tem oziru je enaka ilustrativni, binarni, funkciji. V nasprotnem primeru vrne kar njegovo vrednost, ki je lahko karkoli večjega od 0.

$$F(x) = max(x, 0) \tag{1.2}$$

Povezave med elementi nevronske mreže niso naključne. Organizirane so hierarhično tako, da nevroni tvorijo plasti. Poznamo tri večje tipe plasti. Vhodno plast, ki se ji poda osnovna informacija (npr. atributi problema). Sledi poljubno število skritih plasti, povezanih z uteženimi povezavami, ki nazadnje posredujejo informacijo vse do zadnje, izhodne plasti.

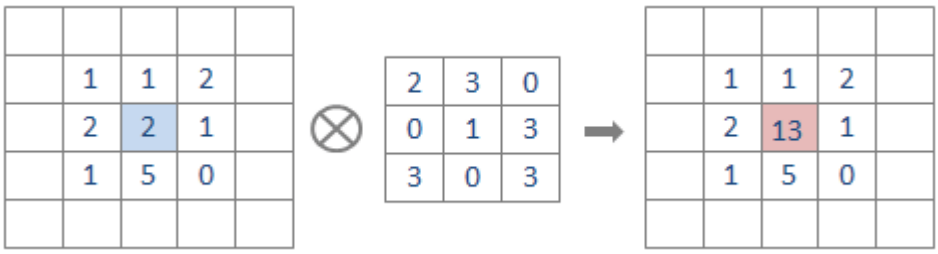
Izpostavili smo, da je ena izmed osnovnih zahtev za inteligenco učenje. Velik del učenja pa predstavlja ponavljanje, kot tudi iterativno popravljanje napak. Nevronske mreže ga implementirajo s t.i. procesom vzratnega razširjanja napak. Ko informacija pripotuje do izhodne plasti, to omogoča utežem, da se za nazaj nekoliko spremenijo (popravijo) tako, da bo v naslednji iteraciji izhod nekoliko bližje dejanski rešitvi problema. Za boljšo predstavo nevronske mreže je spodaj shema nevronov ter njihovih povezav (slika 1.2).



Slika 1.2: Shema nevronov ter povezav med njimi.

Poznamo več vrst nevronskih mrež. Poseben primer nevronske mreže je tudi konvolucijska nevronska mreža (CNN). Glavna sprememba je ta, da so plasti v takšni mreži sestavljene predvsem iz kombinacije konvolucijskih plasti (opravijo konvolucijo na prejetih podatkih), ter plasti, ki zmanjšajo dimenzije vhoda (maksimalno združevanje). Konvolucijske plasti (po katerih dobi ime tudi mreža) opravljajo matematično operacijo konvolucije. To je operacija dveh funkcij, ki vrne tretjo, nekoliko modificirano verzijo prve, originalne funkcije. Na naslednji strani predstavimo preprost primer konvolucije.

Na sliki 1.3 je primer konvolucije v dveh dimenzijah. Najprej pomnožimo vsako število znotraj originalne matrike (leva matrika) s pripadajočim številom v filtru (sredinska matrika). Nato ta števila seštejemo. Izhod (desna matrika) je originalna matrika spremenjena za element v centru filtra, ki je opisan seštevek in zmnožek med konvolucijo. Opisali smo le en korak celotnega procesa konvolucije. Filter se premika po celotni matriki, dokler ga nismo postavili na vsa možna mesta.



Slika 1.3: Shema konvolucije na eni poziciji.

Konvolucijske nevronske mreže omenjamo, ker so zadnja leta prodrle na vsa področja znanosti. Med drugim tudi v medicino. Izkazale so se za posebej uporabne pri analizi slikovnega materiala, ki ga je tukaj ogromno. Zadnja leta se analizira slike magnetne resonance (Bhattacharyya [3], Demirhan [9], Desmukh [17]), slike dojk za prepoznavanje rakavih metastaz (Wang [22], Cire [10], Cruz-Roa [11]), računalniške tomografije za pomoč pri naprimer. detekciji pljučnih obolenj (Xingjian [23], Ginneken [21]) ipd.

Vse omenjene raziskave imajo dobre rezultate, napovedni modeli dosegajo točnosti tudi nad 90 odstotki. Za te specifične primere bi lahko izboljšali preventivno oskrbo. Težava pa je, da imajo omenjene raziskave relativno majhno število razredov ter malo nedefiniranih vrednosti v podatkih. V medicinski diagnostiki pa je stvari, o katerih smo prepričani, zelo malo.

Ob obisku zdravstvene ordinacije zdravnik zapiše le nekaj simptomov. Standardne preiskave ravno tako pregledajo le omejen spekter atributov. Preprosto ni pragmatično, da bi v vsakem primeru opravili na stotine specifičnih analiz. V mnogih tudi ni potrebno, saj že osnovne preiskave pokažejo na bolezen. Če bi želeli, tako kot zdravnik, diagnosticirati s pomočjo računalnika, bi se morali torej boriti z velikim številom nedefiniranih vrednosti v podatkih. Tudi število možnih bolezni (in s tem razredov) je veliko večje, kot pri naprimer prepoznavanju rakavih metastaz (so/niso). Takšni podatki, ki so pisani na kožo CNN, niso slikovni, ampak so preprosti vektorji s po več sto atributi, ki pripadajo več sto razredom.

Takšni so podatki, s katerimi bomo delali v tem diplomskem delu in jih tudi natančno predstavili v naslednjem poglavju. Zaradi moči in popularnosti konvolucijskih nevronskih mrež bomo preizkusili njihovo delovanje tudi na podatkih, ki niso tipično primerni zanje. Preizkusili bomo tudi nevronske mreže in primerjali njihovo učinkovitost. Ker je podatkov veliko in imajo mnogo nedefiniranih vrednosti, bomo posamezne primere, vektorje, pretvorili v matrike. Tako se ohrani več informacije o lokalnosti in se jo celo izpostavi. Določene sorodne preiskave so v podatkih blizu skupaj in želimo si, da bi pretvorba te otočke informacij izpostavila, zaradi česar bi se jih mreža bolje

naučila. Uporabimo lahko iste algoritme, kot se uporabljajo pri analizi slik. Pri tem si pomagamo z visokonivojskim APIjem Keras. Njegovo namestitev in osnovne zakonitosti predstavimo v tretjem poglavju. Temu sledi poglavje metodologij, kjer opišemo strukturo mrež, ki smo jih sestavili, predstavimo pa tudi opravljene transformacije vektorjev v matrike. Temu sledi predstavitev rezultatov. Poglavje predstavi in primerja uspešnost različnih mrež, jih postavi ob bok rezultatom kakšne bolj klasičnih metod strojnega učenja ter primerja časovne zahtevnosti različnih pristopov. Sledi še sklep, v katerem se vrnemo na začetek in ugotovimo ali so konvolucijske nevronske mreže primerne za tak tip podatkov ter predlagamo izboljšave in smernice za nadaljnje delo.

Poglavje 2

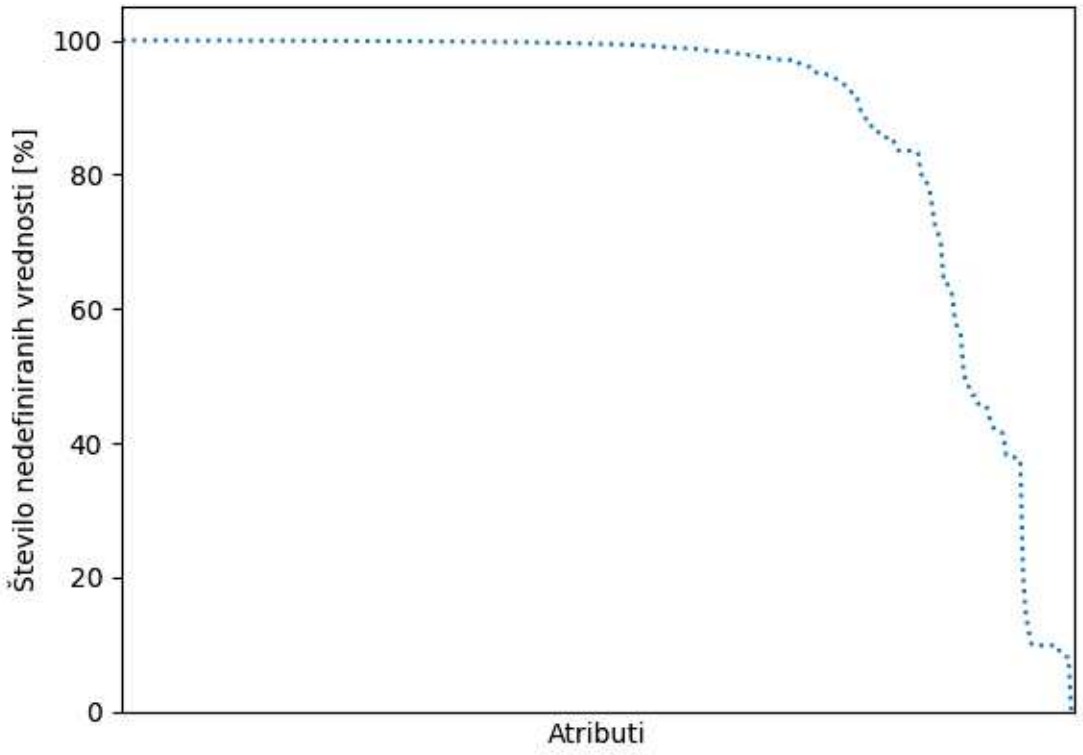
Podatki

Podatki, ki smo jih uporabljali za testiranje, so bili pridobljeni od ene izmed večjih evropskih bolnišnic pod strogo izjavo o nerazkritju (NDA). Podatki so popolnoma anonimizirani, tudi imena preiskav in diagnoz so predstavljena šifrirano. Vse vrednosti so skalirane na interval [-1,+1], kjer -1 predstavlja minimalno, +1 pa maksimalno vrednost v podatkih.

Podatkovni set ima 163.953 primerov (vektorjev). Vsebujejo 400 različnih razredov, med katerimi imajo nekateri le po dva različna primera. To bi se lahko izkazalo za problematično, saj tako malo število primerov potencialno ni dovolj za kakršnokoli posploševanje. Vsak primer ima 337 atributov (rezultati preiskav, simptomi,...). Vsebujejo tudi šifrirane oznake po področjih, ki omogočajo razbitje na večje podmnožice. Izberemo lahko določeno podmnožico ter zanjo zgradimo napovedni model, kar pomaga z ocenjevanjem časovne zahtevnosti grajenja modela na vseh podatkih. Nad njimi smo izvedli različne metode predprocesiranja podatkov, kot so normalizacija, standardizacija,... Najboljše rezultate smo dobili le z uporabo normalizacije.
Podatke smo razdelili na dva dela - na atribute in razrede, ki predstavljajo vhod in izhod. Pri predprocesiranju podatkov smo poleg delitve na dve množici ter normalizacije tudi zapolnili prazna oziroma nedefinirana mesta v podatkih s povprečjem v celotnem setu podatkov. Kot alternativo bi lahko uporabil naprimer konstanto.

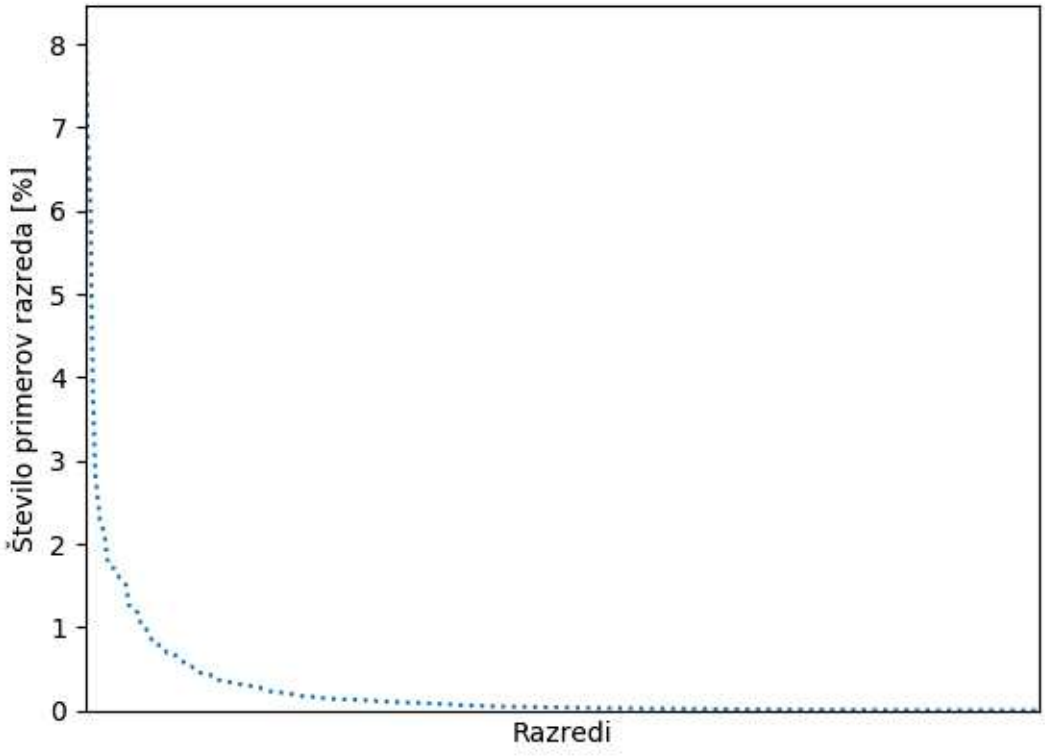
Pred vhodom podatkov v učni model smo izhode (Y) kodirali z načinom ena naenkrat (bolje poznan kot one-hot encoding). Ta metoda vsakem izhodu priredi unikaten vektor, ki ima dolžino enako številu vseh razredov. Vsebuje same ničle z izjemo enice na n-tem mestu za n-ti razred. Ta pristop je bil potreben, saj gre za klasifikacijo več razredov. Tako je zadnja plast nevronske mreže sestavljena iz 400 nevronov (en nevron predstavlja en razred). Vsak izmed teh nevronov vrne vrednost med 0 in 1, ki predstavlja verjetnost, da je posamezen razred tisti, ki pripada vhodnim atributom.

Kot smo omenili že v uvodu delamo s podatkovnim setom, ki ima veliko nedefiniranih vrednosti. Za boljšo predstavo problematike je na naslednji strani prikazan graf (slika 2.1) vseh atributov v odvisnosti s številom vseh nedefiniranih vrednosti. Atributi so urejeni po padajoči frekvenci nedefiniranih vrednosti. Jasno se vidi, da je velik del atributov v večini primerov nedefiniran. To ima velik vpliv na zahtevnost problema, saj ima nevronska mreža toliko manj specifičnih podatkov in variacij, iz katerih se lahko uči.



Slika 2.1: Število neznanih vrednosti po atributih.

Druga posebnost teh podatkov v primerjavi z zgoraj omenjenimi raziskavami, ki so uporabile CNN, je število razredov. Tukaj ne gre za klasifikacijo ali prepoznavanje specifičnega obolenja, ampak poskušamo povezati širok nabor atributov s pripadajočimi razredi. Na naslednji strani je graf (slika 2.2), ki povezuje razrede z njihoviim frekvencami. Opazimo lahko veliko število razredov, ki niso dobro zastopani. Tudi zaradi tega je pravilno klasificiranje razredov oteženo.



Slika 2.2: Število primerov po razredih.

Poglavje 3

Keras API

Za gradnjo nevronskih mrež uporabimo orodje, ki je namenjeno ravno temu - Keras. Ta visokonivojski API za nevronske mreže je spisan v Pythonu. Omogoča preprosto delo z različnimi variacijami nevronskih mrež. Ponuja preprosto implementacjo različnih topologij - ena vrstica kode za eno plast v nevronski mreži. Zato, da lahko ustvarjene nevronske mreže učimo in z njimi napovedujemo, se naslanja na knjižnice, ki so za to posebej specializirane. V zaledju zato uporablja TensorFlow, CNTK [19] ali Theano. Te knjižnice za delo z nevronskimi mrežami izdajajo in vzdržujejo različne institucije.

Keras omogoča uporabo CPUja ter GPUja. Zaenkrat podpira verzije Pythona med 2.7-3.5. Zanaša na knižnice numpy, scipy, yaml, HDF5 ter NVIDAin cuDNN (v primeru uporabe CNN). Omogoča postavitev okolja s pomočjo ukaza pip kot tudi inštalacijo iz izvorne datoteke. Za vzpostavitev zalednih knjižnic je potrebna ločena inštalacija le teh. Naslednji dve podpoglavji opisujeta postavitev TensorFlowa in Theana. Med njima menjujemo tako, da popravimo spremenljivko *backend* v datoteki keras.json na želeno knjižnico.

```
{
    "image_data_format": "channels_last",
    "epsilon": 1e-07,
    "floatx": "float32",
```

```
    "backend": "theano"
}
```

Keras omogoča tudi simultano uporabo obeh knjižnic, a se s tem v okviru tega diplomskega dela ne bomo ukvarjali.

3.1 Namestitev potrebnih orodij

Namestitev Kerasa je relativno preprosta. S pomočjo enega ukaza pip namestimo celotno knjižnico.

```
sudo pip install keras
```

V naslednjih dveh podpoglavjih opišemo namestitev zalednih knjižnic za delo z nevronskimi mrežami Theano in TensorFlow.

3.1.1 TensorFlow

Prvi korak pri namestitvi TensorFlowa je izbira verzije. Na voljo sta dve. Prva podpira le operacije na CPU, medtem ko ima druga podporo tudi za delo na GPU. Slednja je boljša, saj je grafična procesna enota zelo primerna za delo z nevronskimi mrežami in pospeši računanje. Ukaza za namestitev se razlikujeta le v končnici '-gpu'.

```
pip3 install --upgrade tensorflow
```

ali

```
pip3 install --upgrade tensorflow-gpu
```

Namestimo ga lahko tudi s pomočjo Anaconde, a za ta paket ne skrbi ekipa TensorFlowa, zato ga spremlja opozorilo, da se uporablja na lastno odgovornost. Seveda ne zadoščajo vsi računalniki minimalnim zahtevam - ki vključujejo grafično kartico NVIDIA, na katero se lahko naloži potrebna programska oprema. Na uradni spletni strani so povezave na strani, kjer so navodila za namestitev CUDA (https://developer.nvidia.com/cudnn) in cuDNN (https://developer.nvidia.com/cudnn).

V primeru nevšečnosti v času nameščanja je na uradni spletni strani tudi seznam pogostih napak.

Za uporabo GPU ni potrebno specificirati nobenih zastavic - če je zaznan, se ga uporabi avtomatsko (če je namestitev podpirala uporabo GPU).
Pri uporabi je potrebno paziti, da imamo na voljo dovolj spomina (na GPU). Če ga ob zagonu programa ni dovolj, se izvajanje avtomatsko ustavi. Samo sporočilo napake ne kaže naravnost na primanjkljaj prostora, kar bi lahko povzročalo manjše preglavice pri razhroščevanju. Z alokacijo spomina ni imel nobenih težav Theano.

3.1.2 Theano

Predpogoj za namestitev in uporabo Theana je poleg vsega, kar potrebuje Keras, tudi BLAS [4]. Ta omogoča dobro optimizirano delo z osnovnimi vektorskimi ter matričnimi operacijami. Theano ima na uradni spletni strani kar predlogo za namestitev vseh potrebnih (pa tudi nekaj dodatnih) knjižnic.

```
conda install
        numpy scipy mkl-service libpython <m2w64-toolchain>
        <nose> <nose-parameterized> <sphinx> <pydot-ng>
```

Predlagan način inštalacije je s pomočjo Anaconde, podpira pa tudi pip. Tudi tukaj sta na voljo dva tipa namestitve (CPU in CPU+GPU).

```
conda install theano pygpu
```

Uporaba GPU tukaj ni avtomatska - četudi je nameščena GPU verzija Theana. Uporabljajo se tri metode. Prva je s pomočjo zastavice 'THEANO FLAGS', kjer se specificira indeks naprave GPU (gpu0, gpu1,...).

```
import os
os.environ['THEANO_FLAGS'] = "device=gpu,floatX=float32"
```

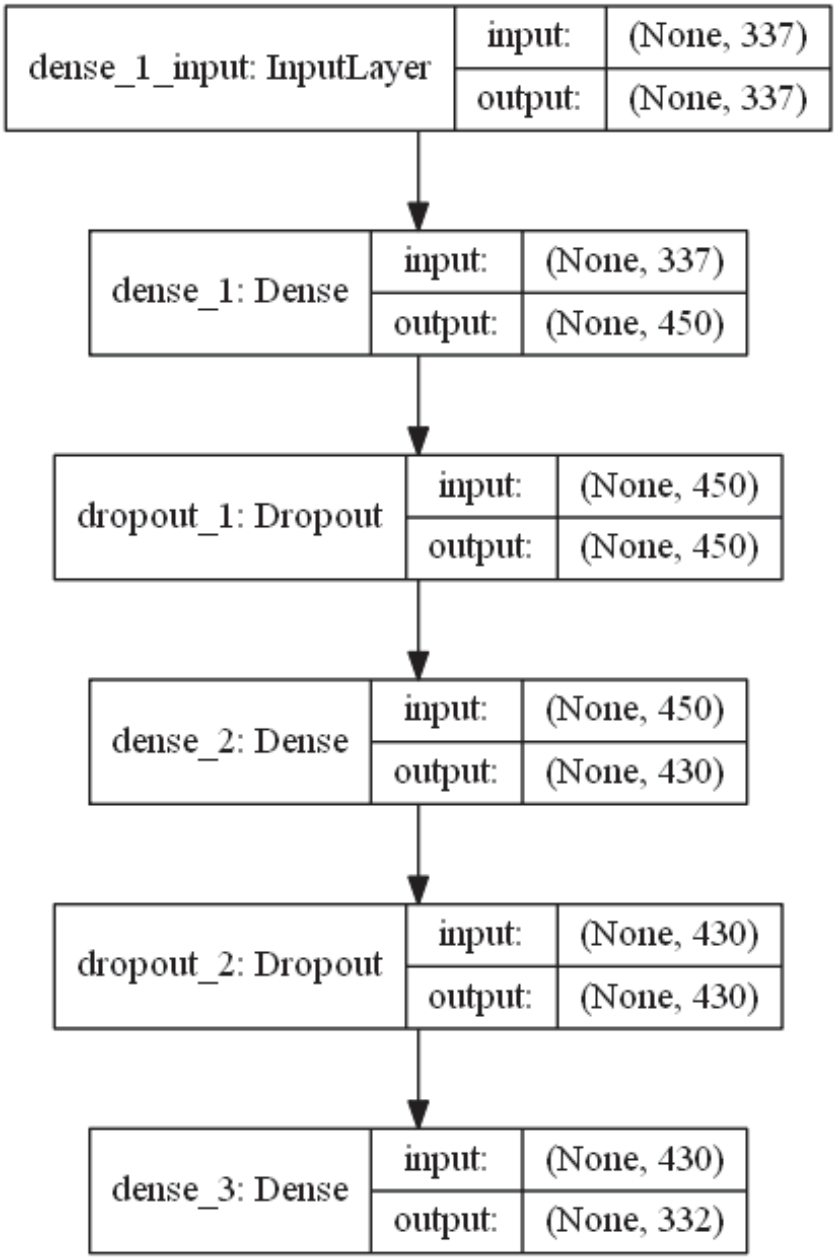
Druga možnost omogoča spremembe v datoteki *theanorc.txt* - tudi tukaj se podobno kot pri zastavicah nastavi indeks naprave. Lahko pa tudi nastavimo napravo na začetku kode (theano.config.device, theano.config.floatX).

Poglavje 4

Metode

4.1 Nevronska mreža

Preden smo sestavili konvolucijsko nevronsko mrežo, smo preizkusili natančnost napovedi nevronskih mrež. Še pred začetkom dela smo poskrbeli, da smo ne glede na topologijo vedno imeli dovolj nevronov po posameznih plasteh. Problem je relativno velik (za tak tip podatkov) in širina mreže bi lahko igrala pomembno vlogo pri prepoznavanju majhnih razlik med razredi. Vedno smo uporabili neko kombinacijo plasti *Dense* ter *Dropout*. Preizkusili smo mnogo kombinacij teh plasti - tako globoke, kot tudi široke arhitekture. Sledili smo vzgledu člankov kot je npr. Deep 'Big Multilayer Perceptrons for Digit Recognition' [7], ki opisujejo uspešnost arhitektur ki vsebujejo veliko skritih plasti. Podoben trend je viden med konvolucijskimi nevronskimi mrežami, a se jih bomo podrobneje dotaknili v poglavju o konvoluciji. Za naš primer se je izkazalo, da niso optimalne ne globoke ne pretirano široke kombinacije plasti. Presenetljivo se je najbolje odrezala relativno majhna arhitektura s štirimi skritimi plastmi (slika 4.1).



Slika 4.1: Shema arhitekture nevronske mreže.

Napovedna točnost na validacijskem setu podatkov je bila izračunana po tem, ko je bila gradnja modela zaključena. Ustavila se je, ko se rezultat na majhnem validacijskem setu ni izboljšal 15-krat. Temu pravimo ustavitveni pogoj in Keras omogoča preprosto implementacijo s pomočjo funkcije *EarlyStopping*.

```
callbacks = [keras.callbacks.EarlyStopping(
                                monitor='val_loss',
                                min_delta= 0,
                                patience=15,
                                verbose=0,
                                mode='auto')]
```

Do te točke smo opisali definicijo modela ter ustavitvenega pogoja. Izbrati moramo tudi optimizacijsko funkcijo in odločili smo se za *categorical crossentropy* (enačba 4.1).

$$H(p,q) = -\sum_x p(x) \times log(q(x)) \tag{4.1}$$

Parametra p in q predstavljata dejansko distribucijo razredov ter predvideno, izračunano distribucijo, ki je izhod *softmax* funkcije. Izhod predstavlja napako, ki jo je algoritem napravil pri gradnji modela. Manjša kot je napaka, bolj točni bodo rezultati. Algoritem, ki poskuša zmanjšati to napako imenujemo optimizator. Izbrali smo verzijo gradientnega spusta imenovano Adam [14]. Empirični rezultati naj bi potrjevali njegovo primernost pri problemih, kjer je treba klasificirati več razredov [18].

Na začetku tega poglavja smo omenili dva primera plasti - *Dense* in *Dropout*. Terminologija je Kerasova, v naslednjih dveh poglavjih pa bomo predstavili uporabo in namen teh plasti.

4.1.1 Plast Dense

Klic *Dense* predstavlja klasično gosto povezano plast nevronske mreže. Sprejme deset argumentov, za nas pa so bili relevantni le *activation*, *units* ter *input_dim*. Slednji se uporabi le v prvi, vhodni plasti in predstavlja dimenzije

vhoda. *Units* predstavlja dimenzijo izhoda, *activation* pa predstavlja našo izbiro aktivacijske funkcije.
Uporabili smo *relu* ter posplošitev logistične funkcije *softmax* na zadnji plasti. Slednja omogoča klasificiranje po razredih.

```
Dense(450, input_dim=dim, activation='relu')
```

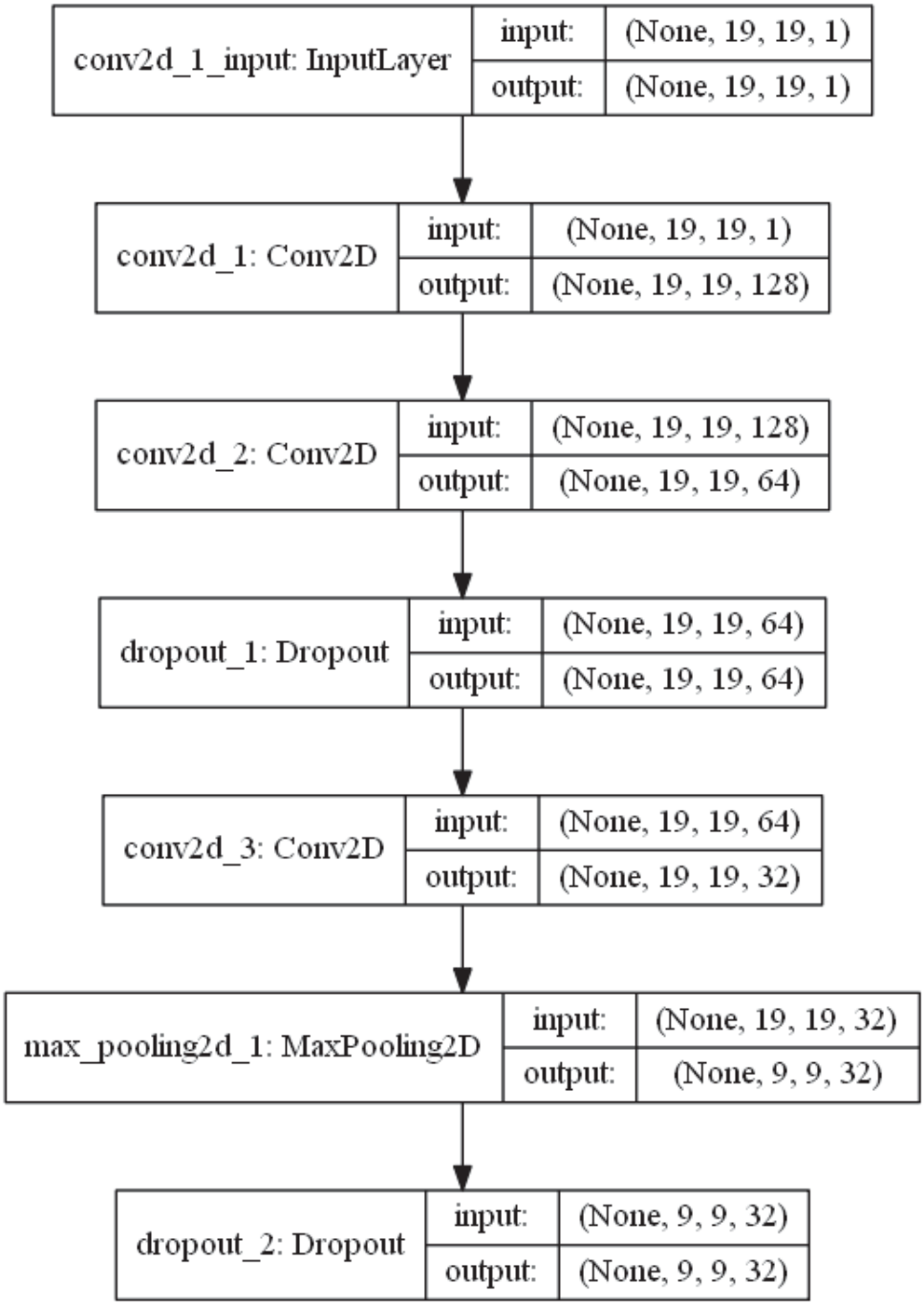
4.1.2 Plast Dropout

Dropout [20] je patentirana metoda internetnega velikana Google Inc., ki preprečuje preveliko prileganje. Pri Kerasu jo kličemo za vsako plastjo *Dense*. Deluje tako, da preprosto izpusti naključne nevrone (vrednosti) ter njihove povezave.
Dropout smo uporabili za vsako plastjo (razen zadnjo). Preizkušali smo vrednosti med 10 ter 40%. Različne vrednosti so bile optimalne pri različnih arhitekturah v našem primeru pa se je najbolje obnesla vrednost 30%, kjer smo uporabili plitko ter relativno ozko arhitekturo.

```
Dropout(0.2)
```

4.2 Konvolucijska nevronska mreža

Drug tip nevronske mreže, ki smo jo zgradili, je konvolucijska. Pri oblikovanju topologije smo se zgledovali po predhodnih uspešnih raziskavah in implementacijah. Tipično je izmenjevanje plasti konvolucije (*Conv2D*) ter maksimalnega združevanja (*MaxPool2D*), čemur sledi majhno število polno povezanih plasti(*Dense*) (Jarret in sodelavci [13], Krizhevsky in sodelavci [16], Ciresan in sodelavci [8]). Tudi mi smo uporabili enak pristop, za preprečevanje prevelikega prileganja učni množici pa uporabimo plast izpuščanja nevronov (*Dropout*). K temu pripomore tudi maksimalno združevanje (*MaxPool2D*). Poleg plasti, ki so specifične za konvolucijo na koncu uporabimo tudi standardne *Dense* plasti, ki vrnejo končni izhod, klasifikacijo po razredih. Na manjšem delu podatkov (da prihranimo na času) smo preizkušali različne kombinacije skritih plasti. Na koncu se je kot najboljša izkazala kombinacija treh zaporedij *Conv2D*, *Dropout* ter *MaxPool2D* plasti (slika 4.2).



Slika 4.2: Shema enega dela arhitekture konvolucijske nevronske mreže.

V tem delu bomo natančneje predstavili pomembnejše plasti in njihovo uporabo v Kerasu.

4.2.1 Plast Conv2D

Conv2D omogoča preprosto implementacijo konvolucijske plasti v Kerasu. Uporaba konvolucije na vhodni plasti zahteva specifikacijo oblike vhodne matrike. Funkcija zahteva parameter *input_shape*. Ker so konvolucijske mreže primerne predvsem za analizo slik, pričakuje 2D konvolucija za vhod 3D matriko. Tretja dimenzija je namenjena RGB delu slike, v našem primeru pa jo zapolnimo kar s konstantami in pri gradnji modela ne doprinese dodatnih informacij.

V ostalih plasteh je prvi parameter, ki ga uporabimo, *filters*. Prejme celoštevilski vhod, ki vpliva na dimenzionalnost izhodnega prostora (število izhodnih filtrov iz konvolucije). *Kernel_size* specificira velikost filtra, velikost koraka pa nastavimo s *strides*. Aktivacijska funkcija se določi s pomočjo parametra *activation*. Uporabljamo tudi *padding*, ki vrne dimenzije izhoda na velikost vhoda (po konvoluciji se dimenzionalnost zmanjša).

```
Conv2D(64, (3, 3), activation='relu', padding='same')
```

4.2.2 Plast MaxPool2D

Maksimalno združevanje ima podoben namen kot plast *Dropout* in je pogosto uporabljena takoj pred ali za njo. Uporaba v Kerasu je dokaj preprosta. Poslužimo se le dveh parametrov. Prvi je velikost okna *pool_size*, ki izbira maksimalne vrednosti, drugi pa je korak *strides*.

```
MaxPool2D(pool_size=(2, 2), strides=2)
```

4.2.3 Flatten

Ta plast služi le temu, da spremeni dimenzije izhoda prejšnje plasti v vektor, ki ga lahko prejme plast *Dense*.

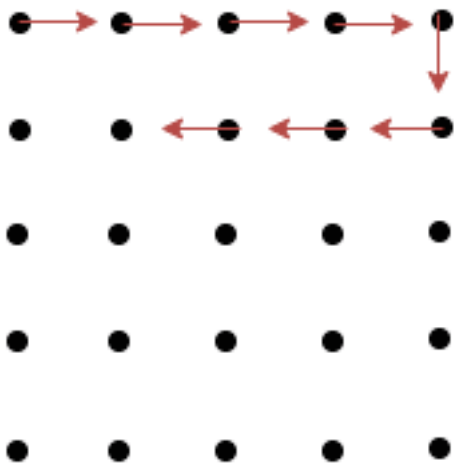
```
Flatten()
```

4.3 Transformacija vektorja v matriko

Opisali smo topologijo naših konvolucijskih nevronskih mrež, manjka nam le še pretvorba podatkov v obliko, ki bi jo lahko Kerasova *Conv2D* uporabila. Naši primeri so v vektorski obliki, zato jih je treba oblikovati, kot da so slike. Raziskave oblikujemo tako, da dobijo matrično obliko. To lahko storimo na mnogo načinov, ker pa želimo ohraniti bližino elementov, uporabimo naslednje tri transformacije.

4.3.1 Zaporedna transformacija

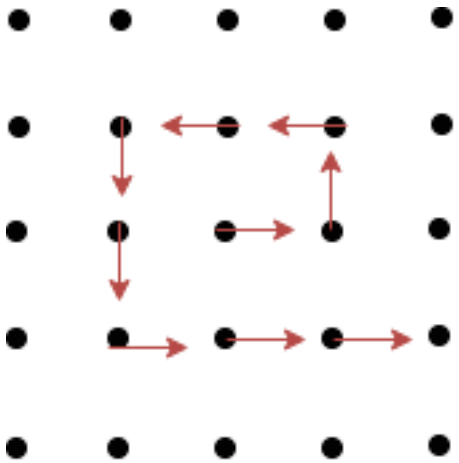
Najpreprostejši način za prenos vektorja 1xN v matriko je, da razkosamo vektor na enake dele, ki jih položimo enega nad drugega (slika 4.3). Če bi želeli iz vektorja [1x400] ustvariti matriko bi tako dobil velikosti [20x20]. Mi delamo z vektorjem [1x337], najbližja enakostranična matrika je velikosti [19x19]. Vrednosti, ki niso definirane od 337. zaporednega mesta naprej zapolnimo enako kot nedefinirane vrednosti znotraj vektorja - s povprečjem. Enaka velikost matrike se uporablja pri naslednjih transformacijah, spremeni se le način prenosa elementov.



Slika 4.3: Zaporedje transformacije *Zaporedna transformacija*.

4.3.2 Spiralna transformacija

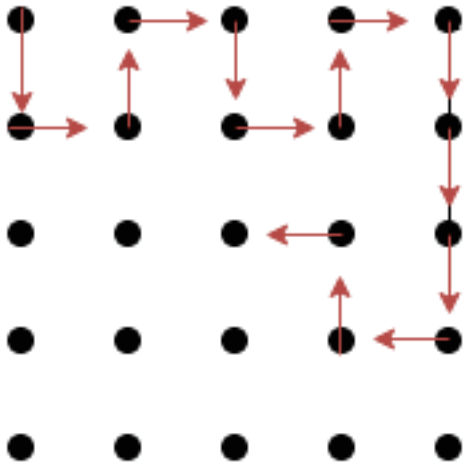
Drug način transformacije, ki smo ga preizkusili, je spirala (slika 4.4). Začnem v centru ter se krožno oddaljujem od centra, da polnim od središča vedno bolj oddaljene krožnice (oziroma natančneje kvadrate). Zdi se, da bi tak način pretvorbe ustvaril zelo specifične vzorce in 'sliko' podatkov, ki ohrani veliko lokalnosti elementov v centru ter vedno manj, ko se od centra oddaljuje. Zadnja iteracija bi ustvarila lokalno popolnoma dislocirane elemente, ki pa lahko tvorijo posamezen vzorec z elementi, ki so bližje centru.



Slika 4.4: Zaporedje transformacije *Spiralna transformacija*.

4.3.3 Transformacija kača

Ta pretvorba vektorja v matriko ohrani največ informacije o lokalnosti (slika 4.5). Vsak naslednji element se postavi največ za eno polje stran od prejšnjega. Najprej se premakne N-krat navzdol. Ko pride N elementov nižje, se prestavi en stolpec v desno ter iterativno nadaljuje za N mest navzgor. Ko pride do vrha (N mest gor) se postopek ponovi. Ko pride do konca matrike (horizontalno 19 mest), se celoten proces ponovi v nasprotni smeri - s tem, da začne N mest nižje ter da se premika v nasprotni smeri kot prej, torej v levo. Celoten proces se ponavlja, dokler ne pride do konca mej matrike - do 19. vrstice. Glavna spremenljivka v tej metodi je N. Spreminjali smo ga vse od 2 do 5.



Slika 4.5: Zaporedje transformacije *Transformacija kača*.

Poglavje 5

Rezultati

Vsi rezultati za nevronske ter konvolucijske nevronske mreže so bili pridobljeni na računalniku s štirijedrnim procesorjem Intel i7 (7700), grafično kartico GeForce GTX 1060 s 6GB ter 16GB RAMom. Pri konvolucijskih nevronskih mrežah smo za vhod uporabili le najbolj uspešno transformacijo - spiralno. Med testiranjem se je namreč izkazalo, da zaporedna transformacija konsistentno proizvaja med 1% in 3% slabše napovedi. Podbno se je obnesla transformacija kača.

Za preverjanje napovedne točnosti smo uporabili K-kratno prečno preverjanje oz. *Stratified K-fold*, kjer smo K nastavili na 10. Deluje tako, da razdeli podatke na K delov. Izbrani so tako, da imajo približno enako porazdelitev razredov, kot jih ima celoten set podatkov. Enega izbere za testno množico, preostali pa služijo kot učna. Zgradi se napovedni model nakar se na testni množici preveri napovedna točnost. Ta postopek se ponovi K-krat. Končna točnost je povprečje točnosti vseh napovedi.

Rezultate smo primerjali s predhodnimi rezultati, ki so bili pridobljeni z uporabo metode gradient boosting [12], implementirane v knjižnici xgboost [6]. Čas za izvedbo desetkratnega prečnega preverjanja je znašal približno 70 ur na strežniku z 12-jedernim Xeon procesorjem in 64GB RAMom.

Rezultate predstavimo v vrstnem redu pridobivanja. Najprej pogledamo,

kako so se obnesle nevronske mreže, nakar predstavimo še uspešnost konvolucijskih nevronskih mrež. Vključeni niso rezultati 1D konvolucije, ki so pridobljeni na modelu z enako strukturo, kot jo ima naša CNN. Vse plasti so enake, le da so 2D operacije zamenjane z 1D. Napovedna točnost 10-kratnega prečnega preverjanja je bila 40.751%.

5.1 Nevronske mreže

Pri interpretaciji rezultatov nevronskih mrež se sklicujemo na tabelo 5.1 (NN).
Najprej se bomo posvetili času izvajanja gradnje modela z 10-kratnim prečnim preverjanjem. Arhitektura GPU je zelo primerna za delo z nevronskimi mrežami, kar se kaže v več kot štirikratni pohitritvi v primerjavi s CPU. S Theanom v zaledju je namreč algoritem na CPU potreboval kar 6h in 45 min, da je zaključil učenje. Na GPU mu je enako uspelo le v 1h in 47min. Delo na GPU smo preizkusili tudi s TensorFlowom. Bil je še nekoliko hitrejši kot Theano z 1h in 38min. 9 min pohitritve med zaledji tukaj ne predstavlja prevelike razlike.

S tem se razlike med rezultati zaledij tudi počasi končajo.
Napovedna točnost je namreč pri vseh evalvacijah algoritma praktično enaka (med 39.67% ter 39.84%). Nevronske mreže so se odrezale slabše kot gradient boosting, ki je imel povprečno napovedno točnost pri 42%. Če je pomembna hitrost, je vsekakor bolje uporabiti nevronske mreže, saj potrebujejo na GPU (10-kratno prečno preverjanje) nekoliko manj kot dve uri. To je več kot 35-krat manj kot pa gradient boosting (70h za 10-kratno prečno preverjanje).

5.2 Konvolucijske nevronske mreže

Pri interpretaciji rezultatov nevronskih mrež se sklicujemo na tabeli 5.1 (CNN) in 5.2 (CNN).
Majhna razlika med časom izvajanja pri nevronskih mrežah se ni prenesla

tudi na konvolucijske. TensorFlow je potreboval za gradnjo modela pri 10-kratnem prečnem preverjanju 17h in 47min. Pričakovan čas izvajanja za Theano bi bil sodeč po prejšnjem poglavju okoli 19h. Potreboval pa je kar 78h in 42 min. Težko je reči čemu botruje tako velika razlika v času izvajanja, saj je pričakovana razlika (v korist TensorFlowa) med 10% in 25% [1]. Theano je tako potreboval celo več časa kot gradient boosting, ki je zaključil delovanje po 70h.

Napovedna točnost za Theano je 41.87%, TensorFlow pa je dosegel 41.78%. Razlika v točnosti med zaledji je zanemarljivo majhna in jo lahko pripišemo različni začetni inicializaciji uteži. Ko primerjamo točnosti konvolucijskih nevronskih mrež ter gradient boostinga, vidimo, da so zelo blizu.

Ocenili smo tudi napovedno točnost za najboljših 5 in 10 rezultatov. To pomeni, da smo napoved razreda šteli za pravilno, če se je nahajala med najboljšimi 5 oz. 10 napovedmi. Rezultati so tukaj veliko bolj vzpodbudni, četudi so praktično enaki med obemi zaledji kot tudi gradient boostingom. Ko vzamemo najboljših 5 predvidevanj, se je pravilen razred znašel med njimi v 72%, če pa pogledamo najboljših 10 se napovedna točnost dvigne na kar 82%.

5.3 Tabele

Zaledje	**Procesiranje**	**Algoritem**	**Točnost [%]**	**Čas izvajanja [h:m:s]**
Theano	CPU	NN	39.847	6:45:52
Theano	GPU	NN	39.782	1:47:33
TensorFlow	GPU	NN	39.674	1:38:51
Theano	GPU	CNN	41.877	78:42:11
TensorFlow	GPU	CNN	41.783	17:47:47
/	CPU	GRAD. BOOST	42	70:00:00

Tabela 5.1: Točnost in čas izvajanja algoritmov za 10-kratno prečno preverjanje

Oznaki TOP 5 in TOP 10 v tabeli 5.2 predstavljata točnost napovednega

modela, če vzamemo za pravilno napoved vsako, kjer je pravilna klasifikacija razreda med prvimi petimi in desetimi napovedmi.

Zalednje	**Procesiranje**	**Algoritem**	**TOP 5** [%]	**TOP 10** [%]
Theano	GPU	CNN	71.892	82.184
TensorFlow	GPU	CNN	72.094	82.233
/	CPU	GRAD. BOOST	72	82

Tabela 5.2: Točnost za TOP 5 in TOP 10

Preizkusili smo tudi pristop, kjer je ustavitveni pogoj število iteracij. Začeli smo pri 200 in končali pri 1000 iteracijami s korakom 100. Pri tako visokem številu epoh je napovedna točnost zgrajenega modela padla veliko nižje kot naša najboljša točnost. Izkazalo se je, da je optimalno število iteracij pridobljeno na tak način med 200 in 300. Prilagamo rezultate za 280 iteracij (tabela 5.3). Čas, ki je potreben za gradnjo modela z velikim številom epoh je neprimerno višji, kot tisti, pri avtomatskem ustavljanju algoritma (ustavitev se je zgodila pri približno 40 iteracijah). Natančnost pri fiksnem številu iteracij (280) je bila v povprečju višja za 1%. Izkazalo se je, da tak pristop zahteva preveč časa, da bi bil uporaben (v primerjavi z avtomatsko ustavitvijo).
Čas izvajanja je med zaledji tako na GPU kot tudi pri avtomatskem ustavljanju, primerljiv. Podobno je CPU potreboval veliko dlje kot GPU, da zaključi z vsemi operacijami.

Zalednje	**Procesiranje**	**Algoritem**	**Točnost** [%]	**Čas izvajanja** [h:m:s]
Theano	CPU	NN (280)	40.658	60:09:18
Theano	GPU	NN (280)	40.673	14:42:53
TensorFlow	GPU	NN (280)	40.66	12:17:47

Tabela 5.3: Točnost in čas gradnje napovednega modela zgrajenega skozi 280 iteracij pri 10-kratnem prečnem preverjanju

Poglavje 6

Sklepne ugotovitve

Keras se je izkazal za odličen API. Sestavljanje topologije mreže je intuitivno in preprosto, obenem pa imamo dovolj svobode, da organiziramo algoritem točno tako, kot želimo. Menjavanje med zaledji je ravno tako preprosto. API zaenkrat redno posodabljajo, spletna skupnost pa je dovolj aktivna, da v primeru težav hitro najdemo rešitve.

Zaledji Theano in TensorFlow sta primerljivi in težko bi rekli, da lahko popularnost slednjega pripišemo veliko boljši implementaciji. Velika razlika je bila pri izvajalnem času CNN, za kar nimamo dobre razlage.

Rezultati so pokazali, da lahko uporabimo konvolucijske nevronske mreže tudi na podatkih, ki niso tipični zanje. Veliko število atributov in razredov, ki so vsakdan v širši medicinski diagnostiki, predstavlja problem, s katerim se konvolucijske nevronske mreže, kot tudi nevronske mreže ne spopadajo bolje kot bolj klasične metode strojnega učenja (rezultati so, glede točnosti, primerljivi).

Izkazalo se je, da lahko vplivamo na napovedno točnost z različnimi transformacijami vektorjev v matrike - razlika se je gibala med 1% in 3%. Verjetno je, da posamezna transformacija ni boljša kot neka druga in da je razlika v točnosti posledica posameznega podatkovnega seta. To pomeni, da bi bilo vedno potrebno uporabiti več različnih transformacij, saj se enaka ne izkaže vedno za najboljšo.

V prihodnje bi lahko preizkusili več različnih transformacij matrik (npr. diagonalizacijo), kot tudi izboljšane topologije nevronskih mrež. Verjetnost, da smo izbrali optimalno kombinacijo plasti za ta problem je zelo majhna. Izboljšali bi lahko tudi podatkovni set. Število nedefiniranih vrednosti bo seveda ostalo veliko, kot bo tudi število razredov. Vplivamo pa lahko na število primerov razredov. Če bi lahko povečali število primerov, kjer je trenutno skopo, bi algoritmi strojnega učenja lažje prepoznali značilnosti njihovih razredov in jih tako bolj točno klasificirali.

Literatura

[1] Rami Al-Rfou, Guillaume Alain, and Amjad Almahairi et al. Theano: A Python framework for fast computation of mathematical expressions. *CoRR*, abs/1605.02688, 2016. URL: `http://arxiv.org/abs/1605.02688`.

[2] Raman Arora, Amitabh Basu, Poorya Mianjy, and Anirbit Mukherjee. Understanding deep neural networks with rectified linear units. *CoRR*, abs/1611.01491, 2016. URL: `http://arxiv.org/abs/1611.01491`.

[3] Debnath Bhattacharyya and Tai-hoon Kim. Brain tumor detection using MRI image analysis. In *International Conference on Ubiquitous Computing and Multimedia Applications*, pages 307–314. Springer, 2011.

[4] L Susan et al. Blackford. An updated set of basic linear algebra subprograms BLAS. *ACM Transactions on Mathematical Software*, 28(2):135–151, 2002.

[5] Maureen Caudill. Neural networks primer, part I. *AI Expert*, 2(12), 1987.

[6] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016.

[7] Dan C. Ciresan, Ueli Meier, Luca Maria Gambardella, and Jürgen Schmidhuber. Deep big multilayer perceptrons for digit recognition. In *Neural Networks: Tricks of the Trade*, pages 581–598. Springer, 2012.

[8] Dan C Ciresan, Ueli Meier, Jonathan Masci, Luca M Gambardella, and Jürgen Schmidhuber. High-performance neural networks for visual object classification. *arXiv preprint arXiv:1102.0183*, 2011.

[9] Ayşe Demirhan, Mustafa Törü, and İnan Güler. Segmentation of tumor and edema along with healthy tissues of brain using wavelets and neural networks. *IEEE journal of biomedical and health informatics*, 19(4):1451–1458, 2015.

[10] Cire S. et al. Mitosis detection in breast cancer histology images with deep neural networks. In *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2013: 16th International Conference, Proceedings, Part II*, pages 411–418. Springer Berlin Heidelberg, 2013.

[11] Cruz-Roa et al. Automatic detection of invasive ductal carcinoma in whole slide images with convolutional neural networks. In *SPIE Medical Imaging*, volume 9041, pages 904103–904103. International Society for Optics and Photonics, 2014.

[12] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[13] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th International Conference on Computer Vision*, Sept 2009.

[14] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *ArXiv e-prints*, dec 2014. `arXiv:{1412.6980}`.

[15] Igor Kononenko. Machine learning for medical diagnosis: history, state of the art and perspective. *Artificial Intelligence in Medicine*, 23(1), 2001.

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2012.

[17] R.S. Khule R. J. Deshmukh. Brain tumor detection using artificial neural network fuzzy inference system. *International Journal of Computer Applications Technology and Research(IJCATR)*, 3, 2014.

[18] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016. URL: `http://arxiv.org/abs/1609.04747`.

[19] Frank Seide and Amit Agarwal. CNTK: Microsoft's open-source deep-learning toolkit. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 2135–2135, New York, NY, USA, 2016. ACM. URL: `http://doi.acm.org/10.1145/2939672.2945397`, `doi:10.1145/2939672.2945397`.

[20] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014. URL: `http://jmlr.org/papers/v15/srivastava14a.html`.

[21] B. van Ginneken, A. A. A. Setio, C. Jacobs, and F. Ciompi. Off-the-shelf convolutional neural network features for pulmonary nodule detection in computed tomography scans. In *2015 IEEE 12th International Symposium on Biomedical Imaging (ISBI)*, pages 286–289. IEEE, 2015.

[22] Dayong Wang, Aditya Khosla, Rishab Gargeya, Humayun Irshad, and Andrew H. Beck. Deep learning for identifying metastatic breast cancer. *arXiv preprint arXiv:1606.05718*, 2016.

[23] Xingjian et al. Yan. Classification of lung nodule malignancy risk on computed tomography images using convolutional neural network: A comparison between 2D and 3D strategies. In *Computer Vision – ACCV 2016 Workshops: ACCV 2016 International Workshops, Taipei, Taiwan, November 20-24, 2016, Revised Selected Papers, Part III*, pages 91–101. Springer International Publishing, 2017.