Smanjenje dimenzionalnosti prostornih podataka pomoću veštačkih neuronskih mreža

Matematički fakultet
Univerzitet u Beogradu

Student:
Miloš Manić

Mentor:
dr Mladen Nikolić

Članovi komisije:
dr Filip Marić, vanredni profesor,
dr Aleksandar Kartelj, docent

Beograd, 2018.

Sadržaj

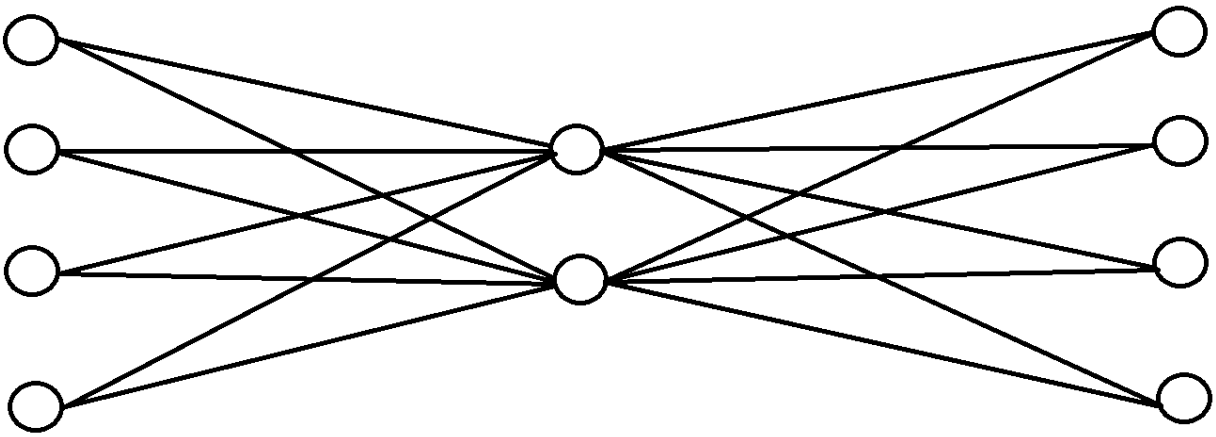
Glava 1

Uvod

Algoritmi mašinskog učenja su u poslednjih petnaestak godina vrlo brzo napredovali i može se videti primena na svakom koraku. Razlog tome je napredak u hardveru, kao i dostupnost uređaja u našem životu koji svakodnevno prikupljaju gigabajte informacija, i sve veće oslanjanje na tehnologiju za obavljanje različitih radnji koje su bile teške ili nemoguće u prošlosti. Neke od najpoznatijih svakodnevnih primena mašinskog učenja su prepoznavanje govora, predlaganje sadržaja, autonomna vožnja automobila i mnoge druge. U poslednjih desetak godina se usled ovog napretka sve češće govori o pojmu „Big Data" i njegovim implikacijama. Količina podataka raste, kao i brzina kojom se oni mogu obrađivati. Da bi se izvukle korisne informacije iz podataka koriste se algoritmi mašinskog učenja. Ovi algoritmi često imaju problem sa visokodimenzionalnim podacima. Ovakvi podaci su veoma česti, naročito u nekim primenama kao što je računarski vid, gde se slike predstavljaju kao matrice piksela.

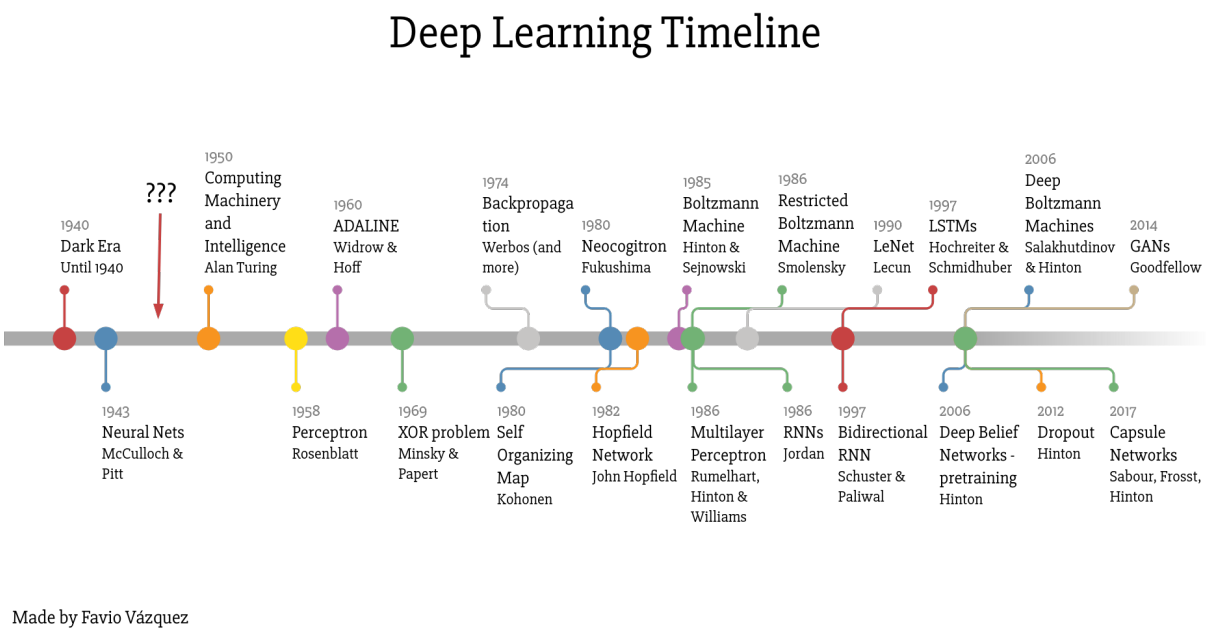
Uobičajeni način smanjenja dimenzionalnosti su algoritmi koje smišljaju eksperti u oblasti, i primenjuju se samo na probleme određenog tipa i domena. Primer su algoritmi za prepoznavanje linija ili ivica na slici, algoritmi za obradu teksta i slično. Ovakvi algoritmi pretpostavljaju određena svojstva podataka i rešavaju samo konkretne probleme, pa je poželjno naći algoritme koji rade za opštije tipove problema. Jedan od takvih metoda je analiza glavnih komponenti, koja koristi principe linearne algebre za smanjivanje broja atributa i eliminisanje korelacija u podacima. Ovaj metod će biti testiran u radu. Autoenkoderi su drugi metod za smanjenje dimenzionalnosti, koji se zasniva na korišćenju neuronskih mreža. Arhitektura autoenkodera je takva da je njegov cilj da kopira ulaz na izlaz, ali uz neko ograničenje u reprezentaciji, tako da ne može da se kopira savršeno. Primer strukture autoenkodera sa četvorodimenzionalnim ulazom i dvodimenzionalnom reprezentacijom je prikazan na slici 1.1. Autoenkoderi su drugi metod koji je testiran u radu.

Neuronske mreže su matematički modeli mašinskog učenja koji se sastoje od neurona. Neuronska mreža se trenira podešavanjem parametara učenja, tako da se smanjuje zadata greška korišćenjem nekog algoritma za optimizaciju. Najčešći algoritmi za optimizaciju koji se koriste za treniranje neuronskih mreža se zasnivaju na praćenju gradijenta. Na slici 1.2 su prikazani zapaženi radovi koji su doprineli napretku neuronskih mreža. Autoenkoderi se, kao specijalne vrste neuronskih mreža, treniraju smanjivanjem greške rekonstrukcije, koja upoređuje polazne podatke i podatke dobijene na izlazu iz autoenkodera. Reprezentacija u novom prostoru atributa se dobija tako što se uzmu izlazi neurona najmanjeg sloja, odnosno poslednjeg sloja enkodera.



Slika 1.1: Primer autoenkodera sa četvorodimenzionalnim ulazom i dvodimenzionalnom reprezentacijom

Jedan od razloga zbog kojih je smanjenje dimenzionalnosti zanimljivo za prostorne podatke je postojanje velikih količina klimatoloških, geoloških, zemljišnih i drugih vrsta karata, koje je nekada potrebno pretraživati. Na primer, neka je za datu temperaturnu kartu potrebno naći dan u prošlosti kojem odgovara slična temperaturna karta. Za to bi bilo potrebno uraditi poređenje sa hiljadama karata visoke rezolucije. Ovo može biti računski zahtevno. Ukoliko bi se karte mogle predstaviti pomoću vektora male dimenzionalnosti (npr. nekoliko desetina), pretraga bi bila drastično brža. Naravno, za to je potrebno da sličnost polaznih reprezentacija visoko korelira sa sličnošću novih reprezentacija. Ovakva svojstva diskutovanih transformacija će biti evaluirana u radu.



Slika 1.2: Važne publikacije koje su doprinele napretku neuronskih mreža [28]

U radu su sprovedeni eksperimenti koji koriste mape padavina na teritoriji Sjedinjenih Američkih Država, na koje se primenjuju različiti autoenkoderi, kao i analiza glavnih komponenti. Performanse ovih metoda se upoređuju na više načina, imajući u vidu primenu

ovih algoritama u praksi. Na kraju su prikazane vizuelizacije rezultata ovih metoda, kao intuitivni dokaz funkcionalnosti onoga što je prikazano. Dobijeni rezultati pokazuju da ove metode mogu biti korisne u praksi, ali ostavljaju još mesta za napredak u slučaju autoenkodera.

Drugo poglavlje ovog rada je uvod u veštačke neuronske mreže, njihovu arhitekturu i podešavanje parametara treninga. Treće poglavlje definiše problem smanjenja dimenzionalnosti, kao i metode koje se koriste za praktičnu primenu u radu. Četvrto poglavlje opisuje alate koji su korišćeni za implementaciju korišćenih metoda. Opis konkretnih eksperimenata i rezultati primene su prikazani u petom poglavlju.

Glava 2

Veštačke neuronske mreže

Veštačke neuronske mreže su matematički modeli inspirisani nervnim sistemom u biologiji. Osnovna jedinica obrade neuronske mreže je neuron, koji je inspirisan biološkim neuronom. Ovakav neuron u veštačkoj neuronskoj mreži je matematička i grubo uprošćena aproksimacija biološkog neurona. Bez obzira na motivaciju, neuronske mreže su se razvijale u pravcu principa softverskog inženjerstva i optimizacionih metoda, i dosta su drugačije od bioloških.

Svaki neuron u veštačkoj neuronskoj mreži se sastoji od ulaznih veza sličnih dendritima u biologiji, aktivacione funkcije koja nelinearno transformiše ponderisanu sumu ulaza u broj na izlazu i ima jedan izlaz koji može da se grana i može da bude ulaz za više različitih neurona (slično aksonu u biološkom neuronu, koji se vezuje za dendrite i stvara sinapse). Sistem uči tako što podešava težine veza (što odgovara jačini sinapse u biološkom sistemu).

Neuronske mreže se mogu predstaviti i kao grafovi, gde su neuroni čvorovi grafa, veze između neurona su veze između čvorova, i smer veza je smer toka podataka. Ulaz u neuronsku mrežu se takođe predstavlja preko čvorova grafa, ali se ovi čvorovi ne smatraju jedinicama obrade mreže, to jest neuronima, jer ne vrše nikakvu obradu. Najčešće se neuroni u mrežama organizuju u slojeve, a najčešće neuronske mreže su mreže sa propagacijom unapred, kod kojih ne postoje veze koje su usmerene iz neurona jednog sloja u prethodne slojeve već samo u sledeći sloj. Tako se neuronska mreža može predstaviti i kao matematički model koji se gradi kompozicijom funkcija, gde je svaki sloj jedna funkcija u toj kompoziciji, primenjen na vektor ulaza, koji je izlaz iz neurona prethodnog sloja [7]. Svi slojevi osim izlaznog se nazivaju skriveni slojevi. Postoje i neuronske mreže kod kojih su dozvoljeni ciklusi. Najpoznatiji primer su rekurentne neuronske mreže (*eng. Recurent Neural Networks, RNN)*. One su teže za treniranje i često se treniraju svođenjem na mreže sa propagacijom unapred.

2.1 Osnovne odluke pri dizajniranju neuronske mreže

Performanse neuronske mreže zavise od mnogo izbora pri modelovanju mreže. Najbitnije odluke pri modelovanju veštačke neuronske mreže su odluke o arhitekturi mreže i odabir funkcije greške pri treningu neuronske mreže. Arhitektura mreže podrazumeva tip mreže, broj slojeva i broj neurona svakog pojedinačnog sloja, kao i njihovu funkciju aktivacije. Funkcija greške pri treningu neuronske mreže se smanjuje pri treningu odabranom optimizacionom

metodom, koja se obično zasniva na praćenju gradijenta.

2.1.1 Funkcija aktivacije

Prva neuronska mreža se sastojala od jednog neurona i nazvana je perceptron [22]. Perceptron ima step funkciju kao funkciju aktivacije i uči podešavanjem težina koje odgovaraju ulaznim vrednostima na osnovu greške treninga. Step funkcija se definiše kao $step = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases}$. Ovakav model može da klasifikuje linearno separabilne klase. Najveća mana perceptrona je što ne može lako da se kombinuje u mrežu, jer step funkcija ima oštre promene, što je čini nediferencijabilnom u svim tačkama, pa je bilo kakva optimizacija gradijentom nemoguća, obzirom da se u nediferencijabilnim tačkama ne može naći gradijent potreban za treniranje. Razvijene su i druge funkcije aktivacije, koje su doprinosile boljoj optimizaciji i kombinovanju većeg broja jedinica. Neke od češće korišćenih funkcija aktivacije su sledeće:

- sigmoidna funkcija σ,
- hiperbolički tangens $tanh$,
- $ReLU$ funkcija i njene varijante.

Sigmoidna funkcija je funkcija koja se računa kao

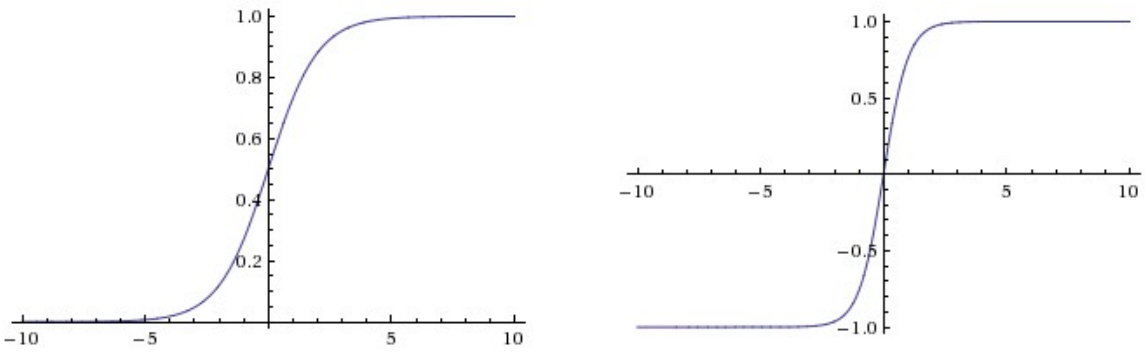
$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$

Domen sigmoidne funkcije je $\mathbb{R}$ i ova funkcija taj domen preslikava u skup $(0, 1)$, tako da veliki pozitivni brojevi budu preslikani u brojeve blizu 1, a veliki negativni brojevi budu preslikani u brojeve blizu 0, kao što je prikazano na slici 2.1.

Sigmoidna funkcija ima oblik sličan step funkciji, ali za razliku od step funkcije ona je glatka i diferencijabilna u svim tačkama. Sigmoidna funkcija je istorijski povezivana sa biološkim neuronima jer, kao i step funkcija, ima dobru intuitivnu interpretaciju, koja može da se protumači kao „uključeno-isključeno", ali ima nedostatke koji su doprineli da se u modernim neuronskim mrežama ne koristi u praksi. Sigmoidna funkcija takođe ima izvod koji se lako računa jer se moze izraziti samo u terminima vrednosti same funkcije $\sigma'(x) = \sigma(x)(1 - \sigma(x))$. Glavni nedostaci ove funkcije su:

- Sigmoidna funkcija u velikim mrežama „zasiti" (*eng. saturate*) i „ubija" (*eng. kill*) gradijente. Problem zasićenja sigmoidne funkcije se sastoji u tome što je gradijent sigmoidne funkcije za velike pozitivne i negativne brojeve skoro 0, jer funkcija vrlo sporo raste. Zbog toga sigmoidni neuroni drastično usporavaju bilo kakvu promenu vrednosti na osnovu gradijenta. Ovo se dešava često u dubokim neuronskim mrežama, a uspeh u treniranju uglavnom zavisi od dobrog inicijalnog podešavanja težina, kao i dobrog odabira koraka učenja pri treningu.
- Izlaz iz sigmoidnog neurona nije simetričan oko nule. Simetrija ulaza u sigmoidnu funkciju, koji je simetričan oko nule, i izlaza, koji je simetričan oko $0,5$ dovodi do neželjenog

„cik-cak" kretanja u prostoru parametara. Posle dovoljno iteracija se ove nepravilnosti poprave i težine se centriraju oko nule. Ovaj nedostatak doprinosi usporenju učenja više nego lošim krajnjim rezultatima, pa je manje ozbiljan od prvog problema, koji može da uzrokuje praktično zaustavljanje učenja.



Slika 2.1: Levo je prikazan grafik sigmoidne funkcije. Funkcija je definisana za ceo skup realnih brojeva i ima horizontalne asimptote u tačkama 0 i 1. Desno je prikazana funkcija hiperbolički tangens. Funkcija *tanh* je definisana za ceo skup realnih brojeva i ima horizontalne asimptote u tačkama -1 i 1.

Funkcija hiperbolički tangens, *tanh*, uzima vrednosti u intervalu (-1,1), što je razlikuje u odnosu na sigmoidnu funkciju po tome što je izlaz simetričan u odnosu na nulu, pa se izbegava drugi prikazani problem sigmoidne funkcije, što znači malo bržu konvergenciju. U praksi se često preferira u odnosu na sigmoidnu. Takođe, važi $tanh(x) = 2\sigma(2x) - 1$, pa je tanh samo skalirana sigmoidna funkcija, što se vidi na slici 2.1. Izvod tanh funkcije je $tanh'(x) = 1 - tanh(x)^2$. Pošto *tanh* funkcija takođe ima horizontalne asimptote u 1 i -1, prvi problem sigmoidne funkcije i dalje postoji u tanh funkciji pa učenje može biti prekinuto prerano.

ReLU (*eng. Rectified Linear Unit*) funkcija se računa kao

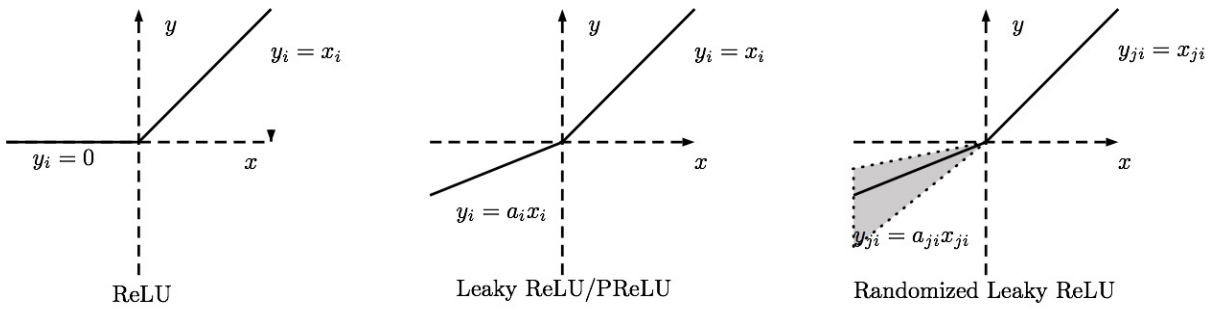
$$ReLU(x) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases}.$$

ReLU funkcija je jednostavnija za implementaciju i manje zahtevna u računskom smislu, jer samo treba implementirati linearnu funkciju, za razliku od sigmoidne i tanh funkcije, koje se računaju složenije. U praksi je primećeno veliko poboljšanje performansi u odnosu na sigmoidnu i *tanh* aktivaciju [29]. Pretpostavlja se da je uzrok to što ReLU ima linearnu formu koja se ne zasićuje, to jest nema horizontalnu asimptotu u 1, za razliku od sigmoidne i tanh funkcije. Sa druge strane postoji šansa da se ReLU neuroni tokom treninga „ugase" ili „umru" (*eng. dead neurons*), to jest da pređu nepopravljivo u stanje gde je izlaz uvek 0, zbog toga što je funkcija konstantna levo od nule, pa je gradijent 0. Ovaj problem se prevazilazi dobrim inicijalizovanjem mreže, kao i dovoljno malim korakom učenja. Još jedan način je koristiti funkciju koja nije identički nula za negativan ulaz već ima oblik:

$$leakyReLU(x) = \begin{cases} x, & x > 0 \\ -\delta x, & x \leq 0 \end{cases}.$$

Gde je δ neki parametar blizak nuli. Ovakva funkcija aktivacije se zove propustljivi ReLU (*eng. leaky ReLU*). Postoje različite varijante propustljivih ReLU funkcija, koje se razlikuju

u tome kako se računa parametar δ. Kod klasičnih *leakyReLU* funkcija δ je hiperparametar i bira se prilikom dizajniranja mreže. Nasuprot tome postoje parametarske ReLU funkcije kod kojih je δ parametar koji se uči treningom mreže. Takođe postoje i randomizovane ReLU funkcije čiji je parametar δ neki nasumičan broj blizak nuli. Primeri ReLU funkcija su prikazani na slici 2.2.



Slika 2.2: Različite ReLU funkcije, levo klasična ReLU funkcija, u sredini propustljiva ReLU funkcija i desno randomizovana ReLU funkcija sa nasumičnim parametrom δ [29]. Sve funkcije su linearne za pozitivni argument, a razlikuju se samo u parametru δ koji opisuje rast funkcije za argumente manje od 0.

2.1.2 Trening neuronske mreže

Treniranje neuronske mreže se postiže minimizacijom funkcije greške. Izbor funkcije greške neuronske mreže zavisi od tipa podataka i od tipa i instance problema koji se rešava.

Funkcija greške je funkcija oblika $l(y, \hat{y})$, gde je $\hat{y}$ očekivani (ispravan) izlaz, a y dobijeni izlaz pri treningu mreže. Funkcija greške uglavnom zavisi od *reziduala* $r = y - \hat{y}$, pa je $l(y, \hat{y}) = l(r) = l(y - \hat{y})$ [27].

Najčešće korišćene funkcije greške za regresiju su:

- L_2 **norma ili kvadratna greška**: $L_2(r) = r^2$. Kvadratna greška ima prednost što je diferencijabilna u svim tačkama, što je korisno za optimizaciju pri treningu, ali je osetljiva na ekstremne vrednosti, jer funkcija kvadratno raste udaljavajući se od nule. Obično se koristi srednja kvadratna greška (*eng. mean squared error, MSE*), koja se računa kao: $mse(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$, gde je n dimenzija ulaza, a $\hat{y}_i$ i-ta komponenta $\hat{y}$ i y_i i-ta komponenta y.
- L_1 **norma ili apsolutna ili Laplasova greška**: $L_1(r) = |r|$. Apsolutna greška nije osetljiva na ekstremne vrednosti, ali nije diferencijabilna u svim tačkama. I za apsolutnu grešku se najčešće koristi kao srednja vrednost, zvana srednja apsolutna greška (*eng. mean apsolute error, MAE*) koja se računa kao $mae(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$,gde je n dimenzija ulaza, a $\hat{y}_i$ i-ta komponenta $\hat{y}$ i y_i i-ta komponenta y.

Težine se uglavnom optimizuju praćenjem gradijenta greške. Trening neuronske mreže se vrši pomoću propagacije unatrag. Propagacija unatrag je način izračunavanja gradijenta neuronske mreže u odnosu na težine pomoću rekurzivne primene pravila izvoda složene funkcije.

Pravilo izvoda složene funkcije opisuje pravilan način računanja parcijalnog izvoda kompozitnih funkcija.

Neka su funkcije $g : \mathbb{R}^m \to \mathbb{R}^n$ i $f : \mathbb{R}^n \to \mathbb{R}$, tada

$$\partial_i(f \circ g) = \sum_{j=1}^{n} (\partial_j f \circ g)\partial_j g_j.$$

Intuitivno, od ulaza se podaci transformišu kroz slojeve sve do izlaza, i taj proces se naziva propagacija unapred, a zatim se parcijalni izvodi propagiraju unazad kroz slojeve. Izlaz se računa pomoću funkcija koje su zadate arhitekturom mreže, a zatim se parcijalni izvod računa tako što se pravilom izvoda složene funkcije nadovezuje parcijalni izvod na parcijalni izvod iz sledećeg sloja [25].

Na osnovu izvoda se menjaju težine veza po određenim pravilima koje se definišu izborom optimizacionog metoda i hiperparametara. Po načinu na koji se prati gradient funkcije greške možemo podeliti optimizacione metode na fiksne, koje ne menjaju način na koji ažuriraju težine, i adaptivne, koje menjaju korak učenja.

Uobičajeni način za minimizaciju greške treninga je ažuriranje parametara (težina veza) suprotno od smera gradijenta. Gradijent je vektor koji sadrži parcijalne izvode funkcije po svim dimenzijama ulaznih promenljivih. Gradijent takođe opisuje pravac maksimalnog rasta funkcije, pa se prilikom minimizacije ide u suprotnom smeru.

Gradijentni spust se sastoji u tome da se naivno prati gradijent funkcije greške:

$$\Theta_{t+1} = \Theta_t - \eta G_t,$$

gde je Θ_t vektor parametara treninga u koraku t, G_t gradijent funkcije greške u koraku t, a η je parametar koraka učenja (*eng. learning rate*). Korak učenja je hiperparametar i po pravilu je fiksiran u toku treninga. Smatra se da je za dovoljno veliki skup podataka za trening i dovoljno mali korak učenja poboljšanje tokom treninga garantovano [25].

Osnovna verzija ovog algoritma se takođe ponekad zove *batch* gradijentni spust. U slučaju da se trenira na velikom skupu podataka i da je skup dovoljno homogen, parametri se mogu ažurirati iterativno računajući gradijent korišćenjem podskupova skupa za treniranje redom. Ovakva optimizacija se naziva stohastički gradijentni spust (*eng. stochastic gradient descent, SGD*). Postoje dve osnovne varijante stohastičkog gradijentnog spusta, u zavisnosti od dimenzije podskupova nad kojima se trenira.

Onlajn stohastički gradijentni spust (*eng. online stochastic gradient descent*) računa gradijent posebno za svaku instancu za treniranje. Eksperimentalni rezultati pokazuju da ovaj način često konvergira dobrim rešenjima, ali ne postoji garancija, s obzirom na to da sama konvergencija zavisi od redosleda elemenata pri treniranju. Ovakav način rada je ponekad neophodan, ako se neuronska mreža trenira na podacima koji se mere u realnom svetu i odmah koriste za modeliranje podataka, što može biti slučaj kada se model koristi za predviđanje vrednosti neke promenljive čija se vrednost meri i odmah koristi za treniranje modela, kao što je na primer predviđanje rasta ili pada cena akcija na berzi u odnosu na prethodne vrednosti i slično.

Matrične operacije imaju dobre performanse na novijim sistemima, naročito na sistemima koji koriste grafičke procesore, tako da je za dobro izabranu količinu podataka bolje

koristiti osnovni gradijentni spust umesto onlajn varijante. Trenutno najkorišćeniji pristup ima prednosti obe metode i zove se Mini-batch stohastički gradijentni spust (*eng. mini-batch stochastic gradient descent*). Ovaj pristup uzima veličinu podskupa nad kojim se računa gradijent za hiperparametar treninga, koji se bira na početku treniranja. Ovaj pristup takođe predstavlja generalizaciju obe metode, s obzirom na to da je za veličinu 1 to onlajn varijanta, a za veličinu n, gde je n veličina celog trening skupa, klasični gradijentni spust.

Postoje različite adaptivne optimizacione metode koje koriste znanje iz prethodnih iteracija da odrede korak učenja tako da se različiti parametri različito ažuriraju. Cilj je praviti velike promene za parametre koji se nisu menjali često i drastično u prošlim iteracijama, i male promene za parametre koji su bili ažurirani češće [2]. Jedna od njih je Adagrad metoda[4]. Adagrad metoda koristi znanje iz prethodnih iteracija kao zbir kvadrata gradijenata. Parametri se ažuriraju pokoordinatno formulom:

$$\Theta_{t+1} = \Theta_t - \frac{\eta}{\sqrt{\Delta_{t+1}} + \epsilon} G_{t+1}$$

Gde je Θ_t vektor parametara u epohi t, η korak učenja, $G_t = \nabla(\Theta_t)$, ϵ je broj blizak nuli, a $\Delta_t = \sum_{i=0}^{t} G_i^2$.

Pomoću sume kvadrata i korenovanja se normalizuje parametar koraka učenja. U praksi se pokazalo da bez ovih transformacija nad promenljivom Δ_t performanse ovog algoritma padaju značajno. Promenljiva ϵ se koristi da bi se izbeglo deljenje nulom. Mana ovog algoritma je što su ažuriranja u praksi vrlo agresivna i učenje se zaustavlja brzo. Jedan od razloga je čuvanje cele istorije treninga, koji je uzrok značajnom smanjivanju koraka učenja [25].

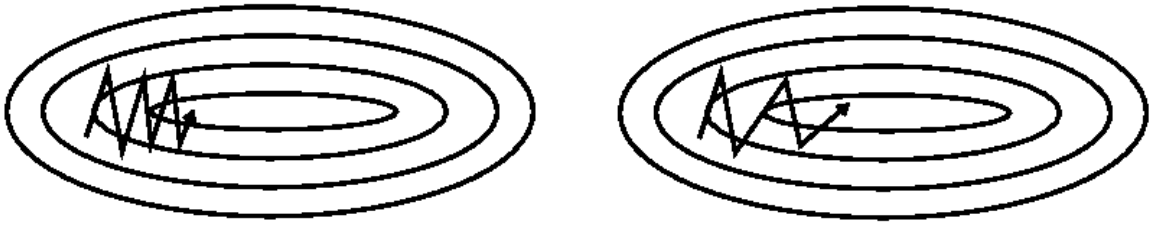
Algoritam Adadelta[30] (i njemu sličan RMSProp[25]) rešava ovaj problem tako što indirektno ograničava broj epoha koje se čuvaju na parametar w. Takođe ne čuva se eksplicitno w prethodnih gradijenata, već je suma rekurzivno određena kao aproksimacija srednje vrednosti svih prethodnih gradijenata:

$$\Delta_t = \gamma \Delta_{t-1} + (1 - \gamma) G_t^2$$
$$\Theta_{t+1} = \Theta_t - \frac{\eta}{\sqrt{\Delta_t} + \epsilon} G_t$$

γ je parametar koji uzima vrednost između $[0.9, 0.999]$. Na promenljivu Δ_{t+1} se ne dodaje nova vrednost kao u Adagrad algoritmu, već se rekurzivno dodeljuje suma umanjene vrednosti i novi parametar normalizacije.

Još jedno poboljšanje u odnosu na Adadelta je algoritam Adam[14], koji uz adaptivni gradijent dodaje i adaptivnu inerciju, koja pomaže bržoj konvergenciji. Naivno praćenje gradijenta pravi problem u situacijama kada je površ nakrivljena po jednoj dimenziji više nego po drugoj, a lokalni minimum je u pravcu manjeg pada. Tada obični gradijentni spust osciluje po dimenziji većeg pada, a malo napreduje u pravcu manjeg pada kao što je prikazano na slici 2.3. Takve površi su česte pri treniranju veštačkih neuronskih mreža [23]. Inercija je pojam iz fizike koji se na sličan način koristi i u optimizaciji. Treniranje se može posmatrati kao kretanje loptice po površi. Cilj optimizacije je da se loptica dovede do tačke na površi sa najmanjom potencijalnom energijom. Smer najbržeg rasta potencijalne energije u okolini loptice se dobija gradijentom funkcije površi u tački gde se loptica trenutno nalazi. Loptica se kreće u suprotnom pravcu, i tako dobija kinetičku energiju, to jest brzinu. U prirodi,

loptica koja pada niz nizbrdicu dobija neko ubrzanje, i njena brzina se uvećava.U prirodi loptica koja ima ubrzanje ne može promeniti smer naglo već menja vektor brzine u skladu sa prethodnim i trenutnim vektorom brzine. Sličan proces se može iskoristiti i za optimizacioni algoritam tako da se računa vektor pravca brzine kao linearna kombinacija prethodnih i trenutnog gradijenta, kao što je prikazano na slici 2.4.

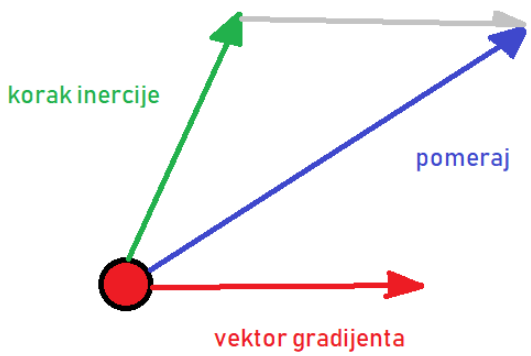


Slika 2.3: Ažuriranje sa i bez korišćenja inercije. Leva slika ilustruje ažuriranje bez inercije, desna slika ilustruje ažuriranje sa inercijom. Inercija pri menjanju parametara pomaže bržem ažuriranju po pravcu prema kome je lokalni minimum sa manje oscilacija [23].

Ovakav koncept se uključuje, uz adaptivna ažuriranja u algoritam Adam, čiji je uprošćeni algoritam prikazan u formuli:

$$\mu_t = \beta_1 \mu_{t-1} + (1 - \beta_1) G_t$$
$$\nu_t = \beta_2 \nu_{t-1} + (1 - \beta_2) G_t^2$$
$$\Theta_{t+1} = \Theta_t - \frac{\eta}{\sqrt{\nu_t} + \epsilon} \mu_t$$

Promenljiva μ_t je procena gradijenta uz korišćenje prethodnih gradijenata. Smatra se da korišćenje prethodnih gradijenata čini algoritam otpornijim na šum, jer je gradijent u tački osetljiv na šum, dok je korišćenje ovakvog uprosečavanja otpornije. Promenljiva μ_t je i način procene prvog statističkog momenta, to jest srednje vrednosti u pređenim tačkama. Promenljiva ν_t je ažuriranje adaptivnog koraka učenja slično Adadelta algoritmu, kao i aproksimacija drugog statističkog momenta, to jest varijanse. U kontekstu opisanog fizičkog problema, promenljiva μ_t bi bila brzina loptice, a promenljiva ν_t ubrzanje. Preporučene vrednosti za hiperparametre su $\epsilon = 10^{-8}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$.



Slika 2.4: Ažuriranje sa korišćenjem inercije. Pomeraj se računa kao neka vrsta vektorskog zbira gradijenta i „koraka inercije", to jest prethodnih koraka [23].

U praksi se trenutno preporučuje algoritam Adam, i većina programskih paketa nudi Adam kao podrazumevani algoritam optimizacije. U ovom radu se takođe najbolje pokazao algoritam Adam, dolazeći i do najboljeg rešenja i u najkraćem vremenskom roku.

Glava 3

Smanjenje dimenzionalnosti

Neuronske mreže, kao i mnogi drugi modela mašinskog učenja, često koriste atribute koji su napravljeni od strane eksperata u domenu iz kojeg se dobijaju podaci. Primeri ekspertskih algoritama za dobijanje atributa iz slika uključuju algoritme za nalaženje ivica i ćoškova, pomoću različitih matematičkih metoda koje nalaze oštre promene u intenzitetu boja, merenjem gradijenta i korišćenjem različitih filtera. Ovakvi algoritmi su zavisni od domena problema i tipa problema koji se rešava.

Postoje algoritmi za smanjenje dimenzionalnosti koji rade nezavisno od problema koji se rešava, dok god se ispunjavaju osnovni uslovi (na primer, za većinu algoritama ovog tipa atributi moraju biti numerički). Ovakvi algoritmi posmatraju samo vrednosti atributa instanci podataka i imaju za cilj odstranjivanje korelacije u podacima. Algoritmi koji se koriste u ovom radu pripadaju ovoj kategoriji.

Postoje dva pristupa smanjenju dimenzionalnosti:

- Selekcija atributa (*eng. Feature selection*)
- Ekstrakcija atributa (*eng. Feature extraction*).

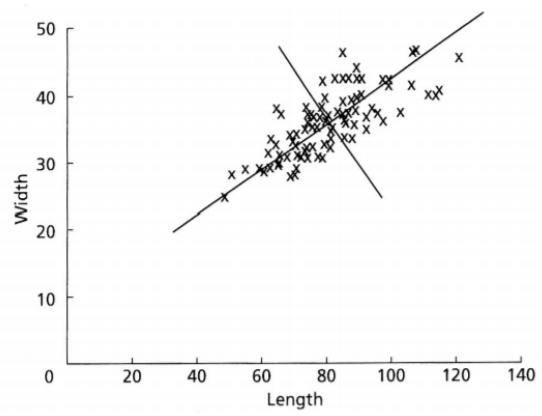
Metode selekcije atributa imaju za cilj izbacivanje manje važnih, odnosno manje diskriminativnih atributa, i zadržavanje bitnih iz početnog skupa. Da bi ovakve metode uspele, osnovni skup atributa mora da sadrži atribute bez kojih će skup sadržati približno istu količinu relevantnih informacija kao i početni. Takva situacija može da se desi ili ako je neki atribut izveden na neki način od drugog, ili ako je sam atribut nebitan za predstojeći zadatak mašinskog učenja ili vizualizacije[24].

Metode ekstracije atributa imaju za cilj da stvore nove atribute koje sadrže istu količinu korisnih informacija, ali tako da stvoreni atributi budu u manje dimenzionom prostoru od polaznih. Obe metode korišćene u ovom radu pripadaju klasi metoda ekstrakcije atributa, jer i analiza glavnih komponenti i autoenkoderi stvaraju nove atribute nastojeći da sačuvaju informacije sadržane u polaznom skupu. U nastavku će biti opisane obe metode.

3.1 Analiza glavnih komponenti

Analiza glavnih komponenti (*eng. Principal Component Analysis, PCA*) je metoda smanjenja dimenzionalnosti koja se oslanja na metode linearne algebre. Metodu je prvi put predstavio Karl Pirson 1901. godine[12]. Zadatak analize glavnih komponenti je da konvertuje skup instanci sa mogućim koreliranim atributima u skup instanci sa nekoreliranim atributima. Ovakvi nekorelisani atributi se nazivaju glavne komponente.

Analiza glavnih komponenti postavlja koordinatne ose u pravcima najveće varijabilnosti. Na taj način se intuitivno čuva najviše informacija. Glavne komponente zavise samo od matrice kovarijanse polaznih atributa. Tako je 1. glavna komponenta u pravcu najveće varijabilnosti skupa, 2. glavna komponenta je ortogonalna na nju, i takođe je u pravcu najveće varijabilnosti od svih ortogonalnih pravaca. Ovakav koncept je prikazan na specijalnom slučaju sa 2 dimenzije na slici 3.1.



Slika 3.1: Primer izvršavanja analize glavnih komponenti na primeru sa 2 dimenzije, visina i širina[11].

Postupak za računanje glavnih komponenti se sastoji od sledećih koraka.

- Računa se matrica kovarijanse po formuli:

$$cov = \frac{1}{n}(x - \mu)(x - \mu)^T,$$

  gde je x matrica koja se dobija tako što se vektori atributa instanci skupa slože jedan ispod drugog u matricu. Matrica μ se dobija tako što se izračunaju srednje vrednosti svih atributa na ulaznom skupu, a zatim se te vrednosti slažu u matricu dimenzija $m \times n$, gde je n broj atributa, a m broj instanci, tako da ceo i-ti red matrice ima vrednost srednje vrednosti i-tog atributa na tom skupu.

- Računaju se sopstvene vrednosti i sopstveni vektori matrice kovarijanse rešavanjem matrične jednačine $Ax = \lambda x$, tako da se dobije n sopstvenih vrednosti $\lambda_1, \lambda_2 \ldots \lambda_n$, i odgovarajući sopstveni vektori $v_1, v_2, \ldots v_n$, pri čemu važi $\lambda_1 > \lambda_2 > ... > \lambda_n$.

- Prva glavna komponenta odgovara sopstvenom vektoru koji odgovara najvećoj sopstvenoj vrednosti. Druga glavna komponenta odgovara sopstvenom vektoru koji odgovara sopstvenoj vrednosti koja je druga po redu po veličini. Svaka sledeća glavna komponenta se dobija na isti način.

U zavisnosti od izabrane dimenzionalnosti prostora novih, nekorelisanih atributa, bira se prvih k glavnih komponenti.

Zanimljiva osobina smanjenja dimenzionalnosti korišćenjem analize glavnih komponenti je da od svih linearnih transformacija atributa koje su moguće, ona ima najmanju kvadratnu grešku rekonstrukcije. Rekonstrukcija se dobija primenjujući inverznu transformaciju na rezultat dimenzionalne redukcije.

Za veće količine podataka i veliku količinu atributa, računanje sopstvenih vektora i sopstvenih vrednosti zahteva veliku količinu resursa i, bez značajnih izmena, zahteva rad sa matricama u memoriji, što predstavlja ograničavajući faktor. Postoje iterativne varijante analize glavnih komponenti koje se ne oslanjaju na dekompoziciju matrica, koje često koriste metodu dekompozicije singularnim vrednostima (*eng. singular value decomposition, SVD*). Neke od ovih metoda su QR algoritam, Jakobijev metod i metod stepenovanja. Metoda dekompozicije singularnim vrednostima se češće koristi u praksi za implementaciju, jer je numerički stabilnija i ne zahteva izračunavanje matrice kovarijanse, što je poželjno kada je veličina uzorka velika.

Za izvršavanje dimenzionalne redukcije nad slikama primenjuje se i tehnika lokalne analize glavnih komponenti (*localPCA, patchPCA*), gde se primenjuje klasična analiza glavnih komponenti nad delovima slike umesto nad celom slikom. Ovakav pristup koristi pretpostavku o prostornoj korelaciji piksela slike.

Mana analize glavnih komponenti i njenih varijanti je da, zbog toga što su bazirane na metodama linearne algebre, transformacije iz početnih atributa u nove atribute su linearne, što predstavlja ograničenje koje autoenkoderi nemaju. Postoje i verzije analize glavnih komponenti koje koriste nelinearne transformacije, kao što je *kernelPCA*, koje koriste prostor funkcija (Hilbertov prostor) umesto linearnog prostora za dekorelaciju atributa.

3.2 Autoenkoderi

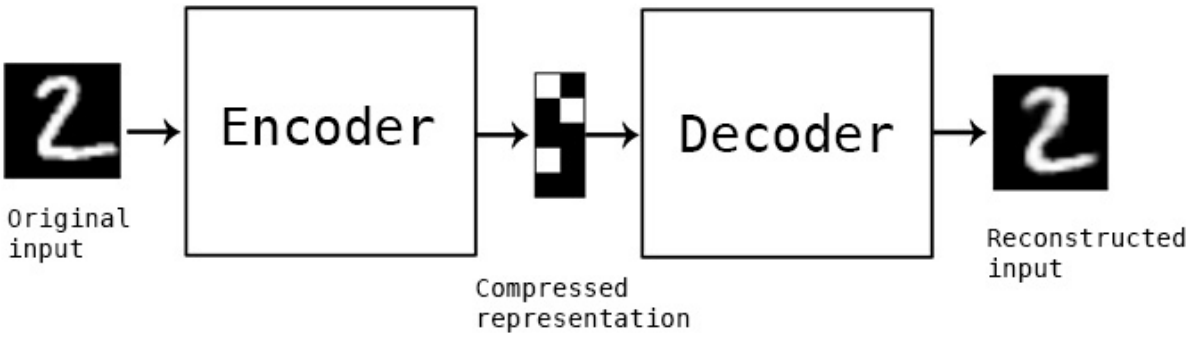
Autoenkoder je neuronska mreža koja je trenirana da približno kopira svoj ulaz na izlazni sloj. Autoenkoder ima skriveni sloj čiji se izlaz koristi za dimenzionalnu redukciju ulaza. Izlaz tog skrivenog sloja se naziva latentna reprezentacija. Mreža se može podeliti na dva dela: enkoder i dekoder. Enkoder predstavlja funkciju $h = f(x)$ koja slika ulaz x u latentnu reprezentaciju h, a dekoder funkciju $g(h)$ koja pokušava da vrati latentnu reprezentaciju h u ulaz x. Kad bi autoenkoder „naučio" da nađe za svako x, $g(f(x)) = x$ tada ovakva mreža ne bi bila od koristi. Umesto toga se autoenkoderi dizajniraju tako da nemaju kapacitet da nauče domen x savršeno. Pošto je model ograničen, on mora da prioretizuje koje komponente ulaza će kopirati, pa često nauči korisna svojstva podataka[7]. Shema autoenkodera je prikazana na slici 3.2.

Autoenkoderi se mogu posmatrati kao bilo koja neuronska mreža sa propagacijom unapred, dakle mogu se koristiti sve tehnike za treniranje, propagacija unazad i slično. Autoenkoderi se takođe mogu trenirati koristeći recirkulaciju (*eng. recirculation*), algoritam za učenje koji se bazira na upoređivanju aktivacija mreže na ulazu i rekonstruisanom ulazu, ali u ovom radu se takva tehnika neće koristiti[9]. Postoje dva načina za ograničavanje autoenkodera:

- Nepotpuni autoenkoderi (*eng. undercomplete*) – autoenkoderi koji imaju manju dimenzionalnost latentne reprezentacije (broja skrivenih neurona u najmanjem skrivenom sloju) od dimenzionalnosti ulaza. Smanjivanjem reprezentacije forsira se vađenje samo najbitnijih osobina ulaza. Proces učenja je opisan minimizovanjem funkcije gubitka $L(x, g(f(x)))$, gde je L funkcija „različitosti" u odnosu na x.
- Prepotpuni autoenkoderi (*eng. overcomplete*) – kada je dimenzija reprezentacije veća ili jednaka dimenziji ulaza koriste se metode regularizacije. Umesto ograničavanja dimenzije reprezentacije, uvode se drugačije restrikcije, kao što je proređenost reprezentacije, ograničavanje veličine izvoda i slično. U ovom radu se neće koristiti regularizovani autoenkoderi.

Za nepotpuni autoenkoder, kada je dekoder sadrži neurone sa linearnom funkcijom aktivacije i L je srednja kvadratna greška (*eng. mean squared error*), autoenkoder koristi isti potprostor kao analiza glavnih komponenti. Autoenkoderi sa nelinearnim funkcijama enkodera i dekodera mogu da nauče mnogo moćnije nelinearne generalizacije analize glavnih komponenti. Sa druge strane, ako su enkoderu ili dekoderu dozvoljeni veliki kapaciteti, autoenkoder može da nauči da vrši prosto kopiranje bez izvlačenja bilo kakve korisne informacije o raspodeli podataka. Takođe, moglo bi se zamisliti da bi autoenkoder sa jednodimenzionom latentnom reprezentacijom i veoma moćnim enkoderom teorijski mogao naučiti da predstavlja svaki trening uzorak $x^{(i)}$ latentnom reprezentacijom i. Ovakva situacija se retko dešava u praksi, ali dobro ilustruje šta se dešava kada se autoenkoderu dozvoli veoma veliki kapacitet[7].

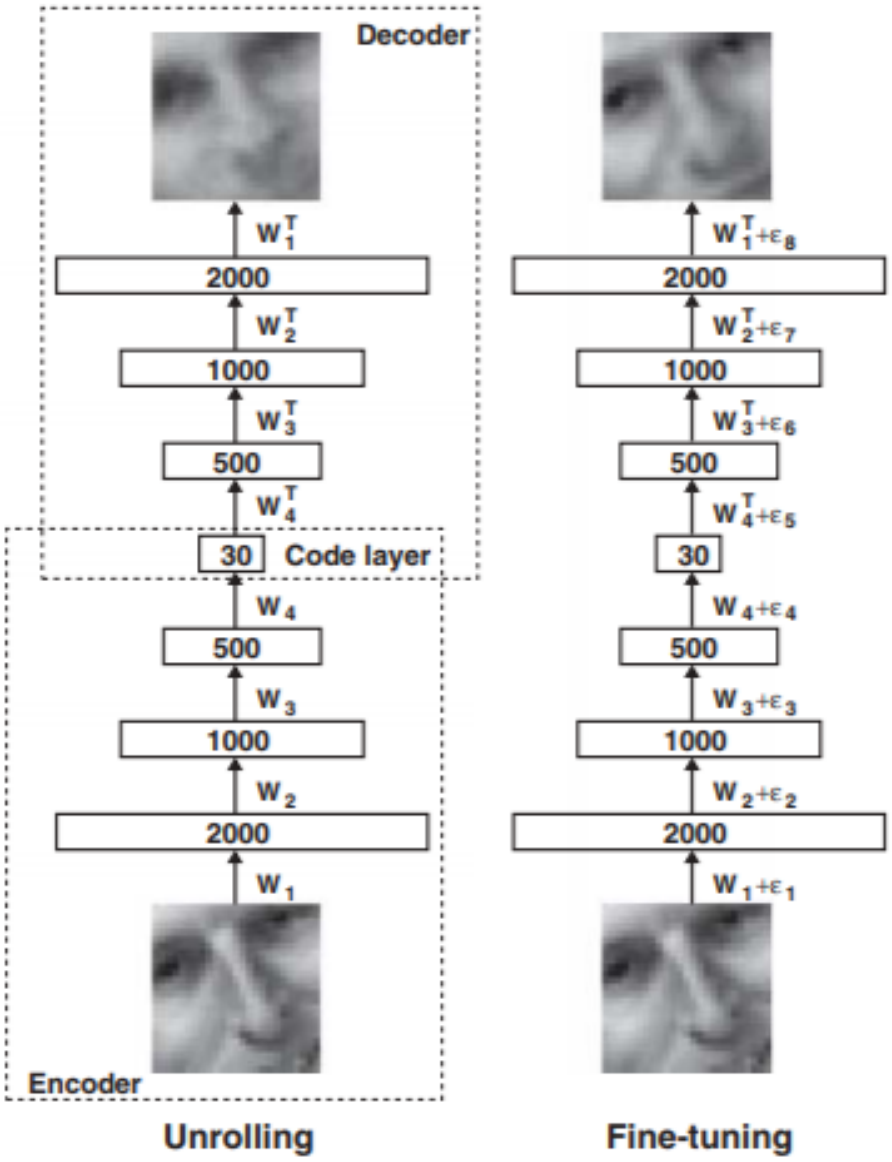
Dubina može eksponencijalno smanjiti cenu reprezentacije nekih funkcija. Takođe, dubina može eksponencijalno da smanji količinu potrebnih trening podataka za neke funkcije. Eksperimentalno, duboki autoenkoderi daju mnogo bolje rezultate pri kompresiji nego odgovarajući plitki ili linearni autoenkoderi. Shema verzije dubokog autoenkodera prikazana je na slici 3.3.



Slika 3.2: Shema autoenkodera na primeru ulaza iz MNIST skupa [3]

3.2.1 Primene autoenkodera

Autoenkoderi su uspešno primenjeni na probleme smanjenja dimenzionalnosti i pretraživanja informacija (*eng. information retrieval*). Problem smanjenja dimenzionalnosti je jedan od prvih problema za koji je iskorišćeno duboko učenje. G. Hinton i R. Salakhutdinov su trenirali red ograničenih Bolcmanovih mašina (*eng. Restricted Bolzman Machine,*



Slika 3.3: Shema autoenkodera predstavljena u radu Džefrija Hintona i Ruslana Salakhutdinova 2006. godine[10]

RBM), za pretreniranje dubokog autoenkodera sa latentnom reprezentacijom veličine 30. Rezultujući model je imao manju grešku rekonstrukcije od analize glavnih komponenti na dimenziju 30, a naučena reprezentacija je bila jednostavnija za razumevanje i razdvajanje u kategorije, koje su se manifestovale u dobro podeljene klastere. Reprezentacije podataka sa manje dimenzija mogu da poboljšaju performanse mnogih zadataka mašinskog učenja, kao što je na primer klasifikacija. Manjedimenzioni podaci znače manje trošenje resursa za računanje. Takođe primećeno je da neke forme redukcije dimenzionalnosti kodiraju semantički slične primerke blizu jedne do drugih.

Još jedan važan doprinos je i u problemu istraživanja informacija, gde zadatak nalaženja primeraka u bazi podseća na upit. Ovaj zadatak, osim običnih prednosti manjih dimenzija, takođe ima dodatnu prednost što su neke pretrage u određenim niskodimenzionim prostorim vrlo efikasne. Specijalno, algoritam za smanjenje dimenzionalnosti proizvodi binarni niskodimenzioni kod, svi unosi u bazu mogu se čuvati pomoću heš tabele. Takođe, u zavisnosti od reprezentacije, mogu se tražiti slični primerci malim promenama u reprezentaciji (inverzija

bitova i slično). Ovakav pristup smanjenja dimenzionalnosti i binarizacije se zove semantičko heširanje (*eng. semantic hashing*) i primenjen je uspešno na tekstualne podatke i slike.

Glava 4

Implementacija

Da bi se uporedile analiza glavnih komponenti i različiti autoenkoderi, korišćene su različite tehnologije. Za definisanje i trening autoenkodera je korišćena biblioteka Keras. Keras je biblioteka visokog nivoa koja je prilagođena za što lakšu implementaciju neuronskih mreža. Izgrađena je nad više biblioteka nižeg nivoa, među kojima i TensorFlow koji je izabran za ovaj rad. TensorFlow se bavi podacima i operacijama, a Keras opštom arhitekturom neuronskih mreža, tako da se kompletno povezani sloj u TensorFlow-u neuronske mreže definiše kao matrica težina koja se sastoji od čvorova operacija i promenljivih, a u Keras-u kao objekat ili funkcija sa ulazom i izlazom. Keras definiše objekte, koji za svoje funkcionalnosti koriste čvorove TensorFlow-a, ne opterećujući korisnika detaljnom implementacijom. U ovom poglavlju će biti opisane obe biblioteke.

Za implementaciju analize glavnih komponenti je korišćena biblioteka Scikit-learn, koja nudi različite algoritme, uključujući i pomenutu analizu glavnih komponenti i mnogo drugih algoritama. Scikit-learn je takođe biblioteka visokog nivoa, koja nudi i korisnicima koji nisu programeri da uspešno koriste statističke algoritme i algoritme mašinskog učenja. U listingu 4.1 je prikazan isečak kôda korišćenog za rad koji pokreće implementaciju analize glavnih komponenti.

Listing 4.1: Primer kôda za analizu glavnih komponenti u paketu scikit-learn.

```
1
2 from sklearn.decomposition import PCA;
3
4 #pravi se objekat pca sa 2 glavne komponente
5 #i primenjuje se na skupu a  koji je prethodno ucitan
6 pca=PCA(n_components=2).fit(a);
7
8 #skup se transformise u novi prostor
9 #pomocu funkcije transform
10 t=pca.transform(a);
```

4.1 TensorFlow

TensorFlow je biblioteka za numerička izračunavanja korišćenjem grafova toka podataka[1]. Originalno je razvijen u Guglovom odseku za mašinsko učenje na Google Brain projektu za potrebe mašinskog učenja. Poseduje fleksibilnu arhitekturu koja omogućava korišćenje jednog ili više procesora za izračunavanje, kao i izračunavanje na jednom ili više grafičkih procesora, servera, desktop računara i mobilnih uređaja kao što su tableti i pametni telefoni. Ovakva fleksibilnost se može iskoristiti bez menjanja kôda programa ili sa malim promenama, u zavisnosti od platforme i funkcije. Među zapaženim primenama su primene u zadacima:

- prepoznavanje govora,
- računarski vid,
- robotika,
- pronalaženje informacija,
- obrada prirodnog jezika,
- pronalaženje geografskih informacija.

TensorFlow biblioteka se razvila iz biblioteke DistBelief, koja je korišćena u mnogim proizvodima kompanije Gugl kao što su:

- Google Search,
- Google Translate,
- Youtube,
- Google StreetView,
- Google Maps,
- Google Photos.

Od 2011. godine, kada je razvijena, Ova biblioteka je korišćena za razne zadatke u okviru ovih proizvoda u kompaniji Gugl, kao što su:

- Problemi nenadgledanog učenja, gde je jedan od zanimljivijih primera uspešno testiranje regularizovanog autoenkodera 3 dana na 1000 računara sa ukupno 16000 jezgara u cilju dobijanja apstraktnog nivoa prepoznavanja lica. Motivacija je teorija o postojanju „baka" neurona (*eng. grandmother neuron*), to jest neurona ili skupa neurona koji reaguje na određena lica. Korišćen je skup podataka koji je dobijen iz nasumičnih frejmova uzetih iz nasumičnih Youtube video snimaka, u rezoluciji 200x200. Delovi reprezentacija ove mreže su zaista pokazali osetljivost na lica ljudi, mačaka, kao i tela ljudi, invarijantne u odnosu na translaciju i rotaciju. Ovakva reprezentacija je takođe korišćena za treniranje prepoznavanja objekata iz „Imagenet" skupa podataka, i dobijena je bolja preciznost u odnosu na trenutni najbolji sistem [16].

- Predstavljanje prirodnih jezika, gde je predstavljeno više različitih jezičkih modela za računanje neprekidnih vektora reprezentacije reči i veoma velikih skupova podataka. Zabeležen je veliki uspeh u preciznosti i manja složenost izračunavanja, u jednom primeru manje od jednog dana za učenje vektora reči iz skupa od 1,6 miliona reči. Neki od modela su „FeedForward Neural Net Language Model" -NNLM i „Reccurent Neural Network Language Model" - RNNLM koji su trenirani na „Google News" skupu podataka koji se sastoji od 6 milijardi reči. Trenirano je 50-100 replika modela paralelno pri čemu su replike bile posebno trenirane na posebnim procesorima [19].

- Modeli za klasifikaciju slika i prepoznavanje objekata na slici, jedan od takvih modela, „GoogLeNet" (kodno ime „Inception") je pobedio na „ImageNet Large Scale Visual Recognition Challenge 2014". Ova konvolutivna neuronska mreža je imala 27 slojeva, zajedno sa 5 agregaciona sloja, i trenirana je paralelno na slikama rezolucije 224x224. Implementacija ove konkretne mreže je dizajnirana za treniranje isključivo na centralnim procesorima, ali autori smatraju da se implementacija lako može izmeniti tako da podržava treniranje na grafičkim procesorima[5].

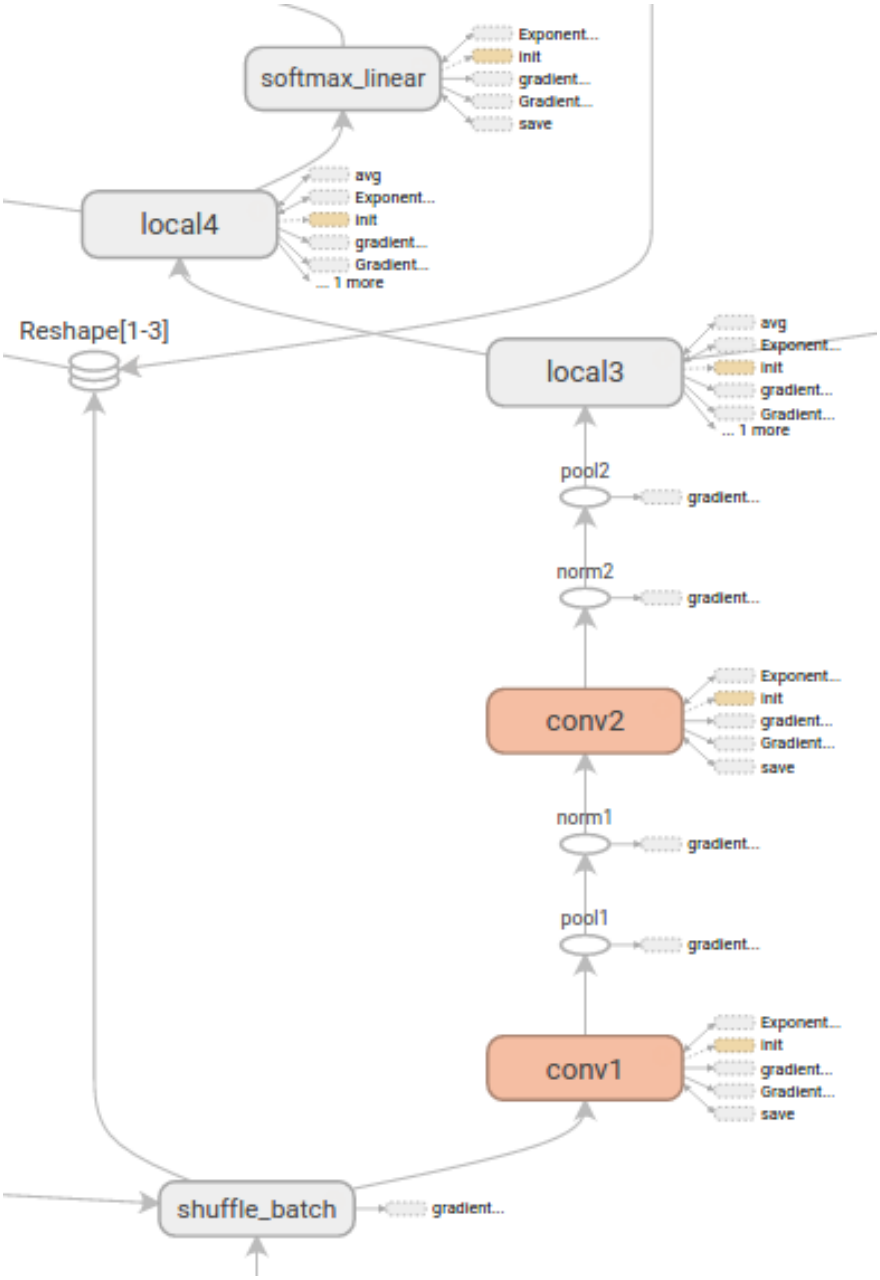
- Klasifikacija video fajlova, od kojih je jako zanimljiv primer neuronska mreža trenirana na skupu od milion Youtube sportskih video fajlova, koji se klasifikuju sa 487 klasa. Novina kod ove mreže su dva toka procesiranja, jedan kontekstni (*eng. context stream*) koji uči osobine video fajlova na nižim rezolucijama i visokorezolucioni „Fovea" tok podataka (*fovea stream*), koji uči osobine video fajlova na visokoj rezoluciji, ali samo u sredini slike (frejma). Treniranje je trajalo mesec dana, a nakon uspešnog treninga mreža je pokazala zadovoljavajuće rezultate i na drugim skupovima podataka sa sportskim video fajlovima, kao što je „UCF-101", koji ima 101 različitu klasu [13].

- Prepoznavanje govora, gde je testirana nova duboka mreža poverenja (*eng. Deep Belief Network*) u spoju sa skrivenim Markovljevim modelima(*eng. Hidden Markov Models*), na različitim zadacima prepoznavanja zvuka kao što su „Bing Voice Search Speech Recognition Task", gde je mreža trenirana 24 sata, i dobila je bolju preciznost od prethodnog najboljeg modela, „Google Voice InputSpeech Recognition Task", gde je zabeleženo poboljšanje od 23%, kao i „Youtube Speech Recognition Task", gde je mreža dala apsolutno poboljšanje od 4,7% u odnosu na prethodni najbolji model, kao i drugi slični zadaci[8].

- Predikcija sekvenci, gde je zapaženo korišćenje LSTM (*eng. Long Short-Term Memory*) neuronskih mreža za prevođenje engleskog jezika na francuski i obrnuto, uz zapažene rezultate na skupu podataka WMT-14, koji se sastoji od 148 miliona francuskih reči i 304 miliona engleskih reči. Treniranje ove mreže se vršilo na 8 grafičkih procesora, gde je svaki sloj mreže koristio posebnu grafičku karticu, u vremenskom periodu od 10 dana [26].

- Odabir poteza u igri Go, korišćenjem konvolutivnih mreža za učenje poteza profesionalaca. Ova mreža je dostigla rang sličan 6-dan ljudskom igraču, i dostigla rezultat sličan Monte Karlo algoritmu koji simulira milion pozicija po potezu [17].

- Učenje potkrepljivanjem (*eng reinforcement learning*), gde je testirana prva masivno paralelizovana i distribuirana arhitektura za duboko učenje potkrepljivanjem-DQN (*eng. Deep Q-Network Algorithm*). Ovaj algoritam je upotrebljen za učenje 49 igara iz „Arcade Learning Environment" skupa podataka, koji se sastoji od igara sa sistema Atari. [20]



Slika 4.1: Primer dela grafa toka podataka u TensorFlow-u. Na slici su prikazani čvorovi konvolucije, *conv*2 i *conv*1, čvorovi agregacije *pool*1 i *pool*2 kao i sloj za klasifikaciju $softmax\_linear$ i drugi sa odgovarajućim tokom podataka koji se sastoji od tenzora [1].

Izračunavanje korišćenjem TensorFlow biblioteke je izraženo usmerenim grafom toka podataka, koji predstavlja izraz koji se izračunava kao što je prikazano na slici 4.1. Ovakav graf se opisuje pomoću programskog jezika, kao što je C++ ili Python. Svaki čvor u ovakvom grafu ima nula ili više ulaza i nula ili više izlaza i predstavlja instancu operacije. Svaka operacija ima ime i atribute koji se zadaju pri konstrukciji grafa. Grane grafa su tenzori koji se koriste kao argumenti operacija. Tenzor je višedimenzionalna struktura, i matrica je specijalan slučaj reda 2. Tip tenzora se zna u trenutku konstrukcije grafa, i ne menja se tokom izvršavanja. Takođe postoje specijalne grane grafa koje ne predstavljaju tok podataka već opis stanja i informacije vezane za kontrolu toka.

Specijalan tip čvora grafa u TensorFlow-u je tip promenljive koji sadrži referencu na mutabilni tenzor. Postoje specijalne operacije `Assign`, koji odgovara operatoru `=`, i `AssignAdd`, koji odgovara `+=`, za menjanje referenciranih promenljivih. Promenljiva ima tip podatka, i dimenzionalnost tenzora, u ovom kontekstu nazvan oblik (*eng. shape*), koji joj se moraju dodeliti prilikom inicijalizacije čvora. Tip i oblik promenljive se ne mogu menjati nakon inicijalizacije promenljive.

Klijentski programi koriste TensorFlow stvaranjem sesije. Sesija se vezuje za jedan graf toka podataka i poseduje funkciju `Run` koja uzima imena čvorova čiji rezultati treba da se dobiju i opcione ulaze koji mogu da se koriste umesto nekih čvorova u grafu. TensorFlow obilazi graf operacija prilikom izvršavanja funkcije `Run` i vraća rezultate kada dodje do traženih čvorova. Sesija čuva sve vrednosti promenljivih, a svi čvorovi koji nisu tipa promenljiva se izračuvanavaju pri pozivu. Postoje čvorovi koji predstavljaju ulaz TensorFlow sesije, tipa *placeholder*, čija vrednost mora da se obezbedi prilikom izvršavanja funkcije `Run`. Funkcija `Run` poseduje opcioni parametar za prihvatanje ulaza, pod nazivom $feed\_dict$, koji prima rečnik čvorova koji imaju tip *placeholder* ili su promenljive, gde je ključ u rečniku TensorFlow ime čvora, a vrednost je vrednost promenljive, koja mora da odgovara tipu i obliku tih čvorova. Imena čvorova koji se traže na izlazu se prosleđuju preko parametra $fetches$, čiji je tip lista niski.

4.2 Keras

Keras je biblioteka za duboko učenje visokog nivoa koja za izračunavanje koristi biblioteke nižeg nivoa kao što je TensorFlow[3]. Visok nivo podrazumeva da Keras pruža funkcionalnosti koje podržavaju stvaranje modela sa fokusom na ideji, a ne na konkretnoj implementaciji. Primarni programski jezik koji Keras koristi je Python. Keras je originalno napravljen nad bibliotekom Theano, ali je ubrzo dodata podrška za TensorFlow. Biblioteke za mašinsko učenje koje danas podržava Keras su Theano, TensorFlow i Microsoft CNTK.

Fokus je na ideji o modelu, a osnovni principi na kojima počiva Keras su:

- Lakoća korišćenja – Keras prati prakse smanjivanja kognitivnog napora, pružajući konzistentan i jednostavan API, i minimizuje broj akcija korisnika koje su potrebne za uobičajene slučajeve. Takođe pruža korisne i jasne informacije o grešci.
- Modularnost – model se posmatra kao niz ili graf samostalnih konfigurabilnih modula koji mogu da se povezuju sa najmanje restrikcija.
- Mogućnost proširenja – Novi moduli se lako dodaju kao nove klase i funkcije, i na isti način se kombinuju i prave kao postojeće klase. Takođe, modularnost Kerasa čini da novi moduli mogu lako da se koriste uz postojeće.
- Python – Keras koristi samo jezik Python za opis modela i ne koristi nikakve dodatne fajlove ili specijalne konfiguracione jezike. Ovako Keras pruža kompaktnost, proširivost i jednostavnije debagovanje.

Keras je zasnovan na principima objektno orjentisanog programiranja. U teoriji se modeli dubokog učenja često posmatraju kao nizovi funkcija, što se intuitivno prenosi na funkcionalnu paradigmu, sa druge strane, takve funkcije bi imale ogromnu količinu parametara koji

se prenose, i manipulacija ovim parametrima bi bila i neprijatna za rad i neefikasna sa tačke gledišta efikasnosti izvršavanja. Zato je u Kerasu svaka funkcionalnost, kao što su modeli, slojevi modela, optimizatori i druge predstavljena objektom. Svim parametrima modela može se pristupiti preko svojstava objekata modela.

Modelovanje arhitekture modela u Kerasu se zasniva na sekvenci slojeva modela. Keras ima svoju strukturu grafa toka podataka nezavisno od biblioteke nižeg nivoa koja se koristi, što čini deljenje modela i kopiranje prostijim, i menjanje biblioteke se obavlja bez velikih menjanja kôda definicije modela. Shema arhitekture modela na primeru LSTM mreže je prikazana na slici 4.2. Keras trenutno podržava dva tipa modela, koji se razlikuju po načinu definisanja. Prvi tip modela je sekvencijalni model (*eng. Sequential*), koji se definiše kao linearni niz slojeva. Može se definisati prosleđivanjem liste slojeva konstruktoru, ili dodavanjem slojeva modelu pomoću funkcije `Add`. Drugi tip modela je funkcionalni model (*eng. Keras functional API*). Funkcionalni model se definiše funkcionalnom paradigmom, i dobar je za definisanje kompleksnih modela sa više izlaza, sa deljenim slojevima, ili definisanje usmerenih acikličnih grafova.

Keras pruža funkcionalnosti modela mašinskog učenja preko metoda klase `Model`. Neke od ovih metoda su:

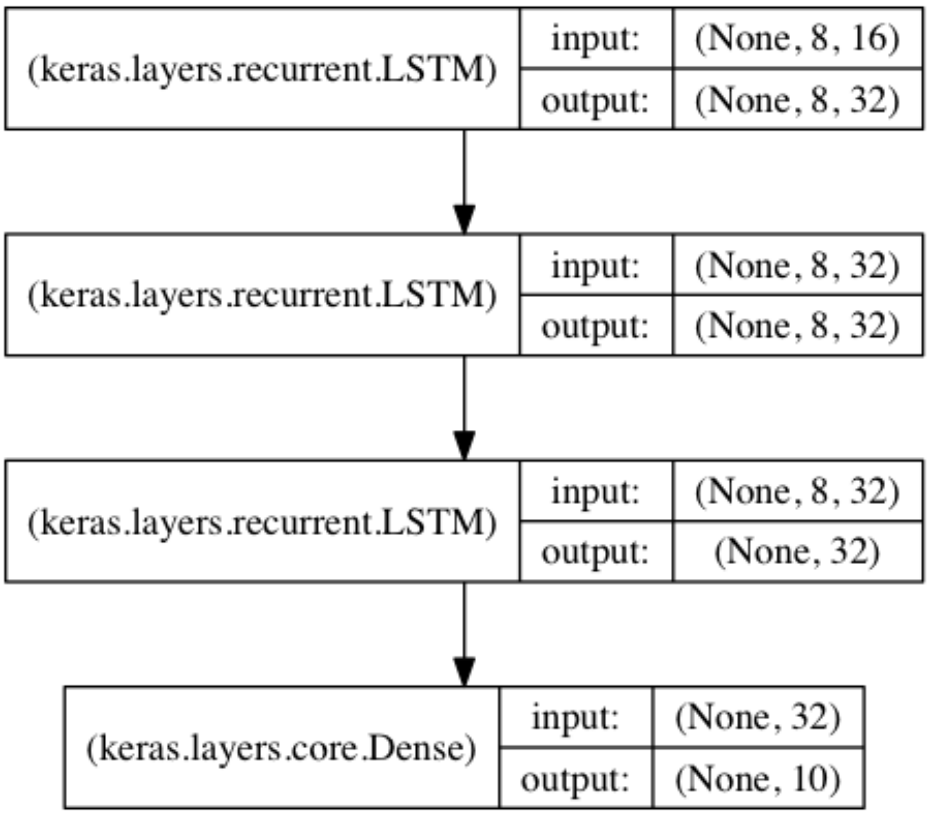
- `compile` - konfiguriše model za trening. Obavezni parametar je tip optimizatora sa svojim parametrima. Ovaj parametar je objekat klase `Optimizer`, koji u zavisnosti od tipa optimizatora ima svoje parametre. U Kerasu takođe postoji opcija unošenja parametra pomoću niske, tako da se unošenjem parametra `"SGD"`, optimizator postavlja na stohastički gradijentni spust sa podrazumevanim parametrima.
- `fit` - trenira model na podacima za trening. Među parametrima za trening su `x` i `y`, koji označavaju ulaz i očekivani izlaz respektivno, `batch_size`, koji označava dimenziju podskupa treniranja, kao i `epochs`, koji označava broj epoha treninga. Ova metoda takođe podržava razdvajanje skupa na skup za trening i skup za validaciju (ili kao procenat podele - `validation_split`, ili kao poseban parametar za skup za validaciju - `validation_data`), kao i podešavanje matrice greške za klasifikaciju preko parametra `class_weight`. Postoji opcija `shuffle`, za čiju vrednost `True`, Keras nasumično permutuje trening skup pri svakoj epohi. Keras nudi mogućnost reagovanja na događaje kao što su početak nove epohe, kraj epohe, izračunavanje greške i slično, preko parametra `callbacks`, koji prima listu objekata koji sadrže metode koje se zovu pri pomenutim događajima. Sekvenca treniranja je prikazana na slici 4.3.
- `predict` - primenjuje naučeni model na skupu koji se zadaje pomoću parametra `x`.

Keras posmatra bilo koju transformaciju podataka kao sloj u arhitekturi mreže. Na primer, kompletno povezani sloj je Keras sloj u mreži, sa svojom funkcijom aktivacije, regularizacijom i ograničenjima, ali su i slojevi aktivacije i regularizacije takođe slojevi sami po sebi. Tako se može napraviti jedan kompletno povezani sloj pomoću nekoliko slojeva u kontinuitetu, počev od kompletno povezanog sloja sa linearnom funkcijom aktivacije, praćen slojem aktivacije, praćen slojem regularizacije. Neke metode preprocesiranja i obrade podataka su takođe dostupne u obliku slojeva. Neki od slojeva koji su dostupni u Kerasu su:

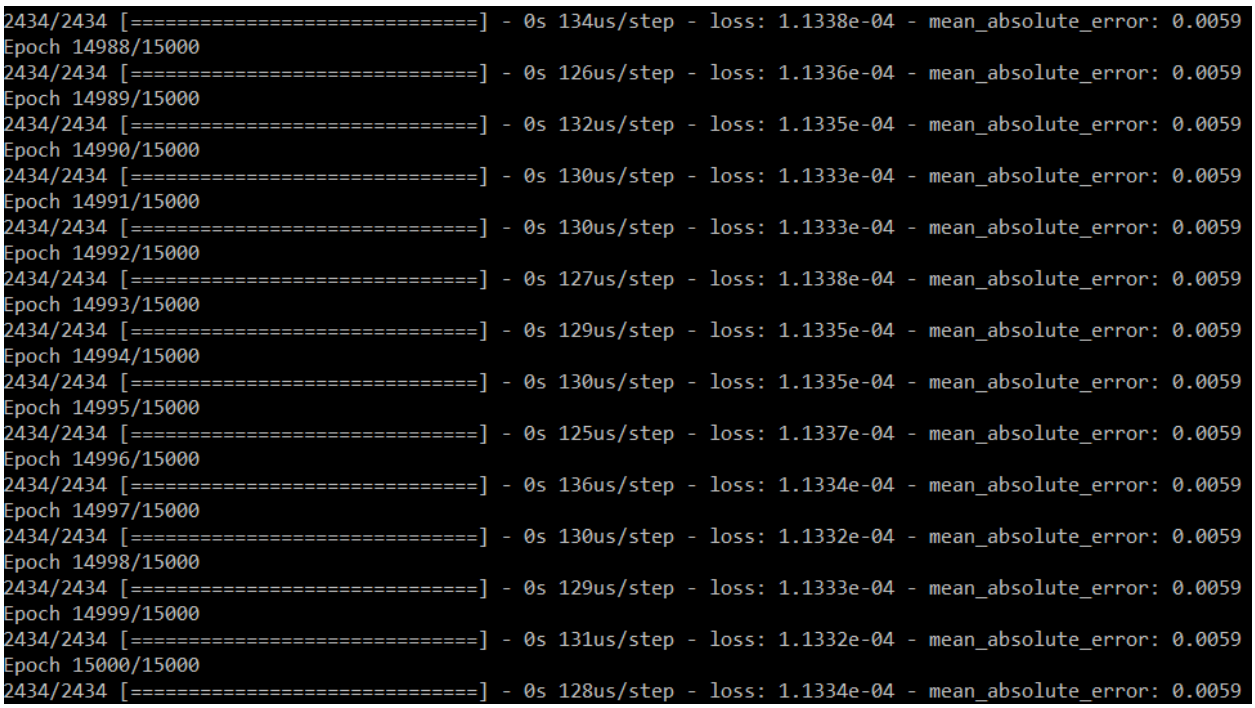
- `Input` - sloj ulaza u neuronsku mrežu. Ovaj sloj transformiše ulazne podatke u Keras

tenzor, koji predstavlja tenzor biblioteke koju koristi keras (TensorFlow, Theano ili CNTK). Konstruktor ima opcioni parametar `shape` koji sadrži oblik ulaznog tenzora.

- `Dense` - regularni kompletno povezani sloj neuronske mreže. Obavezni parametar konstruktora ovog tipa sloja je broj neurona u sloju pod nazivom `units`. U konstruktoru se takođe prosleđuje i tip funkcije aktivacije pomoću parametra `activation`. Podrazumevana vrednost za ovaj parametar je linearna funkcija aktivacije, ali su dostupne sve funkcije aktivacije prikazane u ovom radu, kao i mnoge druge. Inicijalizacija težina se može podešavati pomoću parametra `kernel_initializer`. Dostupne su i mogućnosti regularizacije. Ograničenja težina se postavljaju pomoću parametra `kernel_contraint`, gde se mogu postaviti ograničenja kao što su nenegativnost i maksimalna vrednost.
- Konvolutivni slojevi - među kojima su i slojevi: `Conv1D`, `Conv2D`, `Cropping2D`, `UpSampling1D` i `UpSampling2D`. Obavezni parametri konstruktora su: `filters`, koji odgovara broju konvolutivnih filtera i `kernel_size`, koji odgovara obliku kernela. Može se podesiti korak konvolucije pomoću parametra `stride`, kao i tip inicijalizacije težina. Konstruktor takođe ima parametar `padding`, koji nudi opciju popunjavanja matrice nulama na odabrani način.
- Slojevi rekurentne mreže, među kojima su `SimpleRNN`, `GRU` i `LSTM`.
- `Flatten` - transformiše podatke u jednodimenzioni niz. Ima samo jedan parametar, i to je parametar `data_format`, koji sadrži informacije o redosledu dimenzija.



Slika 4.2: Primer dizajna modela u Kerasu. Dizajnirana je rekurentna LSTM(*Long Short Term Memory*) pomoću četiri sloja od kojih su prva tri LSTM slojevi, a poslednji je gusto povezani sloj [3].



Slika 4.3: Primer izvršavanja treninga u kerasu.

4.3 scikit-learn

Scikit-learn je paket za programski jezik Python koji pruža alate za jednostavno korišćenje metoda mašinskog učenja i istraživanja podataka koji koristi NumPy, SciPy i matplotlib pakete za implementaciju algoritama[21].

Projekat je započeo Dejvid Kornapo (*David Cournapeau*) na Guglovom „Summer of code" događaju 2007. godine. Kasnije te iste godine je Metju Bruše (*Matthieu Brucher*) nastavio rad u okviru svoje teze. Februara 2010. godine je publikovana prva zvanična verzija pod vođstvom francuskog instituta za istraživanje u računarskim naukama i automatizaciji (INRIA). Od tada na svaka 3 meseca izlazi nova verzija paketa scikit-learn.

Alati sadržani u paketu su podeljeni u nekoliko grupa:

- Klasifikacija – koja sadrži implementacije algoritama kao što su metod potpornih vektora (*eng. Suport Vector Machine, SVM*), metoda najbližih suseda (*eng. Nearest neighbors*), slučajne šume (*eng. random forests*) i slično.
- Regresija – metoda najmanjih kvadrata, regresija potpornih vektora (*eng. Suport Vector Regression*), logistička regresija i slični.
- Klasterovanje – metoda k sredina (*eng. k-Means*), spektralno klasterovanje i slično.
- Smanjivanje dimenzionalnosti - analiza glavnih komponenti, nenegativna matrična dekompozicija (*eng. Non-negative matrix factorization, NMF, NNMF*)
- Selekcija modela – metrike, unakrsna validacija (*eng. cross validation*) i slično.
- Preprocesiranje – normalizacija, standardizacija podataka i slično.

Scikit-learn ima otvoreni kôd i pod BSD licencom je. Nekoliko popularnih firmi koristi scikit-learn za svoje potrebe, kao što su Spotify, Inria, Betaworks, Evernote, Booking, Change.org.

Glava 5

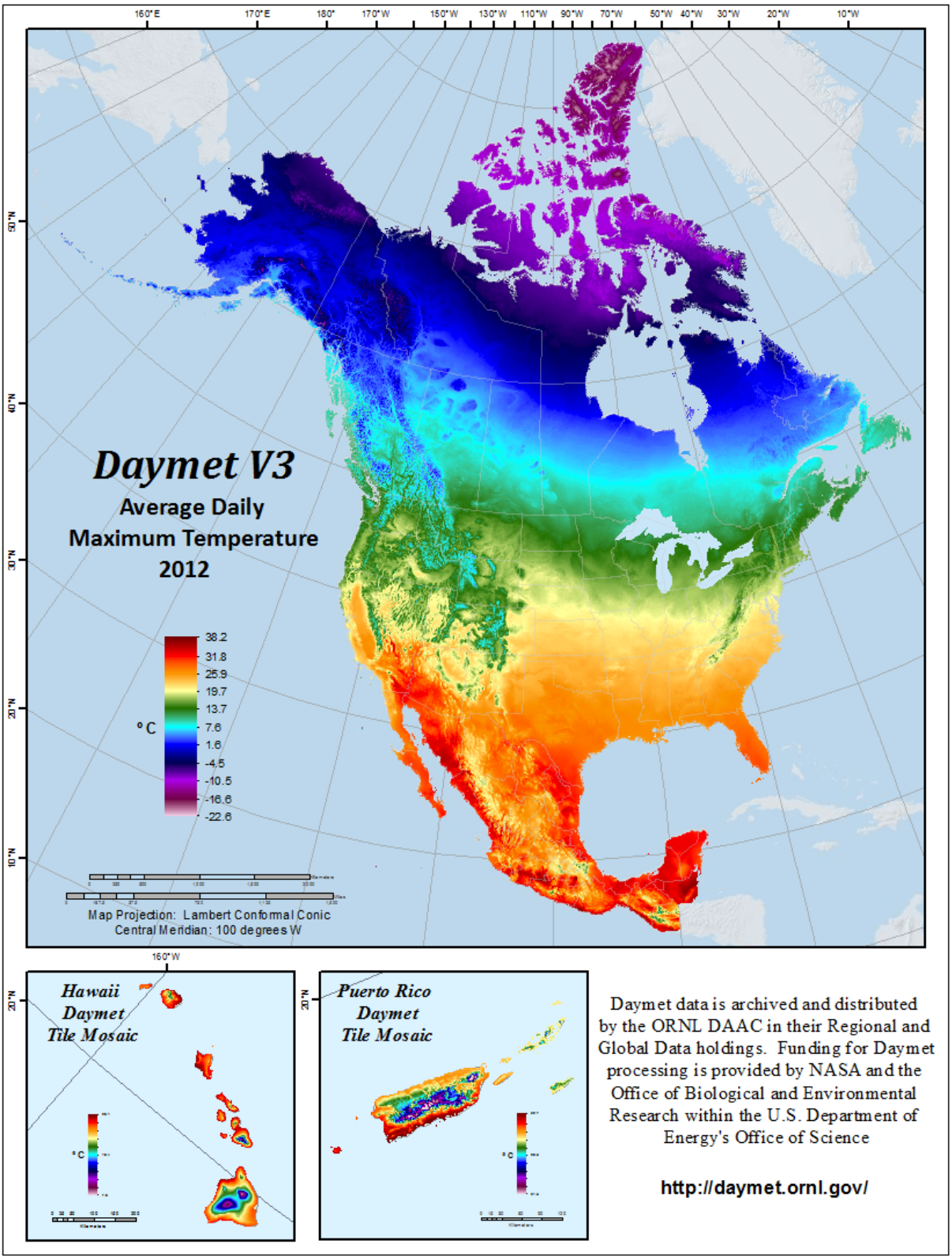
Eksperimentalni rezultati

U ovom poglavlju će biti prikazani rezultati eksperimenata, kao i sam opis eksperimenata, korišćenih modela i podataka. Korišćeni su podaci iz „Daymet" skupa podataka, koji je opisan u sledećem poglavlju. Izabrani prostorni podaci imaju interesantna svojstva, koja izazivaju neočekivane performanse nekih modela. Modeli koji se koriste su analiza glavnih komponenti, koja je opisana u poglavlju 3.1, i nekoliko arhitektura autoenkodera. Autoenkoderi su opisani u poglavlju 3.2, a koriste se plitke verzije autoenkodera sa *tanh* i *relu* funkcijama aktivacije opisanim u poglavlju 2.1, kao i jedna dubokoa arhitektura koja takođe koristi *tanh* funkciju aktivacije. Analiza glavnih komponenti je implementirana u paketu scikit-learn koji je opisan u poglavlju 4.3, a autoenkoderi su implementirani u biblioteci Keras, koja je opisana u poglavlju 4.2. Svi eksperimenti su izvršavani na laptop računaru sa procesorom $i7-7700HQ$ sa $16GB$ RAM memorije.

5.1 Korišćeni podaci

Za testiranje efikasnosti dizajniranih modela mašinskog učenja su korišćeni podaci o padavinama iz skupa podataka „Daymet". Skup podataka „Daymet" sadrži meteorološke podatke koji se odnose na Severnu Ameriku. Na slici 5.1 je prikazana vizualizacija podataka vezanih za temperaturu iz ovog skupa. Ovaj skup sadrži podatke za svaki dan u periodu od 1. januara 1980. godine do 31. decembra 2016. godine. Skup podataka „Daymet" ima sedam izlaznih parametara:

- *dayl* – Dužina obdanice u sekundama po danu (s/dan). Ovaj podatak je izračunat na osnovu perioda dana tokom koga je sunce iznad hipotetičkog horizonta.
- *prcp* – Ukupne dnevne padavine u milimetrima po danu (mm/dan). Suma svih padavina izražena kao sadržaj vode u padavini.
- *srad* – Gustina fluksa radijacije kratkih talasa izražena u vatima po kvadratnom metru (W/m^2), izračunata pomoću srednje vrednosti dužine obdanice u toku dana.
- *swe* – Količina vode sadržana u snegu, izražena u kilogramima po kvadratnom metru (kg/m^2).
- *tmax* – Dnevna maksimalna temperatura izražena u stepenima Celzijusa (C°).

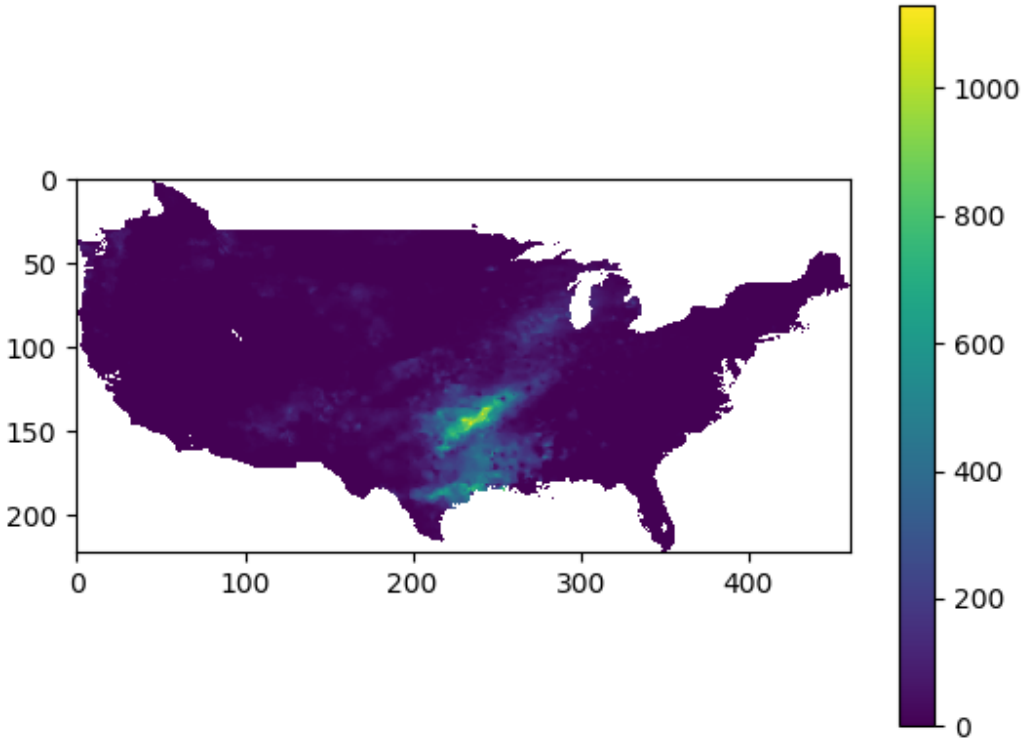


Slika 5.1: Primer vizualizacije podataka iz „Daymet" skupa. Srednja vrednost maksimalnih dnevnih temperatura za 2012. godinu. [15]

- *tmin* – Dnevna minimalna temperatura izražena u stepenima Celzijusa (C°).
- *tmin* – Pritisak vodene pare izražen u paskalima (Pa).

Korišćen je skup podataka koji sadrži količinu padavina na teritoriji kontinentalnih Sjedinjenih Američkih Država (bez Aljaske i države Havaji) u milimetrima iz „Daymet" skupa. Podaci su zapisani u obliku matrica koje mogu biti vizuelizovane kao slike gde jedan piksel odgovara jednom elementu matrice na poziciji koja odgovara poziciji u matrici. Pod prostornom rezolucijom se podrazumeva površina koju predstavlja jedan element matrice, odnosno piksel vizuelicije. Pod vremenskom rezolucijom se podrazumeva vremenski razmak dve uzastopne mape.

Prostorna rezolucija je smanjena da bi se ubrzao trening i smanjila hardverska zahtevnost, pa jedan piksel mape odgovara teritoriji od $12500km^2$. Teritorija van granica kontinentalnih Sjedinjenih Američkih Država je obeležavana kao nedostupna vrednost, odnosno NA, zbog intuitivnijeg tumačenja podataka i rezultata. Vizualizacija ovakvih podataka je prikazana na slici 5.2. Uzet je period od 1. januara 2006. godine do 31. decembra 2015. godine, a vremenska rezolucija je 1 dan.



Slika 5.2: Primer vizualizacije podataka korišćenih za treniranje modela. Korišćeni su prostorni podaci vezani za količinu padavina na teritoriji Sjedinjenih Američkih Država.

Izračunate statistike polaznog skupa su prikazane u tabeli 5.1. Može se videti da 50% podataka ima vrednost manju od $1,1$. Više od 37% podataka ima vrednost tačno 0. Iz tabele se takođe može videti da je maksimalna vrednost $1973,8$. Ovakva raspodela podataka predstavlja dodatni izazov za modele mašinskog učenja, koji često pretpostavljaju simetričnu raspodelu podataka.

Tabela 5.1: Tabela izračunatih statistika za skupove prostornih podataka izmerenih padavina na teritoriji Sjedinjenih Američkih Država.

statistika	*vrednost*
srednja vrednost (μ)	$20,71$
standardna devijacija (σ)	$49,67$
minimum (min)	0
maksimum (max)	$1973,8$
mediana (Q_2)	$1,1$
prvi kvartil (Q_1)	$0,0$
treći kvartil (Q_3)	$17,5$

5.2 Eksperimenti i analiza

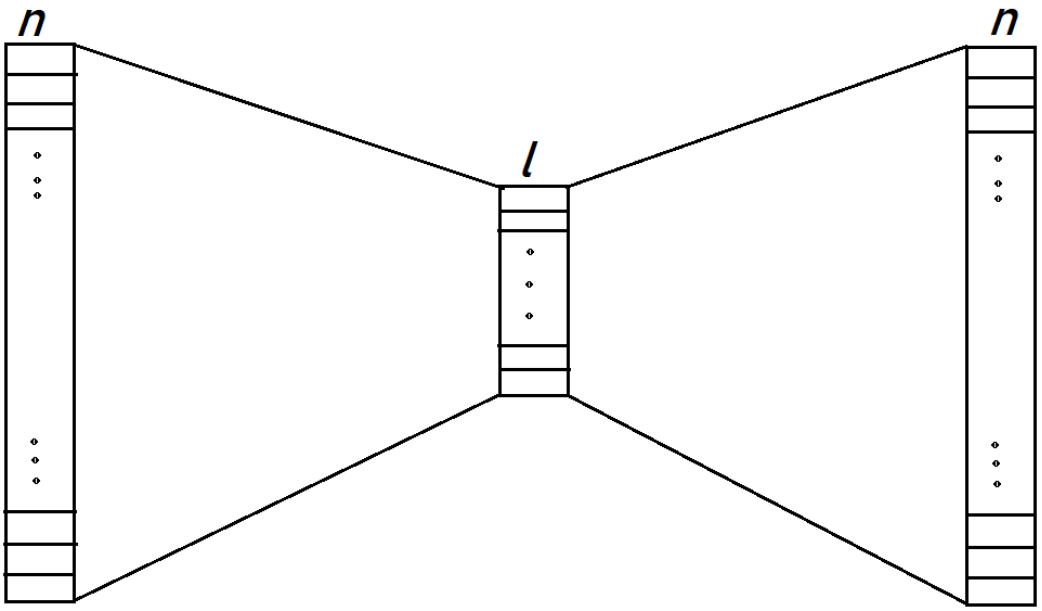
Da bi se podaci opisani u prethodnom poglavlju iskoristili za treniranje i testiranje modela prvi korak je odstranjivanje nedostupnih vrednosti i prebacivanje dvodimenzionalne matrice vrednosti u jednodimenzioni niz koji može da bude ulaz u model. Vrednosti se pakuju po redovima, dakle, i-ti piksel u j-tom redu će biti na poziciji $jn + i$. Ovakav niz može da se posmatra i kao niz atributa jedne instance koja odgovara jednoj mapi za jedan izmereni dan. Odstranjivanje nedostupnih vrednosti i transformacija u jednodimenzioni niz se vrše istovremeno u implementaciji preprocesiranja.

Posle ove transformacije, drugi korak preprocesiranja je skaliranje svih vrednosti na interval $[0, 1]$, gde je maksimalna vrednost celog skupa transformisana u 1, a minimalna u 0. Ova transformacija je pokazala najbolje rezultate kod analize glavnih komponenti, pa je korišćena za treniranje svih modela.

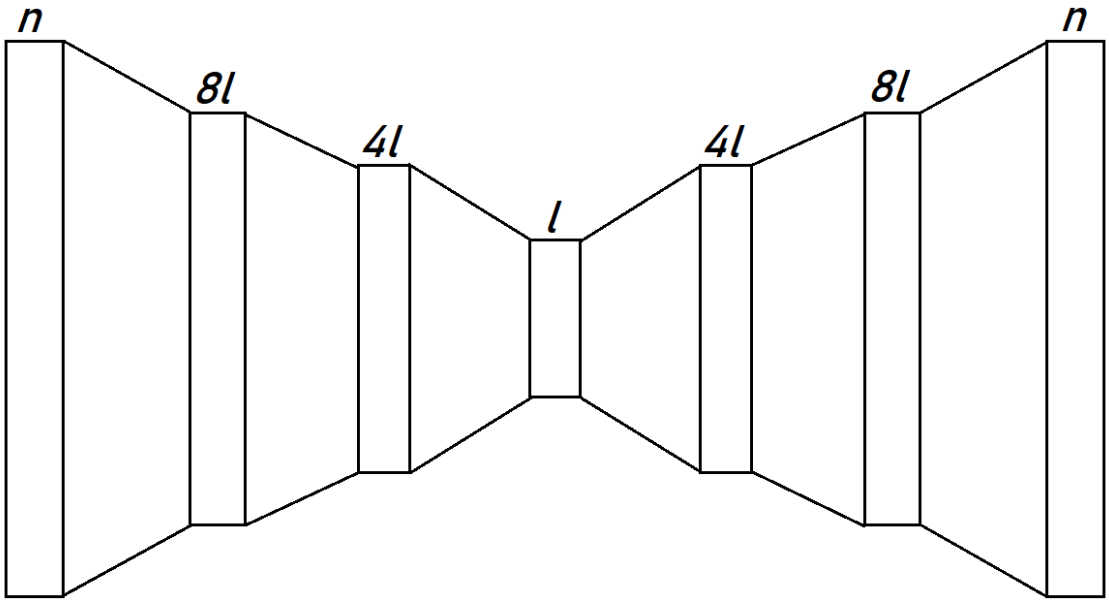
Ovako procesirani podaci se dele na skup za trening, validaciju i test. Podaci se dele u odnosu $7 : 2 : 1$, za trening, validaciju i test, respektivno. Modeli koji koriste neuronske mreže se treniraju pomoću srednje kvadratne greške, a greška koja se meri pri validaciji je srednja apsolutna greška.

Od svih modela koji su testirani izabran je po jedan predstavnik različitih pristupa. Modeli koji se upoređuju su:

- *PCA* - Analiza glavnih komponenti.
- *tanh* - Plitki autoenkoder sa *tanh* funkcijom aktivacije i u enkoderu i u dekoderu. Shema arhitekture ove mreže je prikazana na slici 5.3
- *deep* - Duboki autoenkoder sa 3 sloja u enkoderu i 3 sloja u dekoderu. Svi slojevi koriste *tanh* funkciju aktivacije. Arhitekture enkodera i dekodera su simetrične, pa prvi sloj enkodera i poslednji sloj dekodera imaju $8l$ neurona, sledeći sloj prema unutra ima $4l$ neurona i unutrašnji sloj ima l neurona koji sadrže latentnu reprezentaciju ulaza kao što je prikazano na slici 5.4.
- *relu* - Plitki autoenkoder sa *relu* funkcijom aktivacije i u enkoderu i u dekoderu. Shema arhitekture ove mreže je ista kao shema arhitekture plitkog autoenkodera sa *tanh* funkcijom aktivacije koja je prikazana na slici 5.3.



Slika 5.3: Shema plitkog autoenkodera. Ulazni i izlazni slojevi imaju n neurona, gde je n ukupan broj piksela u ulaznoj mapi. Dimenzija skrivene reprezentacije uzima vrednosti iz tabele 5.2 i te vrednosti se obeležavaju sa l.



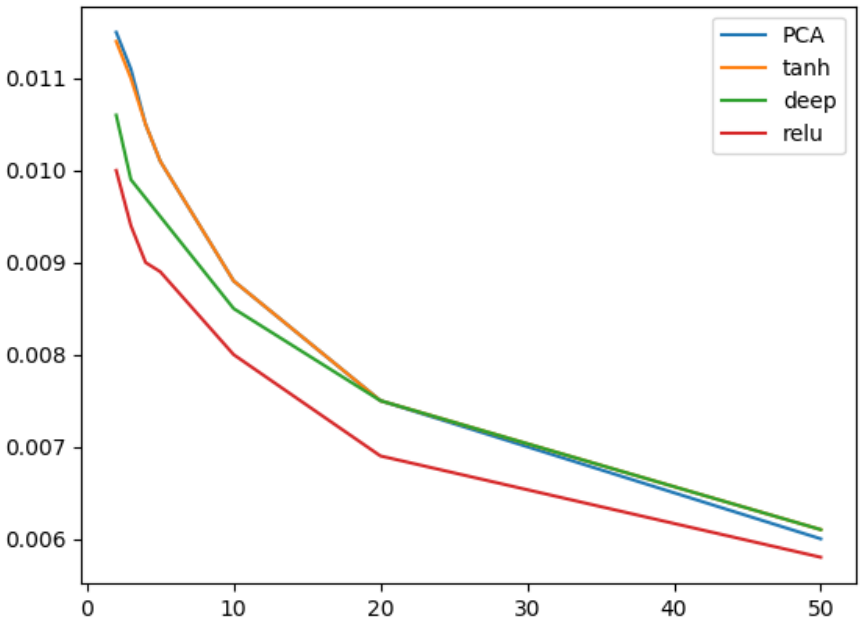
Slika 5.4: Shema dubokog autoenkodera. Ulazni i izlazni sloj imaju n vrednosti, gde je n ukupan broj piksela u ulaznoj mapi. Broj neurona u skrivenim slojevima je umnožak broja l, koji uzima vrednosti iz tabele 5.2. Izlaz skrivenog sloja sa l neurona predstavlja skrivenu reprezentaciju ulaza.

U tabeli 5.2 su prikazane srednje apsolutne greške, kao i standardne devijacije apsolutne greške modela primenjenih na test skup. Apsolutne greške po pikselima se dobijaju tako što se polazni podaci transformišu u njihove reprezentacije, pa se onda reprezentacije transformišu nazad u originalni prostor pomoću inverzne transformacije (dekodera). Srednja apsolutna greška se dobija tako što se uzme sredina apsolutne razlike početne i dobijene

Tabela 5.2: Sa leve strane je tabela apsolutnih grešaka modela primenjenih na skup podataka o padavinama na teritoriji SAD. Sa desne strane je tabela standardnih devijacija apsolutnih grešaka modela primenjenih na skup podataka o padavinama na teritoriji SAD. U svakom redu je podvučen najbolji rezultat za sve modele za određenu dimenzionalnost reprezentacije.

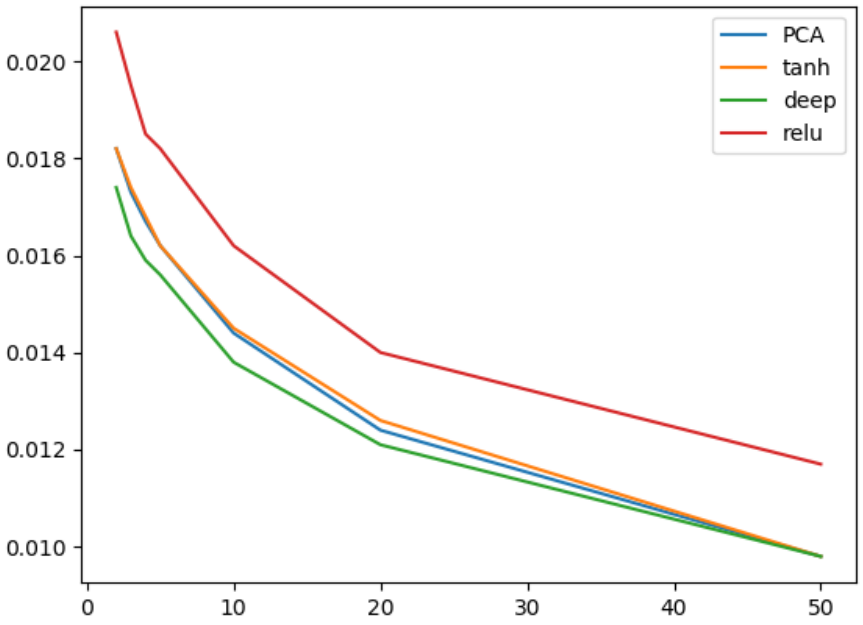
	Srednja apsolutna greška				Standardna devijacija greške			
l	*PCA*	*tanh*	*deep*	*relu*	*PCA*	*tanh*	*deep*	*relu*
2	0,0115	0,0114	0,0106	0,0100	0,0182	0,0182	0,0174	0,0206
3	0,0111	0,0110	0,0099	0,0094	0,0173	0,0174	0,0164	0,0195
4	0,0105	0,0105	0,0097	0,0090	0,0167	0,0168	0,0159	0,0185
5	0,0101	0,0101	0,0095	0,0089	0,0162	0,0162	0,0156	0,0182
10	0,0088	0,0088	0,0085	0,0080	0,0144	0,0145	0,0138	0,0162
20	0,0075	0,0075	0,0075	0,0069	0,0124	0,0126	0,0121	0,0140
50	0,0060	0,0061	0,0061	0,0058	0,0098	0,0098	0,0098	0,0117

vrednosti (kao što je prikazano u poglavlju 2.1.2). Grafik opadanja srednje apsolutne greške u odnosu na rast dimenzionalnosti skrivene reprezentacije je prikazan na slici 5.5. Standardna devijacija apsolutne greške se dobija tako što se na sličan način izračuna standardna devijacija apsolutne greške polaznih vrednosti i rekonstrukcija. Grafik opadanja standardne devijacije apsolutne greške je prikazan na slici 5.6



Slika 5.5: Grafik srednje apsolutne greške modela. Na x osi se nalazi dimenzionalnost skrivene reprezentacije, dok se na y osi nalazi srednja apsolutna greška.

Kao što se vidi iz tabela sa greškama, za ovako procesiran skup podataka analiza glavnih komponenti je i dalje uporediva sa autoenkoderima. Neuronske mreže su modeli čija uspešnost zavisi od mnogo faktora, tako da postoji mogućnost da postoji bolje rešenje od rešenja čiji su rezultati prikazani u ovom radu. Može se videti razlika između plitkog i dubo-



Slika 5.6: Grafik srednje apsolutne greške modela. Na x osi se nalazi dimenzionalnost skrivene reprezentacije, dok se na y osi nalazi srednja apsolutna greška.

kog autoenkodera na nižim dimenzionalnostima latentne reprezentacije, i izjednačavanje po performansama kako raste dimenzionalnost, što ukazuje na to da autoenkoderi više dobijaju od dubine kad je ograničenje izražajnosti veće, to jest dimenzija reprezentacije manja.

Kao što je rečeno u uvodu, korisnost smanjenja dimenzionalnosti podataka se takođe ogleda u tome koliko se dobro čuvaju međusobni odnosi instanci skupa. Jedna od osobina koja treba da se sačuva pri transformaciji je sličnost primeraka po nekom rastojanju, izračunatom na osnovu atributa. Konkretno rastojanje ne mora biti sačuvano, jer se menja prostor atributa, ali je poželjno da rastojanja u početnom prostoru atributa budu u korelaciji sa rastojanjima u prostoru reprezentacija. Korelacije nad transformisanim reprezentacijama su prikazane u tabeli 5.3. Ova mera je dobijena računanjem svih međusobnih rastojanja instanci u polaznom prostoru atributa kao i u prostoru reprezentacije, a zatim računanjem koeficijenta korelacije tih rastojanja [18]. Isečak Python kôda je prikazan u listingu 5.1.

Listing 5.1: Isečak kôda sa funkcijom za izračunavanje korelacije rastojanja.

```
1  def correlation(testset,representations):
2      import numpy as np;
3      original_img_distances=[];
4      representation_distances=[];
5      for i in range(len(testset)):
6          for j in range(len(testset)):
7              if(i!=j):
8                  original_img_distances.append(
9                      l2(testset[i],testset[j]))
10                 representation_distances.append(
11                     l2(representations[i],representations[j]))
12     return np.corrcoef(
```

```
13          original_img_distances, representation_distances)[0, 1]
```

Druga metrika očuvanja rastojanja je pravljena sa akcentom na najbližim instancama. Ideja je uzeti najbliže instance u početnom skupu atributa, i naći njihov rang u prostoru reprezentacije [18]. Pod rangom podrazumevamo indeks u nizu sortiranih elemenata reprezentacija po rastojanju rastuće, u odnosu na jednu instancu. Dakle, uzima se jedna instanca i nalazi se njena najbliža instanca, a zatim se izračunava njen rang u prostoru reprezentacije. Ova vrednost se računa za sve instance skupa, a zatim se računa njen prosek. Vrednosti za primenjene modele su prikazane u tabeli 5.4. Isečak Python kôda je prikazan u listingu 5.2.

Listing 5.2: Isečak kôda sa funkcijom za izračunavanje ranga rastojanja.

```
1  def l2rank(testset,representations):
2      rank=0;
3      for i in arange(len(testset)):
4          odmin=1000000;
5          oimin=-1;
6          for j in range(len(testset)):
7              if(i!=j):
8                  d=np.linalg.norm(testset[i]-testset[j]);
9                  if(d<odmin):
10                     odmin=d;
11                     oimin=j;
12         dist_to_beat=np.linalg.norm(
13                 representations[i]-representations[oimin]);
14         dind=0;
15         for j in range(len(testset)):
16             if(i!=j):
17                 dmin=np.linalg.norm(
18                     representations[i]-representations[j]);
19                 if(dmin<=dist_to_beat):
20                     dind=dind+1;
21         rank+=dind;
22     rank=rank/len(x_test);
23     return rank;
```

Može se videti da se sa povećanjem dimenzionalnosti reprezentacije povećava korelacija rastojanja, što samim tim povećava upotrebljivost novih atributa za algoritme za pretragu u odnosu na L_2 rastojanje. Za modele *deep* i *relu* ovo pravilo ne važi toliko strogo kao za *pca* i *tanh*, što ukazuje na drastičnije razlike u naučenoj reprezentaciji u modelima iste arhitekture sa različitom dimenzionalnosti skrivene reprezentacije. Rang rastojanja takođe pravilno opada sa povećanjem dimenzionalnosti reprezentacije, a modeli *tanh* i *pca* generalno imaju bolje rezultate na ovim merama. Ovo može da bude posledica lošeg odabira arhitekture i treninga što se tiče *deep* modela, ili neadekvatnosti *relu* aktivacije.

Plitki autoenkoder sa *relu* funkcijom aktivacije ima u većini slučajeva manju apsolutnu grešku od ostalih metoda, a kvalitet učenja je dosta loš, što je prikazano na slikama 5.7, 5.8 i 5.11. Takođe se može videti u tabeli 5.2, da *relu* modeli imaju dobru srednju apsolutnu grešku, ali odstupanje od te sredine je veće od ostalih modela. Zbog toga što se mape padavina mogu posmatrati kao retke matrice, a vrednosti koje su iznad nule su male (jer je maksimalna vrednost ekstremna vrednost), velika količina vrednosti u reprezentaciji i na

Tabela 5.3: Tabela korelacije rastojanja parova instanci skupa u polaznom prostoru atributa, i u skrivenoj reprezentaciji. U svakom redu je podvučen najbolji rezultat za sve modele za određenu dimenzionalnost reprezentacije.

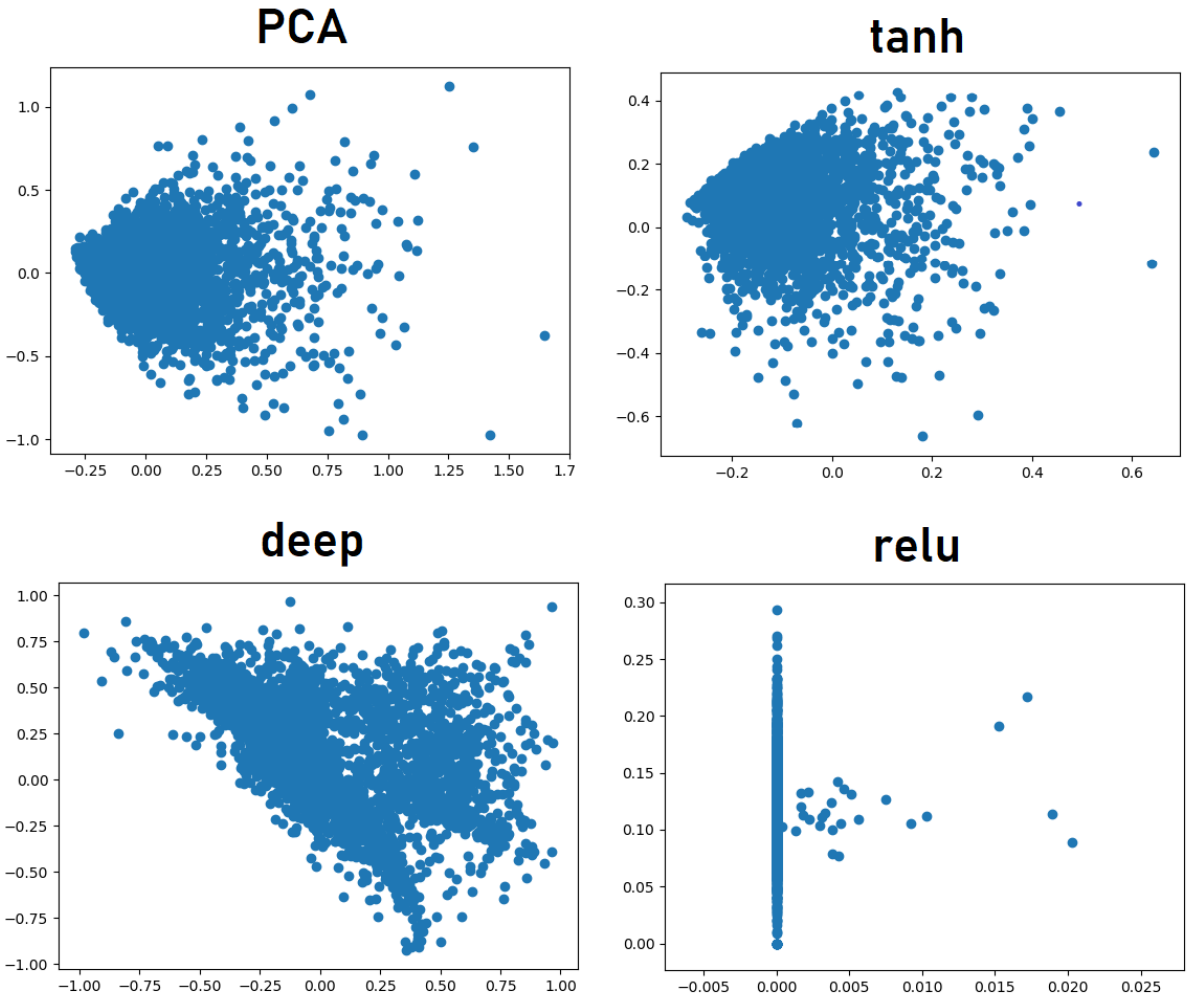
Korelacija rastojanja				
l	PCA	$tanh$	$deep$	$relu$
2	$\underline{0,8237}$	$0,8177$	$0,7413$	$0,7382$
3	$\underline{0,8528}$	$0,8402$	$0,7195$	$0,7372$
4	$\underline{0,8727}$	$0,8603$	$0,7514$	$0,7207$
5	$\underline{0,9001}$	$0,8907$	$0,7862$	$0,8489$
10	$\underline{0,9433}$	$0,9339$	$0,8888$	$0,8811$
20	$\underline{0,9715}$	$0,9692$	$0,9130$	$0,9182$
50	$\underline{0,9875}$	$0,9851$	$0,9326$	$0,9223$

Tabela 5.4: Tabela ranga rastojanja najbližih instanci test skupa. Način izračunavanja ranga rastojanja je pokazan u listingu 5.2. U svakom redu je podvučen najbolji rezultat za sve modele za određenu dimenzionalnost reprezentacije.

Rang rastojanja				
l	PCA	$tanh$	$deep$	$relu$
2	$41,18$	$\underline{40,81}$	$47,13$	$46,77$
3	$26,48$	$\underline{26,07}$	$35,73$	$49,12$
4	$18,92$	$\underline{18,48}$	$23,74$	$31,26$
5	$14,84$	$\underline{14,57}$	$17,28$	$25,21$
10	$7,30$	$\underline{6,93}$	$8,84$	$17,74$
20	$3,69$	$\underline{3,53}$	$4,74$	$11,22$
50	$\underline{2,01}$	$2,14$	$8,01$	$9,08$

izlazu je uvek 0. Relu neuroni su jako korisni kada treba da se dobije retka reprezentacija podataka, ali u slučaju kada podaci već imaju ovakvu raspodelu, ova osobina nije poželjna i preporučuju se druge funkcije aktivacije[6].

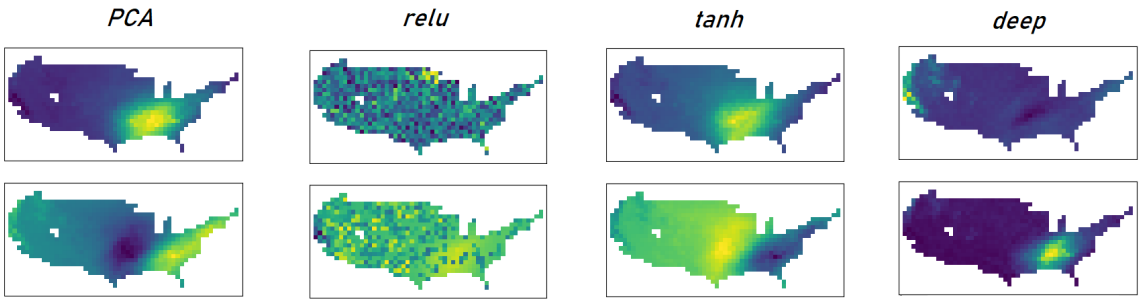
Još jedan način za vizuelizaciju reprezentacije dimenzije 2 je pomoću grafika koji je prikazan na slici 5.7. Prostor u koji preslikava analiza glavnih komponenti je generisan glavnim komponentama prikazanim na slici 5.8, a prostor u koji preslikavaju enkoderi su vrednosti izlaza iz enkodera, čije su težine prikazane su na istoj slici. Prema rezultatima ovog eksperimenta vizualno se može zaključiti da analiza glavnih komponenti i *tanh* model generišu slično preslikavanje kad se primene na skup padavina SAD, i prave jednu veliku grupaciju sa više ekstremnih vrednosti, dok *deep* model pravi 2 široke susedne grupacije. I iz ovog grafika se vidi da *relu* model nije sačuvao previše korisnih informacija, jer je prostor u koji je preslikan ulaz jednodimenzioni sa nekoliko izuzetaka.



Slika 5.7: Vizualizacija skupa padavina SAD u 2 dimenzije, gde se može videti potprostor koji zauzima ovaj skup u prostoru generisanom glavnim komponentama, odnosno individualnim težinama prikazanim na slici 5.8.

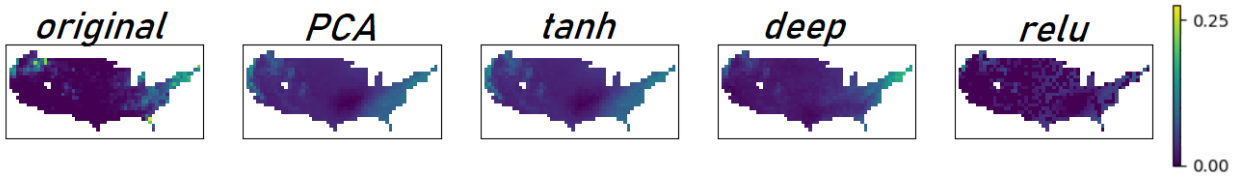
Na slici 5.8 su prikazane vizualizacije vektora koji se preslikavaju u vektore kanonske baze prostora latentne reprezentacije dimenzije 2. Vektori su dobijeni inverznom transformacijom vektora kanonske baze reprezentacije nazad u početni prostor. Za analizu glavnih komponenti se primenjuje inverz matrice transformacije na vektore (ako je dostupna matrica

transformacije, čitaju se kolone inverza te matrice), za autoenkodere se primenjuje dekoder na te iste vektore. Intuitivno, kod analize glavnih komponenti ovi vektori generišu potprostor podataka u originalnom linearnom prostoru.

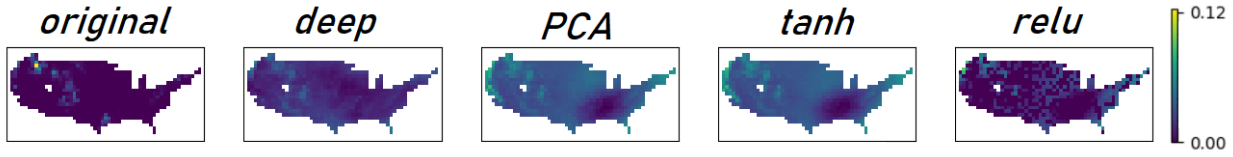


Slika 5.8: Vizualizacija vektora koji se preslikavaju u vektore kanonske baze za modele trenirane na skupu koji sadrži izmerene padavine na području sjedinjenih američkih država. Svi modeli prikazani na slici imaju latentnu reprezentaciju dimenzionalnosti 2.

Na slikama 5.9, 5.10 i 5.11 su prikazane rekonstrukcije modela sa reprezentacijom dimenzionalnosti 2 nekih instanci skupa. Uzete su instance koje su imale najbolje greške rekonstrukcije za modele koji se prikazuju odmah iza originala. Na njima se može videti da su *tanh* i *pca* našli slične transformacije, mogu se videti „ugašeni neuroni" *relu* modela, kao i kompleksniji (i vizualno oštriji) obrasci koje je *deep* model sposoban da nauči u odnosu na ostale.



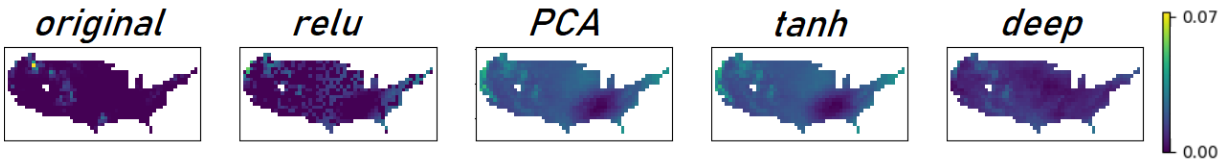
Slika 5.9: Vizualizacija rekonstrukcije svih modela sa latentnom reprezentacijom dimenzionalnosti 2 na instanci skupa koja ima najmanju grešku rekonstrukcije za model *pca*, kao i za model *tanh*.



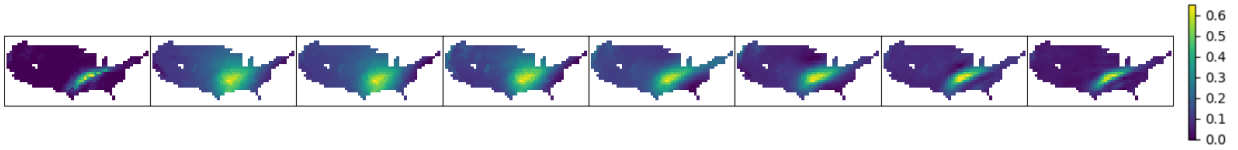
Slika 5.10: Vizualizacija rekonstrukcije svih modela sa latentnom reprezentacijom dimenzionalnosti 2 na instanci skupa koja ima najmanju grešku rekonstrukcije za model *deep*.

Na slikama 5.12, 5.13, 5.14 i 5.15 su prikazane vizuelizacije rekonstrukcija slika za rastuću dimenzionalnost reprezentacije modela. Uzet je datum 18. mart 2008. godine.

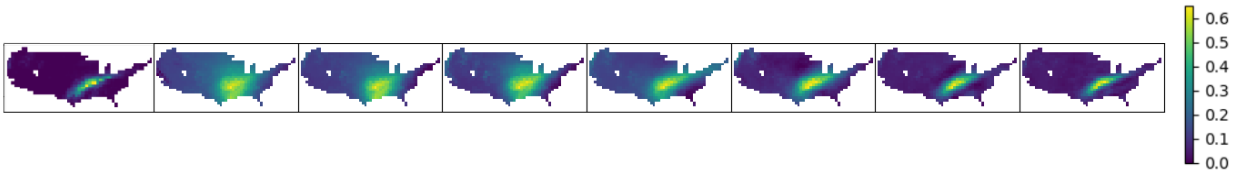
Na slikama 5.16 i 5.17 su prikazane vizuelizacije težina autoenkodera za sve trenirane dimenzionalnosti skrivene reprezentacije, za model *tanh*. Može se videti da model kreće od opštih i pravilnijih oblika prema lokalnijim i oštrijim kako raste dimenzionalnost. Kod težina za autoenkoder sa 50-dimenzionom reprezentacijom se mogu videti sitni oblici koje može da reprezentuje autoenkoder. Na slikama 5.18 i 5.19 su prikazane vizuelizacije težina autoenkodera za sve trenirane dimenzionalnosti skrivene reprezentacije za model *relu*. Mogu



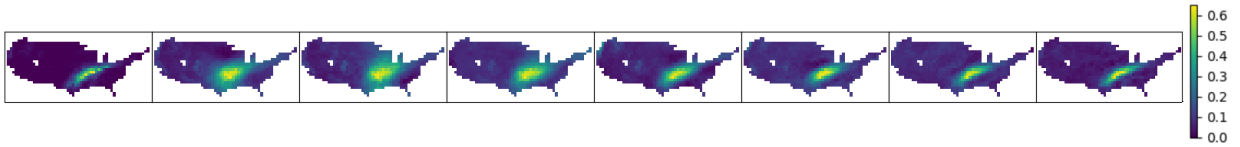
Slika 5.11: Vizualizacija rekonstrukcije svih modela sa latentnom reprezentacijom dimenzionalnosti 2 na instanci skupa koja ima najmanju grešku rekonstrukcije za model *relu*.



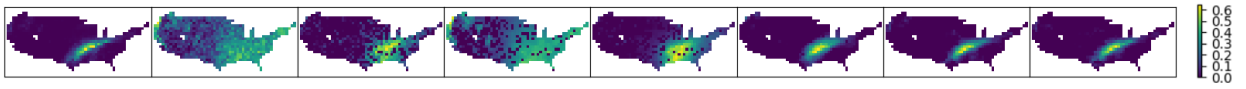
Slika 5.12: Rekonstrukcije jedne mape padavina na području Sjedinjenih Američkih Država prikazane za *pca* model. Prva mapa levo je originalna mapa, sledeće su mape modela sa skrivenom reprezentacijom dimenzija $2, 3, 4, 5, 10, 20$ i 50.



Slika 5.13: Rekonstrukcije jedne mape padavina na području Sjedinjenih Američkih Država prikazane za *tanh* model. Prva mapa levo je originalna mapa, sledeće su mape modela sa skrivenom reprezentacijom dimenzija $2, 3, 4, 5, 10, 20$ i 50.

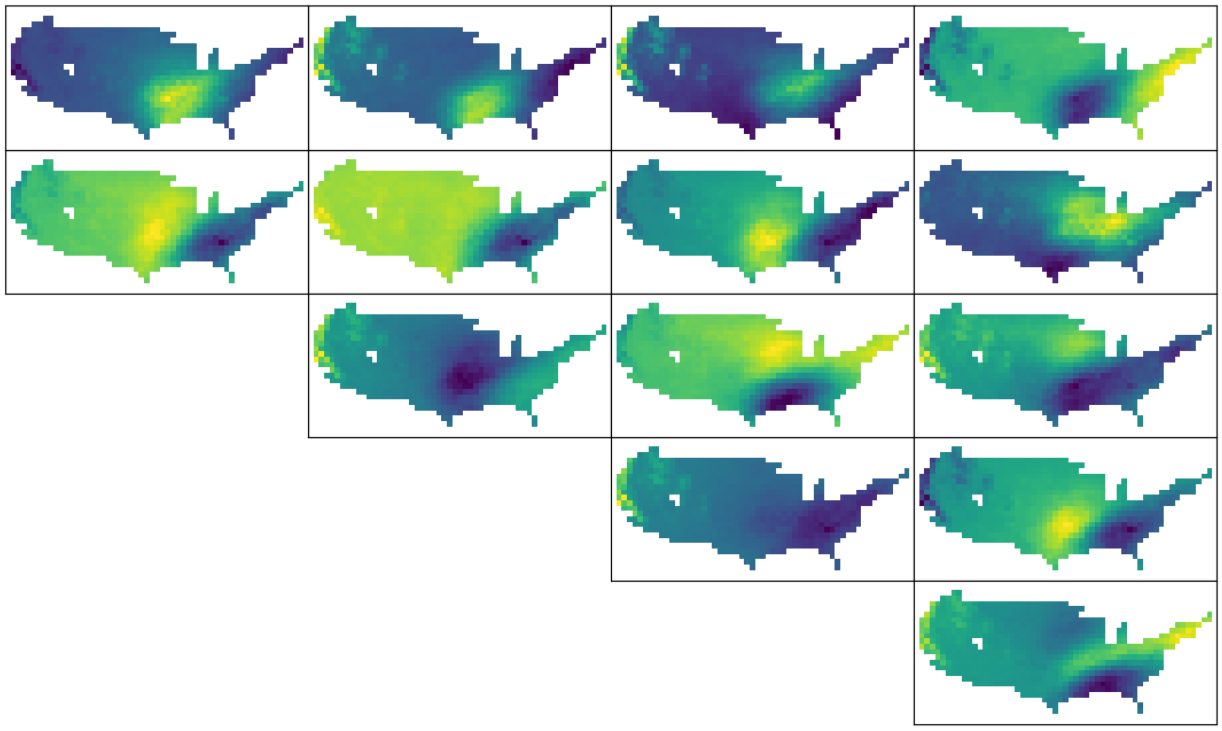


Slika 5.14: Rekonstrukcije jedne mape padavina na području Sjedinjenih Američkih Država prikazane za *deep* model. Prva mapa levo je originalna mapa, sledeće su mape modela sa skrivenom reprezentacijom dimenzija $2, 3, 4, 5, 10, 20$ i 50.

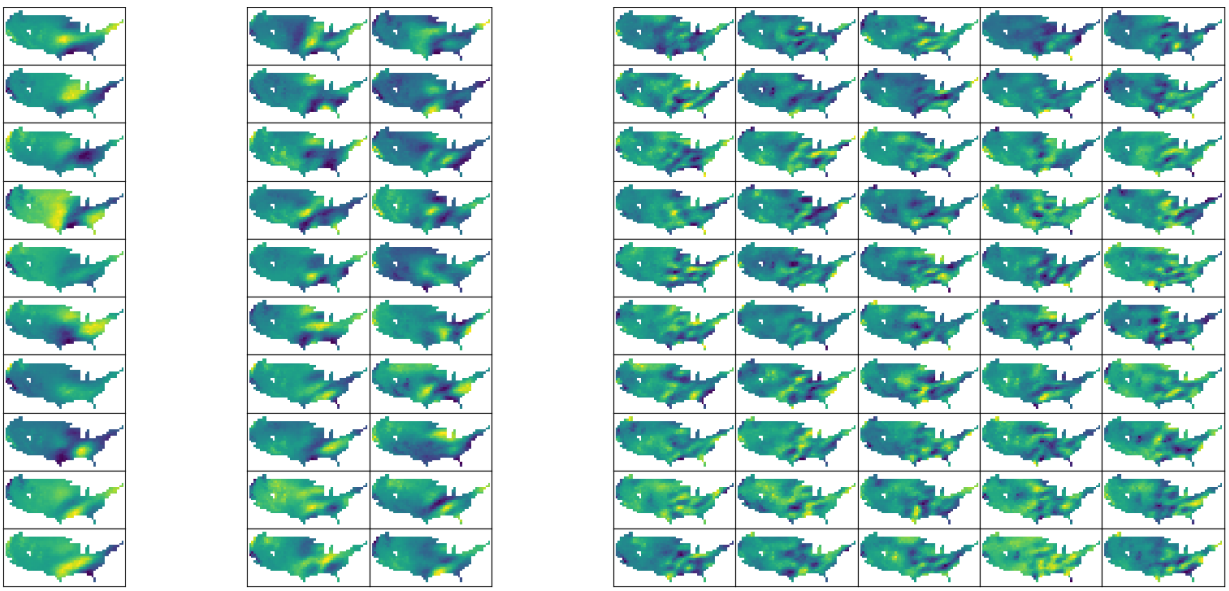


Slika 5.15: Rekonstrukcije jedne mape padavina na području Sjedinjenih Američkih Država prikazane za *relu* model. Prva mapa levo je originalna mapa, sledeće su mape modela sa skrivenom reprezentacijom dimenzija $2, 3, 4, 5, 10, 20$ i 50.

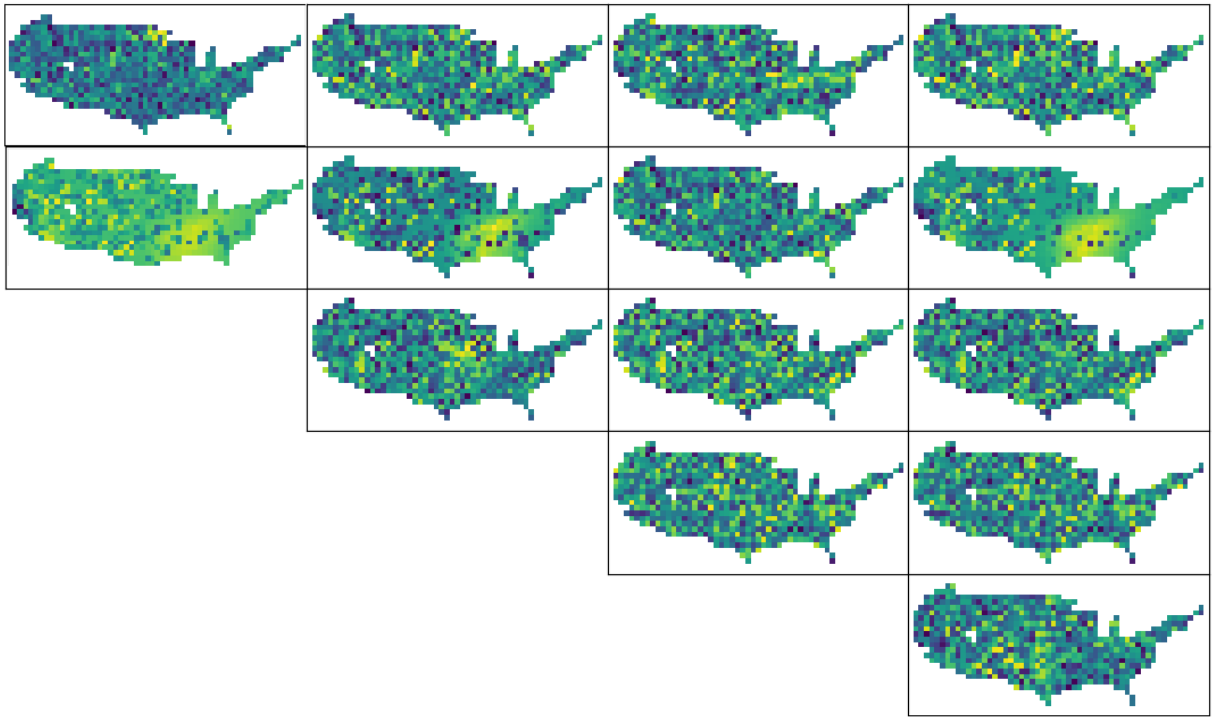
se videti „ugašeni" neuroni, kao i neki obrasci u onim neuronima koji nisu ugašeni. Na slikama 5.20 i 5.21 su prikazane vizuelizacije glavnih komponenti dobijenih primenom analize glavnih komponenti na ulazni skup. Takođe se mogu videti složeniji i oštriji obrasci u kasnijim komponentama.



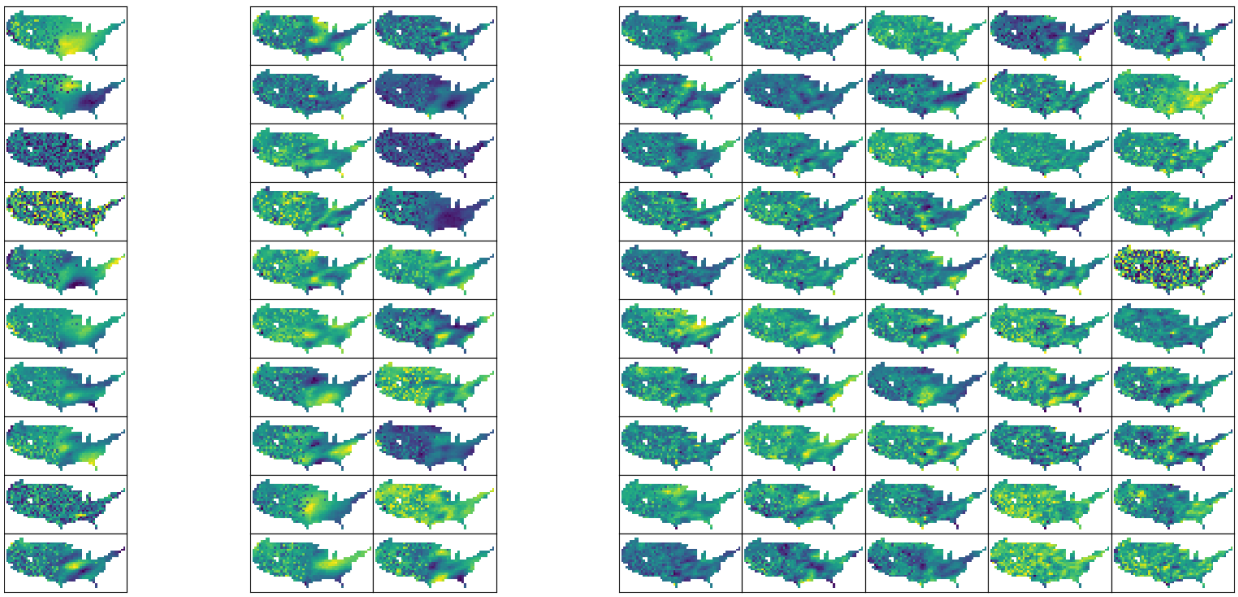
Slika 5.16: Vizuelizacija težina dekodera za model *tanh*, za modeliranje padavina na teritoriji SAD. Težine su poređane po kolonama, prva kolona prikazuje težine modela dimenzionalnosti 2 a sledeća 3, 4 i 5.



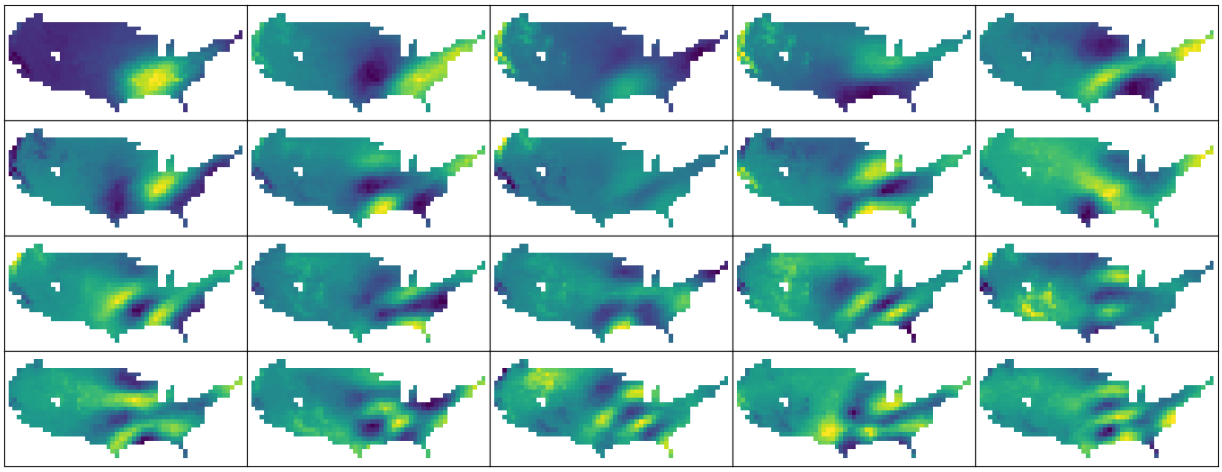
Slika 5.17: Vizuelizacija težina *tanh* modela sa skrivenom reprezentacijom 10 (leva kolona), 20 (srednje kolone) i 50 (desne kolone).



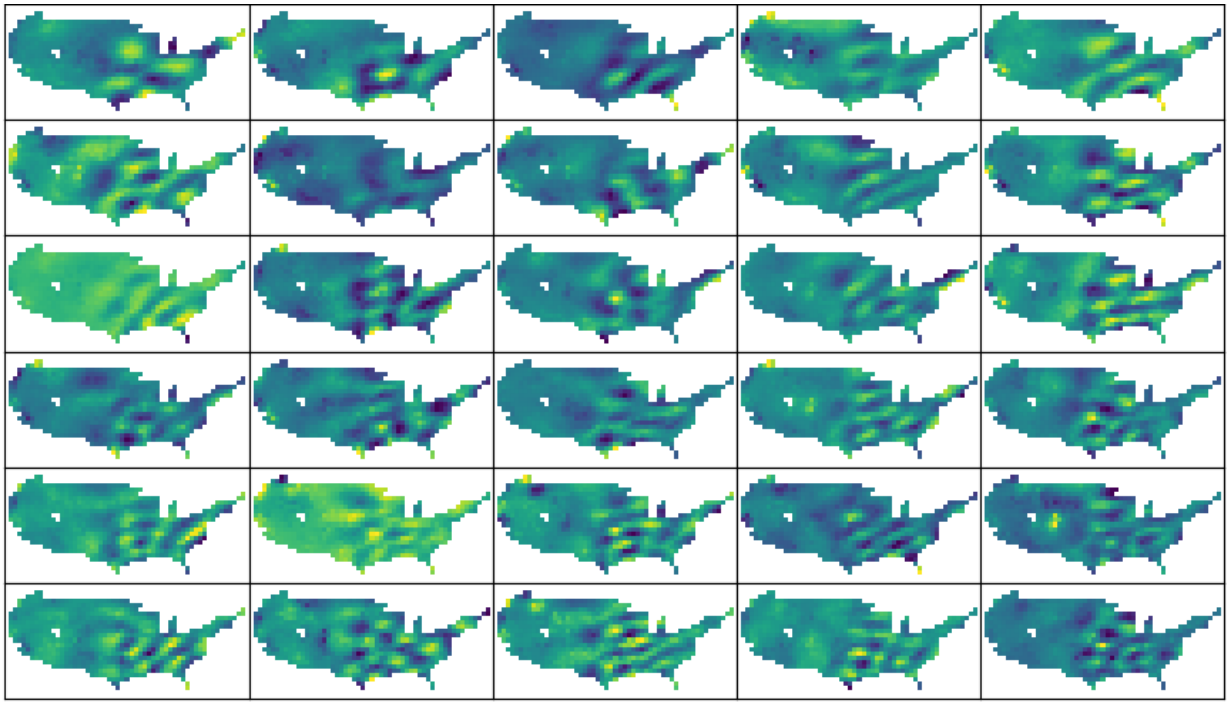
Slika 5.18: Vizuelizacija težina dekodera za model *relu*, za modeliranje padavina na teritoriji SAD. Težine su poređane po kolonama, prva kolona prikazuje težine modela dimenzionalnosti 2 a sledeća 3, 4 i 5.



Slika 5.19: Vizuelizacija težina *relu* modela sa skrivenom reprezentacijom 10 (leva kolona), 20 (srednje kolone) i 50 (desne kolone).



Slika 5.20: Vizuelizacija glavnih komponenti za padavine SAD. Prikazano je prvih 20 glavnih komponenti.



Slika 5.21: Vizuelizacija glavnih komponenti za padavine SAD. Prikazane su komponente 20-50.

Glava 6

Zaključak

Cilj ovog rada je bio da se evaluira primenljivost dimenzionalne redukcije na prostorne podatke i da se uporedi efikasnost različitih metoda, sa akcentom na autoenkoderima. Korišćeni su prostorni podaci o padavinama na teritoriji Sjedinjenih Američkih Država. Primenjena je analiza glavnih komponenti, kao metoda koja se godinama koristi za dimenzionalnu redukciju. Testirani su plitki autoenkoderi sa *relu* i *tanh* funkcijama aktivacija, i pokazano je da za podatke o padavinama *relu* funkcija aktivacije sama po sebi nije dovoljna. Plitki autoenkoder sa *tanh* funkcijom aktivacije je našao slične reprezentacije kao analiza glavnih komponenti, što znači da su ove metode, pod odgovarajućim uslovima, uporedive, ali to može biti posledica svojstava podataka na kojima je evaluacija vršena. Takođe je testiran duboki autoenkoder sa *tanh* funkcijom aktivacije, koji je imao dobre rezultate u pogledu kvaliteta rekonstrukcije, ali rastojanja skrivenih reprezentacija nisu pouzdana, i zavise od mnogo faktora pri treningu.

Neuronske mreže imaju veliku mogućnost odabira parametara arhitekture, ali su u ovom radu testirane samo neke od njih. Uz napredak hardvera, broj slojeva autoenkodera se može povećati još, kao i broj neurona u njima. Takođe se mogu dizajnirati nehomogene arhitekture, kao što su nesimetrični autoenkoderi, gde enkoder i dekoder imaju različitu strukturu, kao i mešanje slojeva sa različitim funkcijama aktivacije.

U ovom radu su korišćene mape padavina sa veoma malom prostornom rezolucijom (12500km^2). Ova rezolucija je u „Daymet" skupu $1km^2$. Da bi se trenirala mreža koja može da koristi ovakve mape, potreban je jači hardver i pametnija upotreba memorije. Mogu se koristiti grafičke kartice za trening, koje ga mogu znatno ubrzati. Takođe se mogu iskoristiti različite metode pretreninga, kao što je prikazano u radu [18].

Svi rezultati ovih eksperimenata su pokazali da postoji nada da se količina prostornih podataka koja se trenutno koristi za ulaz u algoritme mašinskog učenja može znatno i pouzdano smanjiti. Ovo smanjenje može da utiče na povećanje brzine i performansi drugih algoritama mašinskog učenja i pretrage nad ovim podacima.

Literatura

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Medium Corporation Anish Singh Walia. Types of optimization algorithms used in neural networks and ways to optimize gradient descent, 2017.

[3] François Chollet et al. Keras. https://github.com/fchollet/keras, 2015.

[4] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[5] Andrea Frome, Greg S Corrado, Jon Shlens, Samy Bengio, Jeff Dean, Tomas Mikolov, et al. Devise: A deep visual-semantic embedding model. In *Advances in neural information processing systems*, pages 2121–2129, 2013.

[6] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323, 2011.

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[8] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[9] Geoffrey E Hinton and James L McClelland. Learning representations by recirculation. In *Neural information processing systems*, pages 358–366, 1988.

[10] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[11] Steven M Holland. Principal components analysis (pca). *Department of Geology, University of Georgia, Athens, GA*, pages 30602–2501, 2008.

[12] Ian T Jolliffe. Principal component analysis: a beginner's guideintroduction and application. *Weather*, 45(10):375–382, 1990.

[13] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

[14] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[15] Oak Ridge National Laboratory. Daymet: Daily surface weather data on a 1-km grid for north america, version 3.

[16] Quoc V Le. Building high-level features using large scale unsupervised learning. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8595–8598. IEEE, 2013.

[17] Chris J Maddison, Aja Huang, Ilya Sutskever, and David Silver. Move evaluation in go using deep convolutional neural networks. *arXiv preprint arXiv:1412.6564*, 2014.

[18] Miloš Manić and Mladen Nikolić. Feature extraction for rastdrs using autoencoders. *Geostatistics and Machine Learning Conference*, 2017.

[19] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[20] Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, et al. Massively parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1507.04296*, 2015.

[21] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[22] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[23] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[24] Carlos Oscar Sánchez Sorzano, Javier Vargas, and A Pascual Montano. A survey of dimensionality reduction techniques. *arXiv preprint arXiv:1403.2877*, 2014.

[25] Stanford. Cs231n: Convolutional neural networks for visual recognition.

[26] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[27] David Rosenberg (New York University). Loss functions for regression and classification, 2015.

[28] Favio Vázquez. Towards data science, a weird introduction to deep learning.

[29] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

[30] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.