УНИВЕРЗИТЕТ У НОВОМ САДУ
ПРИРОДНО-МАТЕМАТИЧКИ ФАКУЛТЕТ
ДЕПАРТМАН ЗА МАТЕМАТИКУ И
ИНФОРМАТИКУ

Suština podataka i njen značaj za autoenkodere

Seminarski rad

Miloš Vujasinović

Novi Sad, jul 2020

Sadržaj

Uvod

Autoenkoderi su već godinama zlatni standard u smanjenju dimenzionalnosti podataka. Način na koji rade se pokazao kao veoma efikasan u otklanjanju šuma i dopunjavanju podataka koji su oštećeni. Motivisan datim primerima, ovaj rad pokušava da prikaže novi način gledanja na smanjene dimenzionalnosti podataka mašinskim učenjem. Zatim, primenom iznesenih ideja i nekih od principa koji se nalaze u osnovi autoenkodera se uvodi model neuronske mreže koji ima za cilj da iz podataka koji se prosleđuju modelu izdvoji najbitnije odlike za klasifikaciju. Kroz ovaj postupak se takođe razmatraju novi načini treniranja i evaluacije modela, a na kraju se rezultati datog modela porede sa rezultatima tradicionalnih autoenkodera.

1 Suštinski autoenkoderi

1.1 Suština podataka i problem sa tradicionalnim autoenkoderom

Autoenkoderi su neuronske mreže koje se treniraju da oponašaju identičko preslikavanje, to jest da ulaz preslikava na samog sebe. Ovo samo po sebi ne izgleda veoma korisno, međutim autoenkoderi, pored navedenih, sadrže i osobinu da je broj neurona u najmanjem skrivenom sloju mreže manji od broja neurona ulaznog i izlaznog sloja. Ovo za implikaciju ima da se prilikom prolaska podataka kroz mrežu u datom sloju nalazi reprezentacija podataka za čije je predstavljanje potreban manji broj memorijskih jedinica od originalnih podataka i ovo može da varira od nekoliko jedinica, pa do redova veličina manje. Dati skriveni sloj se naziva *usko grlo* (engl. *bottle neck*) i on deli neuronsku mrežu na dva dela: od ulaza do sebe i od sebe do izlaza. Ovi delovi se redom nazivaju *enkoder* i *dekoder*, a oni se koriste za preslikavanje originalnih podataka u reprezentaciju smanjenih dimenzija i nazad. U nastavku rada će se autoenkoderi definisati na formalniji način, ali za trenutne potrebe dato objašnjenje je dovoljno.

Prilikom treniranja autoenkodera se greška računa kao greška između svakog neurona ulaza i njemu odgovarajućeg neurona izlaza. Ovo je veoma intuitivno rešenje ako želimo da rezultati na izlazu budu naizgled što sličniji ulaznim podacima. Međutim dato preslikavanje je u praksi gotovo nemoguće napraviti tako da bude bude savršeno. Kao posledica dolazi do izvesne greške između originalnih podataka i rezultata mreže koji možda ne izgledaju značajno ili su nam gotovo neprimetni, ali, teoretski, mogu značajno da utiču na rezultate klasifikacije ako bi se rezultati autoenkodera pustili kroz klasifikator i uporedili sa rezultatima klasifikacije originalnih podataka. Na primer, ako imamo klasifikator koji traga za odlikom koja je sadržana u veoma malom broju memorijskih jedinica u poređenju sa veličinom jedne instance podataka, a autonkoder zaključi da bi uvrštavanje date odlike samo povećalo grešku, jer bi njenim uvrštavanjem bilo onemogućeno uvrštavanje neke druge odlike koja je iz pogleda greške autoenkodera važnija. Kao posledica, redukcija podataka ovim postupkom je ne potencijalno samo beskorisna, nego i čini podatke neupotrebljivim ako želimo da ih klasifikujemo ili, još gore, koristimo za treniranje klasifikatora.

Diskutovani problem svakako postavlja pitanje kako se može prevazići, ali pre davanja odgovora na njega, potrobno je da definišemo podelu koja će da omogući uočavanje ovog problema jasnijim i lakšim. Iz prikazanog rezonovanja se jasno može uvideti da se autoenkoderi mogu trenirati ili u svrhu očuvanja izgleda podataka ili u svrhu očuvanja onoga što podaci ustvari jesu, odnosno njihove *suštine*. Treba imati u vidu da podaci imaju gotovo bezbroj različitih suština u zavisnosti od toga iz kog se ugla posmatraju, to jest da se u zavisnosti od pitanja koje se postavlja o njima njihova suština menja i odatle možemo da spojimo ideju o suštini sa problemom klasifikacije tako što ćemo reći da pitanje koje postavlja klasifikacioni problem određuje suštinu podataka o kojoj u datom trenutku pričamo. Ove ideje se koriste za podelu definisanu u definiciji 1. Iz ove definicije je jasno da su tradicionalni autoenkoderi ustvari izgledni autoenkoderi, dok autoenkoderi kojima se teži u svrhu prevazilaženja diskutovanog problema su suštinski autoenkoderi, mada se i za njih, u zavisnosti od implementacije, postavlja pitanje koliko dobro mogu da očuvaju odlike bitne za klasifikaciju.

Definicija 1 (Podela autoenkodera po načinu treniranja) *Autoenkoder za čije se treniranje koristi znanje o samim podacima se naziva* izgledni autoenkoder*, dok autoenkoder za čije se treniranje koristi znanje o klasama kojima podaci pripadaju se naziva* suštinski autoenkoder*. Treba imati u vidu da date osobine nisu međusobno disjunktne, pa postoji i* izgledno-suštinski autoenkoder*.*

1.2 Treniranje suštinskih autoenkodera

Da bismo razmatrali kako se klase kojima podaci pripadaju mogu uključiti u treniranje autoenkodera definišimo prvo autoenkodere na formalniji način. Za ovo ćemo iskoristiti definiciju 2 preuzetu iz [1, Poglavlje 2].

Definicija 2 (Opšti autoenkoder *framework* [1]) *$n/p/n$ autoenkoder je definisan kao t-torka $n, p, m, \mathbb{F}, \mathbb{G}, \mathcal{A}, \mathcal{B}, \mathcal{X}, \Delta$ gde važi:*

1. *$\mathbb{F}$ i $\mathbb{G}$ su skupovi.*
2. *n i p su pozitivni celi brojevi. (Autor razmatra slučaj kada je $0 < p < n$ što se podrazumeva prilikom pravljenja autoenkodera)*
3. *$\mathcal{A}$ je klasa funkcija koje preslikavaju ulaz iz $\mathbb{G}^p$ na $\mathbb{F}^n$*
4. *$\mathcal{B}$ je klasa funkcija koje preslikavaju ulaz iz $\mathbb{F}^n$ na $\mathbb{G}^p$*
5. *$\mathcal{X} = \{x_1, \ldots, x_m\}$ je skup od m (trening) vektora u $\mathbb{F}^n$. Ako se radi o spoljašnjem skupu ciljeva u $\mathbb{F}^n$ označavamo ga sa $\mathcal{Y} = \{y_1, \ldots, y_n\}$.*
6. *Δ je funkcija različitosti ili distorcije definisana nad $\mathbb{F}^n$*

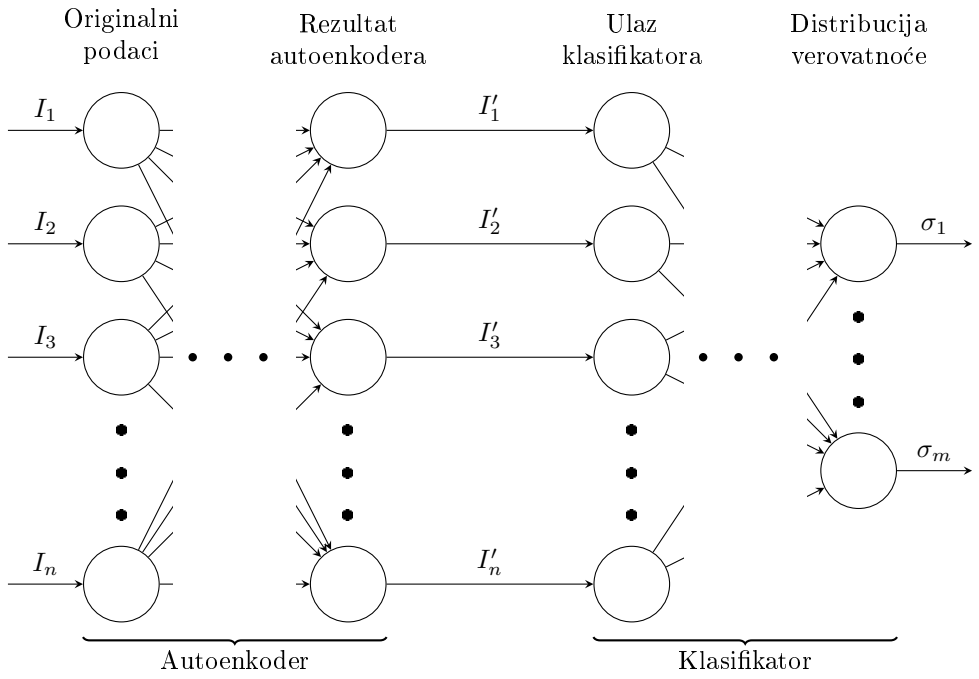
Da bismo bolje razumeli datu definiciju i bliže se upoznali sa terminologijom koja će se koristiti u nastavku prođimo kroz sve elemente date t-torke koja se koristi za predstavljanje autoenkodera. Kao što smo ranije pričali ideja iza autoenkodera je da nauči da slika ulaz na samog sebe, a da pri tome postoji skriveni sloj koji je manjih dimenzija od samog ulaza i na ovaj način se postiže smanjenje dimenzionalnosti podataka. n je ništa manje nego broj dimenzija ulaza kao i izlaza, dok je p broj dimenzija *uskog grla*[1], pošto se cilja da je broj dimenzija uskog grla manji od broja dimenzija ulaza zato se i pretpostavlja da je $0 < p < n$. Svaka memorijska jedinica u ulaznom i izlaznom sloju, odnosno uskog grlu sadrži podatke koji redom pripadaju skupovima $\mathbb{F}$ i $\mathbb{G}$. $\forall B \in \mathcal{B}$ je funkcija koja preslikava ulazne podatke u reprezentaciju smanjenih dimenzija i ona se naziva *enkoder*. $\forall A \in \mathcal{A}$ radi obrnut proces od funkcija iz $\mathcal{B}$ i preslikava reprezentaciju smanjenih dimenzija na izlaz i ova funkcija se naziva *dekoder*. Treba primetiti da se ovde delovi neuronske mreže posmatraju kao funkcije, jer oni to i jesu, i ova notacija će se u nastavku koristiti kada se govori o neuronskim mrežama i njenim delovima. Za treniranje autoenkodera su potrebni podaci i skup trening podataka je u datoj definiciji označen sa $\mathcal{X}$. Međutim, podaci se ne koriste samo za treniranje i možemo imati podatke koji se koriste za evaluaciju ili podatke koji se puštaju kroz autoenkoder u cilju smanjenja dimenzionalnosti. Ovakvi skupovi podataka su u definiciji označeni sa $\mathcal{Y}$. Na kraju, ostala je još da se objasni funkcija različitosti ili distorcije Δ koja igra važnu ulogu u pprocesu treniranja autoenkodera. Cilj treninga autoenkodera je pronalazak funkcija $A \in \mathcal{A}$ i $B \in \mathcal{B}$ za koje je vrednost funkcije distorcije minimalna nad celim skupom trening vektora $\mathcal{X}$. Funkcija distorcije Δ je primer onoga što kod neuronskih mreža znamo kao funkcija greške ili *loss* funkcija.

Funkcija greške je jedna od glavnih karakteristika treninga neuronskih mreža i ona govori u kom smeru će se treniranje kretati. Kao takva ona je očigledna opcija za uključenje klasa kojima podaci pripadaju u trening i na taj način dobijanje suštinskog autoenkodera. Međutim, ovo nije tako lako izvesti kao što zvuči. Dok se kod tradicionalnih, izglednih, autoenkodera srednja kvadratna greška nudi kao idealna funkcija greške, kod suštinskih autoenkodera stvari su malo komplikovanije i zahtevaju se određeni trikovi u cilju dobijanja željenog efekta. Glavni problem je što kod treniranja izglednih autoenkodera je vrednost koju mreža treba da proizvede poznata,

[1] sloj u kome se prilikom prolaza podataka kroz autoenkoder nalazi reprezentacija podataka smanjenih dimenzija

dok kod suštinskih autoenkodera je poznata samo vrednost koja treba da se dobije kada se traženi rezultati puste kroz određenu funkciju. Na osnovu ovoga definicija 3 uvodi dva načina treniranja na koja se model mašinskog učenja može učiti.

Definicija 3 (Podela načina treniranja modela mašinskog učenja) *Ako je prilikom treniranja modela mašinskog učenja poznata vrednost koja se treba dobiti kao proizvod rada modela onda se dati proces treniranja naziva* direktno treniranje. *Nasuprot tome, treniranje u kome je poznata samo vrednost koja se treba dobiti nakon što se rezultati rada modela puste kroz određenu funkciju se naziva* indirektno treniranje.



Slika 1: Model klasifikatora dobijen spajanjem autoenkodera i manjeg klasifikatora

Iako su suštinski autoenkoderi dati kao primer indirektnog treniranja, uz pomoć malog trika, se može napraviti da njihovo treniranje bude deo direktnog procesa treniranja. Pošto znamo da rezultat suštinskog autoenkodera kada se pusti kroz klasifikator treba da da nama poznatu klasu, možemo ovo svesti na istovremeno treniranje autoenkodera i klasifikatora. Ideja je da imamo strukturu autoenkodera i klasifikatora i da od njih napravimo novu neuronsku mrežu tako što ćemo staviti da ulaz prvo prolazi kroz autoenkoder, potom da izlaz autoenkodera bude ulaz u dati klasifikator, a sve zajedno treniramo kao da je jedan veliki klasifikator. Pošto znamo da je izlaz autoenkodera istih dimenzija kao i ulaz, koji je ujedno istih dimenzija kao i ulaz klasifikatora, nema problema prilikom prosleđivanja rezultata autoenkodera klasifikatoru, jer su dimenzije odgovarajućih slojeva jednake. Grafik na kojem je prikazan jedan ovakav model se može videti na slici 1. Ova ideja iako zvuči moguće ipak sa sobom nosi dva problema. Prvi je što imamo veliki model što za posledicu ima duže vreme treniranja i može da naškodi kvalitetu istreniranog modela. Drugi problem je ono što čini dati pristup beskorisnim, ali da bismo njega shvatili moramo se vratiti na koncept suštine podataka i njene korelacije sa originalnim podacima. Suština podataka je rezultat razmišljanja kako iz podataka može da se ukloni što je više moguće osobina

tako da rezultat klasifikacije podataka koji sadrže preostale osobine bude isti kao rezultat klasifikacije originalnih podataka. Ovo bi značilo da ako imamo savršen suštinski autoenkoder i znamo da se ulazni podaci klasifikakuju u određenu klasu da bi onda i rezultat autoenkodera za date podatke morao da se klasifikuje u istu klasu, kao i obrnuto. Rezultat prethodno opisanog procesa treniranja bi bili autoenkoder i klasifikator i iako bi rezultat klasifikacije kroz dati klasifikator podataka propuštenih kroz dati autoenkoder težio ka klasi ka kojoj je treniran, u datom procesu treniranja ne postoji mehanizam koji obezbeđuje da će originalni podaci biti dobro klasifikovani kada se puste kroz dati klasifikator. Ovo kao posledicu ima da je dobijeni klasifikator neispravan, što dalje uzrokuje da se dobijeni autoenkoder ne može koristiti, jer je treniran na neispravnom klasifikatoru.

Prethodno diskutovani proces treniranja, kao što smo videli, sadrži mane i probleme koji ga čine neupotrebljivim, međutim on postavlja dobru osnovu u načinu i smeru u kojem treba da se razmišlja o ovom problemu. Ono što smo uvideli iz prethodne diskusije je da je za pravljenje ispravnog suštinskog autoenkodera potreban ispravan klasifikator, a mašinsko učenje i neuronske mreže se bave temom klasifikatora već dugo vremena i znamo kako se on može napraviti. Kada imamo istreniran klasifiaktor možemo da iskoristimo ideju da spojimo autoenkoder i klasifikator i treniramo novi model kao jedan veliki klasifikator, međutim moramo da zamrznemo sve parametre već istreniranog klasifiaktora, jer samo tako možemo da budemo sigurni da će istreniran klasifikator biti ispravan tokom celog treninga i da će se izbeći problem iz prethodnog primera. Realizacija ovoga je po prilici veoma prosta, i dalje se pušta nazadna propagacija[2] kroz celu mrežu, ali uz izmenu da menja samo parametre autoenkodera, dok parametri istreniranog klasifikatora preskače kada na njih dođe red da se izmene. Dati autoenkoder se na ovaj način trenira da preslikava ulaz na rezultat koji se klasifikacijom kroz dati istrenirani autoenkoder svrstava u traženu klasu.

Iako je prethodna ideja veoma blizu rešenju koje problem zahteva, ona ipak ima jednu manu. Ako se prisetimo priče o suštini podataka, rekli smo da se originalni podaci i rezultat autoenkodera trebaju klasifikovati u istu klasu, a nema garancije da će istrenirani klasifikator ispravno klasifikovati originalne podatke, dok mi treniramo autoenkoder kao da je ovo slučaj. Iz ovog razloga treba da napravimo izmenu onoga čemu veliki klasifikator, koji se sastoji od autoenkodera i istreniranog klasifikatora, cilja. Umesto da vodi trening ka klasi kojom su originalni podaci labelovani treba da cilja na distribuciju verovatnoće[3] koja se dobije kada se originalni podaci puste kroz istrenirani klasifikator.

Diskutovano rešenje je, kao što vidimo, ispravno, ali je računski veoma zahtevno i bilo bi idealno ako bi mogao da se nađe način da se dati proces ubrza. Ono što može da se uradi po ovom pitanju je da popustimo uslov i ne prolazimo kroz ceo istrenirani klasifikator tokom treninga, nego da idemo samo do određenog sloja datog klasifikatora i treniramo autoenkoder tako da vrednosti u datom sloju kada se rezultati puste kroz klasifikator budu iste kao i u slučaju originalnih podataka. Ono što je interesantno je da je način treniranja tradicionalnih autoenkodera i prethodno diskutovano rešenje ustvari specijalan slučaj ovog pristupa: ako se uzme samo ulazni sloj mreže, dobija se tradicionalno rešenje, dok ako se uzme ceo autoenkoder dobija se originalno rešenje izneseno u ovom radu. Realizacija ove ideje je analogna originalnom rešenju, međutim prilikom izbora ovog pristupa treba biti svestan da on sa sobom nosi manu da u opštem slučaju što se uzme manji broj slojeva da će rezultat klasifikacije podataka dobijenih autoenkoderom biti dalje od rezultata klasifikacije originalnih podataka.

Svi autoenkoderi o kojima smo do sada pričali ili su bili čisto izgledni ili čisto suštinski autoenkoderi, a ako se prisetimo definicije 1, u njoj smo uveli i treću vrstu koja kombinuje ove

[2] algoritam koji se koristi za treniranje neuronskih mreža

[3] klasifikator preslikava ulazne podatke na distribuciju verovatnoće koja govori sa kojom verovatnoćom podaci pripadaju svakoj od mogućih klasa

dve diskutovane ideje. Kombinovanje ove dve ideje se može izvesti na veoma jednostavan način: ako imamo funkciju distorcije $\Delta_1 : \mathbb{A} \to \mathbb{B}_1$ specifičnu za treniranje izglednog autoenkodera, funkciju distorcije $\Delta_2 : \mathbb{A} \to \mathbb{B}_2$ specifičnu za treniranje suštinskog autoenkodera, kao i binarni operator $\oplus : \mathbb{B}_1 \times \mathbb{B}_2 \to \mathbb{B}$ možemo definisati funkciju distorcije $\Delta' : \mathbb{A} \to \mathbb{B} : x \to \Delta_1(x) \oplus \Delta_2(x)$ koja je specifična za treniranje izgledno-suštinskog autoenkodera. Primere funkcija distorcije Δ_1 i Δ_2 već znamo ili smo ih diskutovali u ovom radu, dok za operator $\oplus$ možemo izabrati neki od operatora koji se najčešće koriste, na primer operator sabiranja $+$ ili operator množenja $\cdot$. Ono o čemu treba voditi računa prilikom primene biblioteka za rad sa neuronskim mrežama je da neke od njih rezultat funkcije greške dobijaju kao višedimenzionalne tenzore koji ne moraju strogo biti istih dimenzija i zato se treba primeniti funkcija koja će date vrednosti da preslika na rezultate koji pripadaju skupovima na koje se može primeniti odabrani operator.

1.3 Identičnost rezultata klasifikacije pri smanjenju dimenzionalnosti podataka

Diskutovali smo suštinske autoenkodere i načine na koje se oni mogu trenirati, sledeće logično pitanje je kako možemo da ocenimo kvalitet suštinskog autoenkodera i uporedimo dva ovakva autoenkodera da odredimo koji je bolji. Ideja suštinskih autoenkodera se fundamentalno razlikuje od gotovo svega do sada viđenog u oblasti neurosnkih mreža i mašinskog učenja, s toga poznate standardne metrike nisu dobar pokazatelj kvaliteta modela ovog tipa autoenkodera. Jedna ideja je da ako imamo skup podataka za evaluaciju $\mathcal{Y} = \{y_1, \ldots, y_k\}$ da pustimo podatke prvo kroz autoenkoder, a zatim proverimo koliki je procenat njih svrstan u klasu u koju su labelovane. Ovo izgleda kao dobra metrika, ali ona nije u potpunosti u skladu sa motivacijom iza suštine podataka. Ako se setimo cilj iza suštine podataka je da se podaci u što većoj meri klasifikuju u istu klasu u koju i rezultati kada se ti isti podaci puste kroz suštinski autoenkoder. Možemo pogledati definiciju 4 koja definiše pojam identičnost rezultata klasifikacije pri smanjenju dimenzionalnosti podataka, u nastavku samo identičnost rezultata, koja je upravo mera ovoga do sada diskutovanog. Ova mera predstavlja procenat trening vektora iz skupa $\mathcal{Y}$ koji su klasifikovani klasifikatorom g u istu klasu u koju su istim klasifikatorom klasifikovani i rezultati kada se isti ti trening vektori puste kroz autoenkoder g.

Definicija 4 (Identičnost rezultata klasifikacije pri smanjenju dimenzionalnosti) *Za autoenkoder $f : \mathbb{A}^n \to \mathbb{A}^n$ i klasifikator $g : \mathbb{A}^n \to \mathbb{B}$ (relacija $=$ mora biti definisana nad $\mathbb{B}$) koji vraća klasu u kojoj se ulaz sa najvećom veroatnoćom nalazi definišemo meru* identičnost rezultata klasifikacije pri smanjenju dimenzionalnosti podataka Υ nad skupom $\mathcal{Y} = \{y_1, \ldots, y_k\}, \mathcal{Y} \subset \mathbb{A}^n$ *kao:*

$$\Upsilon(f, g; \mathcal{Y}) = \frac{|\{y \mid y \in \mathcal{Y} \ \wedge \ g(y) = g(f(y))\}|}{|\mathcal{Y}|}.$$

Problem kod identičnosti rezultata je pitanje izbora klasifikatora koji ćemo da koristimo u datoj metrici i kao logičan odgovor na ovo se daje klasifikator koji je korišćen tokom treniranja autoenkodera. Međutim, ovde se lako može primetiti da se tako može dobiti da je identičnost rezultata na datom klasifikatoru visoka, a na ostalim klasifikatorima koji se koriste za isti klasifikacioni problem veoma mala, to jest određeni vid *overfitting*-a. Ovo budi mnogo dublja pitanja. Ako se prisetimo, mi do sada još nismo formalno definisali suštinu podataka i najbliže što smo rekli o njoj je da ako se iz podataka uklone sve osobine koje nisu bitne za klasifikaciju da ono što ostaje je ustvari suština podataka. Ono što je ovde problem je ako postoje dva klasifikatora g_1 i g_2 koji redom tragaju za skupovima osobina $\mathcal{G}_1$ i $\mathcal{G}_2$ nad određenim podacima u svrhu pronalaženja odgovora na isto klasifikaciono pitanje i na to sve još važi da je $\mathcal{G}_1 \cap \mathcal{G}_2 \neq \emptyset$. Preneseno rečima,

ovo znači da dati klasifikatori tragaju za različitim osobinama u svrhu klasifikacije podataka, što bi značilo da suština podataka nije jedinstvena između klasifikatora koji rešavaju isti klasifikacioni problem. Ovaj problem izlazi iz okvira ovoga rada i s toga ostaje otvoreno pitanje, a ako je odgovor potvrdan, što je verovatno i slučaj, treba razmisliti o načinu da se suština iz ugla više klasifikatora može spojiti u jednu celinu. Do trenutka dok se ne odgovori na ova pitanja svako korišćenje suštinskih autoenkodera sa klasifikatorima sa kojima dati autoenkoder nije treniran je nepreporučljivo.

1.4 Izdvajanje suštine podataka

Do sada je bilo mnogo priče o suštini podataka, ali je njeno razumevanje, u smislu da možemo pogledati podatke i uvideti šta je njihova suština, veoma teško. Međutim suštinski autoenkoderi govore o očuvanju suštine i postavlja se pitanje da li možemo nekako ovu ideju da iskoristimo za izdvajanje suštine iz samih podataka. Ako se zamislimo kako klasifikatori rade, oni u suštini tragaju za određenim osobinama, što bi intuitivno značilo da odbacuju one za koje vide da nemaju značaja za klasifikaciju i tako kako se ide dublje u klasifikator se u suštini sve veći broj osobina filtrira i pri samom kraju ostaje reprezentacija koja sadrži osobine ulaznih podataka koje su najbitnije za klasifikaciju. Ono što je cilj odavde je da probamo da od ove reprezentacije dobijemo podatke na ulazu i ako je intuicija tačna onda su se osobine koje nisu bitne za klasifikaciju izgubile, što znači da bi rekonstruisana verzija sadržala samo osobine bitne za klasifikaciju, što je ustvari ono što smo definisali kao suštinu podataka. Važno je primetiti da se ovako gledano klasifikator ponaša slično enkoderu ako uzmemu prvih nekoliko slojeva klasifikatora, ali obavezno izostavimo poslednji sloj koji govori o distribuciji verovatnoće da podaci pripadaju svakoj od datih klasa. Šta više, popuštanjem opšteg autoenkoder *framework*-a (definicija 2) možemo ovo uklopiti u njega. Cilj treninga autoenkodera se svodi na pronalazak funkcija $B \in \mathcal{B}$ i $A \in \mathcal{A}$ za koje je funkcija distorcije Δ po skupu trening vektora minimalna. Uzimanjem prvih nekoliko slojeva klasifikatora da igraju ulogu enkodera mi ustvari već dobijamo funkciju B iz definicije i proces trenigna bi se ustvari sveo samo na pronalazak funkcije A. Ovo se može realizovati na sličan način na koji smo realizovali treniranje suštinskog autoenkodera uz pomoć unapred istreniranog klasifikatora, a to je samo zamrznemo parametre datog dela klasifikatora, dok ostatak treniramo kao i obično propagacijom u nazad. Za treniranje ovakve mreže se mogu koristiti načini treniranja izglednih, suštinskih, kao i izledno-suštinskih autoenkodera, šta više u nastavku rada će se uporediti rezultati svakog od ovih načina treniranja. Ovakva mreža će se u nastavku zvati *KeyFeaturesNet* ili skraćeno *KFN*.

2 Rezultati testiranja

U ovom poglavlju ćemo prikazati kakav uticaj različiti načini treniranja imaju na metrike *KeyFeaturesNet*-a, kao i videti kako se njihovi rezultati nose sa rezultatima tradicionalnih (izglednih) autoenkodera. Svakom modelu treniranom posebnim načinom učenja kao i tradicionalnom autoenkoderu su dodeljene posebne identifikacione oznake radi lakšeg prepoznavanja. Tradicionalni autoenkoder ćemo označavati kao **AE**. Za treniranje *KFN*-a koristićemo sve do sada analizirane načine učenja. Prvi od njih je način na koji se treniraju izgledni autoenkoderi, a to je primena srednje kvadratne greške između rezultata *KFN*-a i originalnih podataka kao funkcije distorcije. Modelu koji koristi ovu funkciju za treniranje dodelićemo oznaku **KFN_MSE**. Zatim, prisetimo se načina za treniranje suštinskih autoenkodera koje smo diskutovali u podpoglavlju 1.2. Prvi od njih je da treniramo *KFN* tako da distribucija verovatnoće klasifikacije rezultata *KFN*-a bude jednaka distristribuciji verovatnoće klasifikacije originalnih podataka. Ako se prisetimo realizacija ovoga je išla spajanjem tada autoenkodera i istreniranog klasifikatora u novi model koji se trenira kao jedan veliki klasifikator, ali kome su parametri delova modela konstantni i njihova izmena se preskače tokom propagacije u nazad. Za treniranje velikog klasifikatora se kao funkcija distorcije uzima kros entropija (engl. *cross entropy*) između distribucija verovatnoće klasifikacije istreniranim klasifikatorom rezultata autoenkodera i originalnih podataka. Po nazivu funkcije koja se koristi ovaj model ćemo označiti sa **KFN_CE**. Sledeće što smo rekli je da ne moramo puštati vrednosti kroz ceo klasifikator, već da možemo ići do određenog sloja i da podešavamo vrednosti parametara tako da vrednosti originalnih podataka i rezultata autoenkodera kada se puste kroz dati klasifikator budu jednake u datom sloju. Prisetimo se da se veoma slična ideja koristi za enkoder deo *KFN*-a i s toga možemo odabrati isti sloj da do njega idemo i za treniranje, a kao funkciju distorcije uzimamo srednju kvadratnu grešku između rezultata originalnih podataka i izlaza *KFN*-a kada se u istreniranom klasifikatoru puste do datog sloja. Ovde treba napomenuti da se klasifikator od kojeg je uzeto prvih nekoliko slojeva kao enkoder *KFN*-a koristi i tokom opisanog načina treniranja. Model treniran ovim načinom označavamo sa **KFN_MID**. Na kraju, možemo kombinovati date načine treniranja izglednih i suštinskih autoenkodera kako bismo dobili izgledno-suštinski *KFN* autoenkoder. Operator koji je u testiranju korišćen za kombinovanje funkcija distorcije je operator sabiranja + nad realnim brojevima. Biblioteka za rad sa neuronskim mrežama korišćena u radu vrednosti funkcija greške može da vrati kao višedimenzionalne tenzore i iz tog razloga su korišćene funkcije redukcije kako bi se dati tenzori sveli na realne brojeve. U ovu svrhu korišćena je funkcija sumiranja koja vraća zbir svih elemenata datog tenzora, a pored nje je korišćena i funkcija srednje vrednosti koja dati tenzor preslikava na srednju vrednost njegovih elemenata. U zavisnosti koja je funkcija redukcije korišćena imamo redom sufikse SUM i AVG. Kombinacijom KFN_MSE i KFN_CE dobijamo model sa oznakom **KFN_CE_MSE** i u zavisnosti od korišćene funkcije redukcije imamo **KFN_CE_MSE_SUM** i **KFN_CE_MSE_AVG**. Za KFN_MSE i KFN_MID analogno imamo model pod oznakom **KFN_MID_MSE** i u zavisnosti od korišćene funkcije redukcije imamo **KFN_MID_MSE_SUM** i **KFN_MID_MSE_AVG**.

Mere koje su korišćene za evaluaciju klasifikatora su srednja vrednost funkcije greške, kao i preciznost klasfikacije na evaluacionom skupu podataka. Za razliku od ovoga za autoenkoder i modele *KFN*-a prikazana je identičnost rezultata klasifikacije pri smanjenju dimenzionalnosti podataka i, manje bitne, srednja vrednost funkcije greške (što se još naziva *loss*) i tačnost klasifikacije kada se rezultati odgovorajaćih modela puste kroz istrenirani klasifikator. Za evaluaciju je korišćen poseban skup podataka koji je nezavisan od skupa korišćenog za treniranje bilo kog od navedenih modela.

Dosta je bitno videti kako se modeli ponašaju nad skupovima podataka različite kompleksnosti, odnosno koliko su skalabilni u zavisnosti od složenosti klasifikacionog problema koji im

je dat. Iz ovog razloga prikazani su rezultati testiranja nad dva skupa podataka koji se razlikuju po kompleksnosti, a to su **MNIST** i **CIFAR-10**. Oba skupa su skupovi slika i ovo nije uopšte slučajno, jer ćemo moći mnogo lakše da uporedimo originalne podatke sa rezultatima datih modela nego što bi to bio slučaj da se radi o skupovima podataka koji nisu slike i šta više glavna primena autoenkodera je u smanjenju dimenzionalnosti slika. Zbog osobine problema svi modeli će koristiti konvolucijske strukture mreža koje su mnogo efikasnije u radu sa slikama nego što je to slučaj sa potpuno povezanim slojevima. Za svaki od problema su modeli *KFN*-a identični i razlikuje se samo način njihovog treniranja, dok struktura datog tradicionalnog autoenkodera može malo da varira od njih u svrhu dobijanja boljih rezultata (varijacije su na nivou broja filtera u slojevima i korišćenju različitih slojeva pri povećanju dimenzionalnosti koji rade isti posao uz manje izmene).

2.1 MNIST

MNIST je skup podataka koji se sastoji iz dva dela: trening skupa sa 60000 primera i test skupa sa 10000 primera. On se sastoji od ručno napisanih cifara od 0 do 9 u vidu crno-belih slika dimenzija 28×28 piksela. MNIST je veoma jednostavan skup podataka namenjen koji se preporučuje ljudima koji žele da nauče tehnike i metode prepoznavaća osobina na podacima iz stvarnog sveta [2]. Kao takvog ovaj skup podataka možemo koristiti kao dobar pokazatelj da li su do sada diskutovani metodi uopšte primenljivi nad stvarnim podacima i kako se nose sa njima.

Model	*Loss* pri klasifikaciji rezultata modela	Tačnost pri klasifikaciji rezultata modela	Identičnost rezultata klasifikacije pri smanjenju dimenzionalnosti
AE	0	0	0
KFN_MSE	0	0	0
KFN_CE	0	0	0
KFN_MID	0	0	0
KFN_CE_MSE_SUM	0	0	0
KFN_CE_MSE_AVG	0	0	0
KFN_MID_MSE_SUM	0	0	0
KFN_MID_MSE_AVG	0	0	0

Tabela 1: Rezultati evaluacije različitih modela na skupu podataka MNIST

2.2 CIFAR-10

[3]

Model	*Loss* pri klasifikaciji rezultata modela	Tačnost pri klasifikaciji rezultata modela	Identičnost rezultata klasifikacije pri smanjenju dimenzionalnosti
AE	0	0	0
KFN_MSE	0	0	0
KFN_CE	0	0	0
KFN_MID	0	0	0
KFN_CE_MSE_SUM	0	0	0
KFN_CE_MSE_AVG	0	0	0
KFN_MID_MSE_SUM	0	0	0
KFN_MID_MSE_AVG	0	0	0

Tabela 2: Rezultati evaluacije različitih modela na skupu podataka CIFAR-10

Zaključak

Literatura

[1] Pierre Baldi. Autoencoders, unsupervised learning, and deep architectures. In Isabelle Guyon, Gideon Dror, Vincent Lemaire, Graham Taylor, and Daniel Silver, editors, *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, volume 27 of *Proceedings of Machine Learning Research*, pages 37–49, Bellevue, Washington, USA, 02 Jul 2012. PMLR.

[2] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[3] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research).