Zahvaljujem mentoru prof. dr. sc. Siniši Šegviću na savjetima i strpljenju tijekom pisanja ovog rada.

Sadržaj

Uvod

Računalni vid (*eng. Computer Vision*) je područje umjetne inteligencije (*eng. Artificial Intelligence)* koje se bavi obradom, analizom i razumijevanjem slika. U posljednje vrijeme s dostupnošću sve većeg broja podataka te bolje infrastrukture za treniranje kompleksnih modela područje dubokog učenja preuzima sve najbolje rezultate u polju umjetne inteligencije te računalnog vida. U klasičnoj podjeli strojnog učenja modele možemo podijeliti na diskriminativne i generativne. Diskriminativni modeli koriste osobine podataka da dođu do zaključka kojoj kategoriji podatak pripada dok generativni modeli pokušavaju modelirati razdiobu ulaznih podataka s ciljem da tu distribuciju iskoriste za generiranje reprezentativnih podataka koji predstavljaju ulazne podatke. Problem kojim se bavimo u ovom radu je vezan uz generativne modele (*eng. Generative Models*) koji su jedan od dominantnih pristupa modeliranja podataka koji imaju kompleksnu strukturu. S napretkom dubokog učenja također dolazi do napretka generativnih modela te se rađa nova kategorija dubokih generativnih modela koji su kombinacija generativnih modela i dubokog učenja. Između popularnih dubokih generativnih modela možemo pronaći varijacijski autoenkoder (*eng. Variational Autoencoder*), modele zasnovane na suparničkom gubitku (*eng. Generative Adversial Network*), auto-regresivne modele (*eng. Auto-regressive Models*) te modele s normalizirajućim vjerojatnosnim tokom. Trenutno modeli zasnovani na suparničkom gubitku daju najbolje vizualno generirane podatke, ali takvi modeli su skloni kolapsirati modove i ne mogu računati izglednost podataka. S druge strane, generativni modeli koji optimiziraju izglednost kao što su varijacijski autoenkoder i model s normalizirajućim vjerojatnosnim tokom skloni su pretjerano generalizirati na način da mnogim slikama dodjele više vjerojatnosti nego što bi zapravo trebali. Ovaj rad je pokušaj da iskoristimo najbolje od oboje, da se postigne visoka kvaliteta generiranih podataka i pokrivenost svih modova razdiobe skupa za učenje.

1. Uvod u strojno učenje

Strojno učenje (eng. Machine Learning) je područje istraživanja umjetne inteligencije (eng. Artificial Intelligence) posvećeno razumijevanju i izgradnji metoda koje koriste podatke za poboljšanje performansi na nekom skupu zadataka. Sve većim brojem dostupnih podataka te razvojem grafičkih kartica dolazi do naglog napretka strojnog učenja, konkretno dubokog učenja jer omogućuje učenje kompleksnijih funkcija koje su potrebne da modeliramo kompleksnost podataka. Algoritmi strojnog učenja grade model na temelju danih podataka za učenje kako bi mogli predviđati ili donositi odluke bez izričitog programiranja. Model strojnog učenja predstavlja skup svih hipoteza H, odnosno funkcija preslikavanja h: X → Y.

$$H = \{ h(x;\ \theta) \}_\theta \tag{1.1}$$

Učenjem modela se smatra optimizacijom parametra θ. Strojno učenje možemo smatrati kao pretraživanje skupa svih hipoteza H u potrazi za najboljom funkcijom preslikavanja h ∈ H. Najboljom hipotezom smatramo onom koja daje točan, očekivani izlaz obzirom na ulaz modela. Postoje 3 glavne paradigme učenja: nadzirano učenje, nenadzirano učenje i potporno učenje.

Nenadzirano učenje je učenje za koje nemamo podatke o očekivanom izlazu modela, već samo podatke o ulazu. Najčešći problemi koje rješavamo nenadziranim učenjem su grupiranje, otkrivanje anomalija u podacima, smanjenje dimenzionalnosti podatka. Možemo svrstati i učenje generativnih modela u ovu kategoriju jer su nam za to potrebni samo podaci bez oznaka.

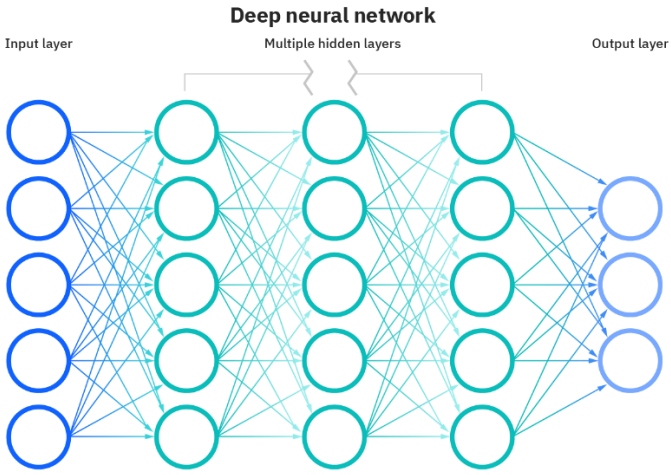
Nadzirano učenje je učenje za koje sa svakim ulaznim podatkom dolazi pripadajuća oznaka očekivanog izlaza. Oznake ulaznih podataka su ručno dodane podatcima i služe da damo povratnu informaciju modelu o tome u kojem smjeru se može poboljšati. Problemi koje najčešće rješavamo nadziranim učenjem su klasifikacija i regresija. Klasifikacija je kada za svaki primjer na ulazu model pokušava predvidjeti kategoriju kojem taj ulazni podatak pripada. Jedan primjer takvom problema je klasifikacija ručno pisanih znamenki u jedan od 10 kategorija za svaki broj od 0-9. Regresijski problem je kada za neki ulaz pokušavamo predvidjeti kontinuirani izlaz. Jedan primjer

takvog problema bi bio predviđanje cijena stanova obzirom na ulaze kao što su broj soba, lokacija i slično.

Potporno učenje je osposobljavanje modela strojnog učenja za donošenje niza odluka. Agent uči postići cilj u neizvjesnom, potencijalno složenom okruženju. Računalo metodom pokušaja i pogrješke dolazi korak po korak do rješenja problema. Algoritam se navodi na poboljšanje tako da definiramo nagradu ili kaznu za određene akcije ili stanja u kojima se agent nalazi. Cilj agenta je maksimizirati ukupnu nagradu. Uobičajeno je da se agent modelira Markovljevim lancem, a rješenje problema je sekvenca akcija koje agent mora napraviti. Primjeri koji se rješavaju ovom metodom su autonomna vožnja i igranje šaha.

1.1. Umjetne neuronske mreže

Umjetne neuronske mreže se nalaze u srži dubokih modela. Njihovo ime i struktura su inspirirani ljudskim mozgom, oponašajući način na koji biološki neuroni signaliziraju jedan drugome. Umjetne neuronske mreže sastoje se od niza slojeva neurona koji sadrže ulazni sloj, jedan ili više skrivenih slojeva te izlazni sloj. Najčešće korištena verzija umjetnih neuronskih mreža je potpuno povezana umjetna neuronska mreža koju karakterizira povezanost svakog neurona u jednom sloju sa svakim neuronom u idućem sloju neurona. Svaki neuron ima jedan ili više ulaza sa pripadnim težinama koje predstavljaju parametar učenja te aktivacijski prag koji definira da li će se vrijednost izlaza neurona propagirati na iduće neurone.



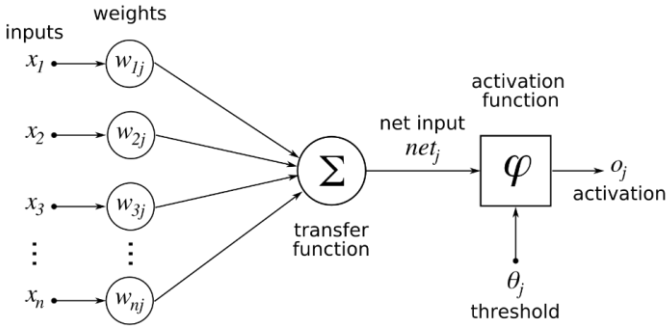
Slika 1.1: Umjetna neuronska mreža sa tri skrivena sloja

Umjetne neuronske mreže koriste podatke da bi poboljšale svoju točnost kroz vrijeme. Na jedan umjetni neuron u mreži možemo gledati kao na algoritam linearne regresije sa dodanom aktivacijskom funkcijom na kraju, sastavljen od ulaznih podataka, težina, pristranosti (eng. bias) i izlaza.

$$output = \sum_{i=1}^{N} w_i x_i + bias \tag{1.2}$$

Ako dodamo aktivacijsku funkciju dobijemo finalni izlaz neurona.

$$neuronOut = f(output) \tag{1.3}$$



Slika 1.2: Model funkcioniranja jednog neurona unutar neuronske mreže

Težine w_i određuju važnost određenog ulaza, što je vrijenost veća, više ulazni podatak vezan uz tu težinu utječe na izlaz neurona. Za učenje neuronske mreže važna su nam 2 pojma: funkcija gubitka i algoritam optimizacije od kojih ćemo objasniti onaj osnovni, algoritam povratne propagacije. Funkcija gubitka kvantificira razliku između očekivanog ishoda i ishoda dobivenog modelom strojnog učenja. Iz funkcije gubitka algoritmom povratne propagacije možemo izračunati gradijente gubitka u odnosu na težine u mreži (tj. koliko je koja težina w_i utjecala na finalni gubitak). Nakon što izračunamo gradijente, težine u mreži ažuriramo pomoću:

$$w_i = w_i - \alpha \frac{\partial Loss}{\partial w_i} \tag{1.4}$$

Gdje α predstavlja faktor učenja i predstavlja jedan od hiperparametara učenja. Ako izaberemo veliki faktor učenja nemamo garanciju da će učenje konvergirati u optimalnu vrijednost, dok ako izaberemo mali faktor učenja onda učenje može trajati duže nego što bi to bilo u optimalnijem slučaju.

1.2. Aktivacijske funkcije

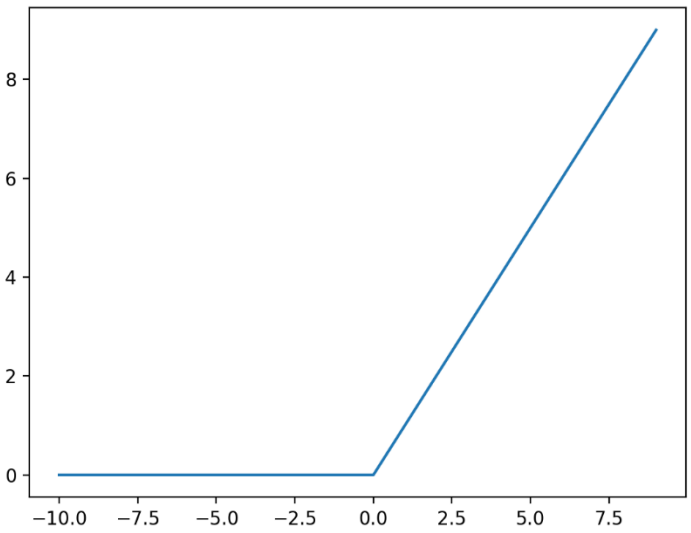
Aktivacijska funkcija u neuronskoj mreži definira kako se težinski zbroj ulaza pretvara u izlaz iz neurona u nekom sloju. Mnoge aktivacijske funkcije su nelinearne i upravo one su te koje uvode nelinearnost u neuronsku mrežu. Izbor aktivacijske funkcije ima veliki utjecaj na sposobnost i performanse neuronske mreže, a različite aktivacijske funkcije se mogu koristiti u različitim dijelovima modela neuronske mreže. Svi skriveni slojevi obično koriste istu aktivacijsku funkciju. Izlazni sloj će obično koristiti drugačiju aktivacijsku funkciju od skrivenih slojeva i ovisi o vrsti izlaza modela. Obzirom da se najčešće za učenje parametara koriste algoritmi povratne propagacije, važno je da se aktivacijska funkcija može derivirati. Još jedno važno svojstvo aktivacijski funkcija je da su nelinearne te time unose nelinearnost u neuronsku mrežu što je važno za formiranje kompleksnih funkcija. Postoji mnoštvo različitih aktivacijskih funkcija koje se koriste, iako se samo mali broj njih koristi u praksi. Za

skrivene slojeve u mreži se najčešće koriste ReLU, Sigmoid i Tanh, dok se za izlazne slojeve najčešće koriste linearna aktivacijska funkcija, Sigmoid i Softmax.

1.2.1. ReLU

Trenutno vjerovatno najkorištenija aktivacijska funkcija za skrivene slojeve. Jednostavna je za implementaciju i učinkovito rješava probleme ostalih aktivacijskih funkcija, manje je osjetljiva na problem nestajućeg gradijenta koji ograničava učenje dubokih modela. Nije ni ova aktivacijska funkcija bez svojih problema, kao na primjer problem umirućih neurona za koji postoje varijante koje pokušavaju eliminirati problem. ReLU funkciju računamo izrazom:

$$f(x) = \max(0, x) \tag{1.5}$$



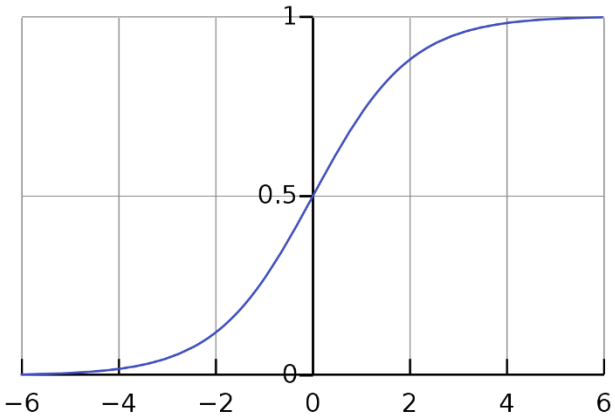
Slika 1.3: ReLU aktivacijska funkcija

1.2.2. Sigmoid

Prima na ulaz bilo koji broj na ulazu i daje broj između 0 i 1 na izlazu. Što je broj veći na ulazu to će izlaz biti bliže 1, dok što je broj na ulazu manji to će izlaz biti bliže 0. Sigmoid aktivacijska funkcija osim u skrivenom sloju se koristi i u izlaznom. Kada se

koristi u izlaznom sloju obično označava vjerojatnost pripadnosti nekom od razreda kada imamo samo 2 razreda izlaza. Sigmoid funkciju računamo izrazom:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{1.6}$$

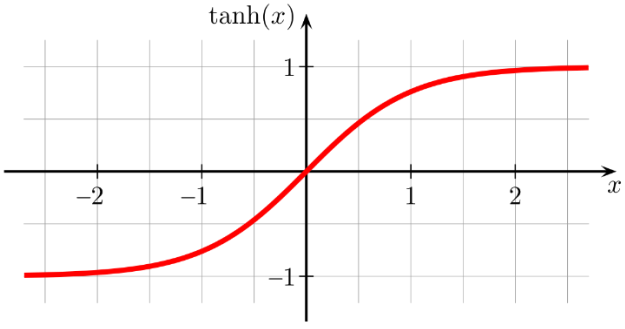


Slika 1.4: Sigmoid aktivacijska funkcija

1.2.3. Tanh

Tangens hiperbolni ili Tanh je aktivacijska funkcija vrlo slična Sigmoid funkciji uz najveću razliku da se ulaz preslikava u izlaz sa rasponom [-1,1]. Tanh funkciju računamo izrazom:

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{1.7}$$



Slika 1.5: Tanh aktivacijska funkcija

1.2.4. Linearna aktivacijska funkcija

Linearna aktivacijska funkcija jednostavno preslikava ulaz na izlaz. Ona se uglavnom koristi kada pokušavamo riješiti regresijski problem sa kontinuiranim izlazom. Linearna funkcija se računa izrazom:

$$f(x) = x \tag{1.8}$$

1.2.5. Softmax

Softmax aktivacijska funkcija se koristi u izlaznom sloju i daje na izlazu vektor koji kada se vrijednosti zbroje dobije se 1. Možemo na izlaz gledati kao vjerojatnost pripadanja ulaza nekom od predefiniranih izlaznih razreda. Ulaz je vektor brojeva, a izlaz je vektor s istom dimenzijom kojim zbrojem dobijemo 1. Softmax funkcija se računa izrazom:

$$f(x) = \frac{e^x}{sum(e^x)} \tag{1.9}$$

1.3. Funkcije gubitka

Umjetne neuronske mreže uče uz pomoć funkcija gubitka. Funkcija gubitka predstavlja metodu procjene koliko dobro određeni algoritam modelira dane podatke. Ako predviđanja previše odstupaju od stvarnih rezultata, funkcija gubitka bi na izlazu dala visok broj. Postupno, uz pomoć neke funkcije optimizacije, neuronska mreža uči kako smanjiti pogrešku u predviđanju i samim tim funkcija gubitka daje sve manji izlaz. Ne postoji funkcija gubitka koja ide uz sve tipove algoritama strojnog učenja. Funkcije gubitka se mogu klasificirati u dvije glavne kategorije ovisno o vrsti zadatka učenja s kojim se bavimo, regresijski gubici i gubici klasifikacije. Najčešće korištene funkcije koje možemo svrstati u regresijske funkcije gubitka su srednja kvadratna pogreška (MSE) i srednja apsolutna pogreška (MAE), dok u klasifikacijske funkcije

gubitka možemo svrstati funkciju gubitka unakrsne entropije i vrlo sličnu funkciju gubitka zvanu negativnu log vjerojatnost koja se najčešće smatra istom metodom.

1.3.1. Srednja kvadratna pogreška

Srednja kvadratna pogreška se koristi u regresijskim problemima u kojima očekujemo kontinuiranu vrijednost na izlazu. Računa se kao prosjek kvadratne razlike između izlaza neuronske mreže i stvarne, očekivane vrijednosti. Zbog kvadriranja, predviđanja koja su daleko od stvarnih vrijednosti više su kažnjavana u usporedbi s predviđanjima koja manje odstupaju. Uz sve to ovaj funkcija gubitka ima matematička svojstva koja olakšavaju računanje gradijenta. Funkcija gubitka srednje kvadratne pogreške računa se izrazom:

$$MSE = \frac{\sum_{i=1}^{n}(y_i - y_i')^2}{n} \tag{1.10}$$

1.3.2. Srednja apsolutna pogreška

Funkcija gubitka vrlo slična funkciji srednje kvadratne pogreške uz razliku da računamo srednju apsolutnu pogrešku. Za ovu metodu je nedostatak jer je zahtjevnije izračunati gradijent, ali je ova funkcija otpornija na anomalije jer ne računa kvadrat pogreške. Funkcija gubitka srednje apsolutne pogreške računa se izrazom:

$$MAE = \frac{\sum_{i=1}^{n}|y_i - y_i'|}{n} \tag{1.11}$$

1.3.3. Funkcija gubitka unakrsne entropije (negativna log vjerojatnost)

Funkcija gubitka unakrsne entropije se koristi u klasifikacijskim problemima. Entropija predstavlja broj bitova potrebnih za prijenos slučajno odabranog događaja iz distribucije vjerojatnosti. Unakrsna entropija se temelji na ideji entropije iz teorije

informacija i izračunava broj bitova potrebnih za predstavljanje ili prijenos prosječnog događaja iz jedne distribucije u usporedbi s drugom distribucijom. Razlikujemo binarni gubitak unakrsne entropije i više generalnu verziju za gubitak unakrsne entropije više razreda. Binarni gubitak unakrsne entropije računa se izrazom:

$$CrossEntropyLoss = -(y_i \log(y_i') + (1 - y_i) log(1 - y_i')) \quad (1.12)$$

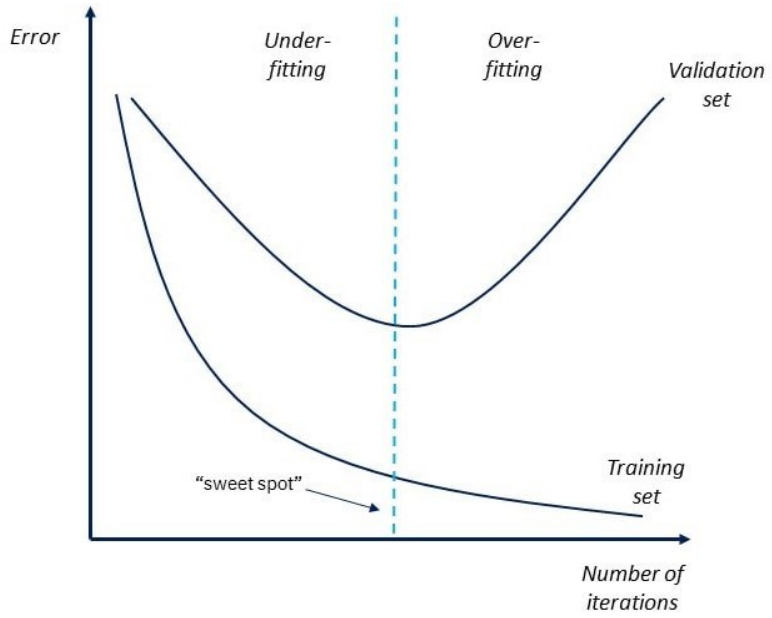
Gubitak unakrsne entropije za više razreda računa se izrazom:

$$CrossEntropyLoss = -\sum_{i=1}^{C} y_i \log(y_i') \quad (1.13)$$

Funkcija gubitka unakrsne entropije na ulazu očekuje vjerojatnosnu razdiobu za svaki izlazni razred. Ponekad postoji razlika između funkcije gubitka unakrsne entropije i negativne log vjerojatnosti kao npr. u pytorch implementaciji gdje funkcija gubitka unakrsne entropije na ulazu očekuje logite te interno računa vjerojatnosnu razdiobu iz tog dok funkcija gubitka negativne log razdiobe očekuje odmah vjerojatnosnu razdiobu na ulazu.

1.4. Algoritmi optimizacije

Algoritmi optimizacije su pojam vrlo blizak strojnom učenju i funkciji gubitka. Za probleme strojnog učenja obično prvo definiramo funkciju gubitka te nakon toga možemo koristiti algoritam optimizacije u pokušaju da minimiziramo gubitak. U optimizaciji, funkcija gubitka se često naziva ciljnom funkcijom optimizacijskog problema. Većina algoritama optimizacije bavi se problemom minimizacije funkcije gubitka. Može se činiti da ova metoda ne ide bez svojih problema, ali važno je pripaziti na prenaučenost modela jer pogreška treninga i generalizacije se obično razlikuju. Do prenaučenosti dolazi kada pogreška treniranja, koja je zapravo ciljna funkcija algoritma optimizacije se smanjiva dok se pogreška generalizacije u isto vrijeme povećava. Do toga dolazi jer se model počeo previše prilagođavati podacima za treniranje.



Slika 1.6: Vizualiziranje prenaučenosti modela

Osnovni algoritam za algoritme optimizacije se zove algoritam gradijentnog spusta. Iako se ne koristi više u toj verziji u rješavanju problema strojnog učenja danas, važno ga je razumjeti jer mnogi algoritmi u bazi imaju baš ovaj algoritam i njegovu ideju.

1.4.1. Algoritam gradijentnog spusta

Algoritam gradijantnog spusta je iterativni algoritam koji počinje od slučajne točke na nekoj funkciji i putuje niz njen nagib malim koracima dok ne dođe do minimuma te funkcije. Gradijentno spuštanje u jednoj dimenziji dobar je primjer za objasniti zašto algoritam gradijentnog spusta može smanjiti vrijednost funkcije gubitka. Razmotrimo neku kontinuirano diferencijabilnu realnu funkciju $f: \mathbb{R} \to \mathbb{R}$. Koristeći Taylorovu ekspanziju dobivamo:

$$f(x+\varepsilon) = f(x) + \varepsilon f'(x) + O(\varepsilon^2) \tag{1.14}$$

U aproksimaciji prvog reda $f(x+\varepsilon)$ dana je vrjednošću funkcije $f(x)$ i prve derivacije $f'(x)$. Iz ovoga možemo pretpostaviti da za mali ε sa pomakom prema negativnom gradijentu će smanjiti vrijednost funkcije. Ako izaberemo neku malu

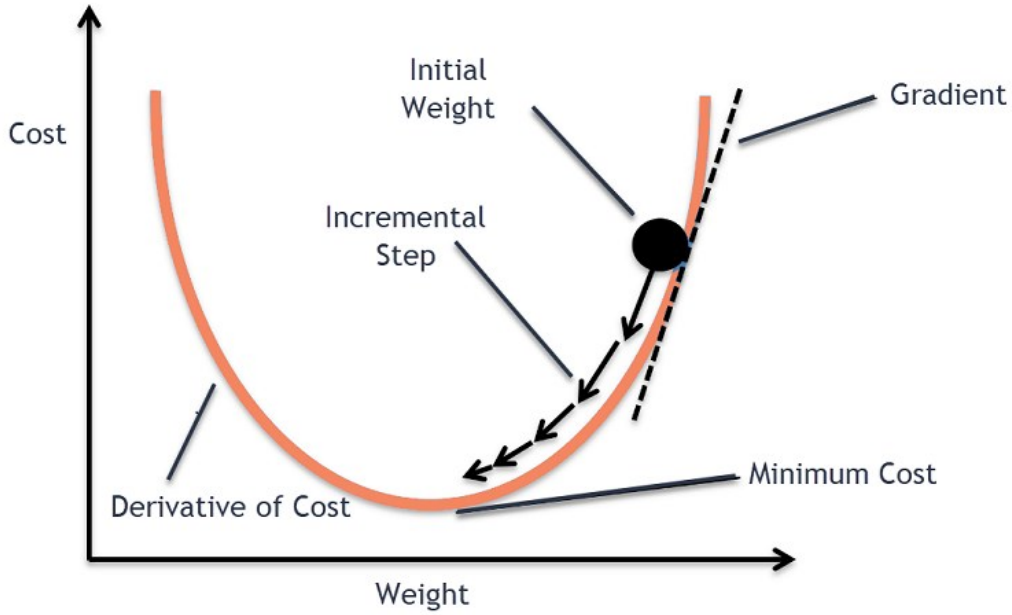
konstantu pomaka $\alpha > 0$ i zamjenimo $\varepsilon = -\alpha f'(x)$ i uvrstimo zamjenu u Taylorovu ekspanziju dobivamo:

$$f(x - \alpha f'(x)) = f(x) - \alpha f'^2(x) + O(\alpha^2 f'^2(x)) \quad (1.15)$$

Ako je prva derivacija $f'(x) \neq 0$ smanjivamo vrijednost funkcije jer $\alpha f'^2(x) > 0$. Ako izaberemo malu vrijednost α te time možemo zanemariti doprinos $O(\alpha^2 f'^2(x))$ i dobivamo:

$$f(x - \alpha f'(x)) \leq f(x) \quad (1.16)$$

To znači ako primijenimo $x \leftarrow x - \alpha f'(x)$, možemo očekivati da će funkcija $f(x)$ imati padajuću vrijednost kroz iteracije. U algoritmu gradijentnog spusta prvo izaberemo inicijalnu vrijednost od x i konstantu $\alpha > 0$. Nakon toga ih koristimo da iteriramo x vrijednostima do nekog uvjeta zaustavljanja koji na primjer može biti minimalna vrijednost derivacije ili broj iteracija.



Slika 1.7: Funkcioniranje gradijentnog spusta

Vrijednost α se u području strojnog učenja naziva faktor učenja i on predstavlja jedan od osnovnih hiperparametara modela. Odabir prave vrijednosti faktora učenja je važno

jer premala vrijednost može značiti da će vrijeme učenja biti puno duže nego as optimalnom vrijednosti zbog malog gradijentnog pomaka u svakoj iteraciji. Suprotno tome, ako koristimo visoku stopu učenja, jedan od faktora Taylorove ekspanzije $O(\alpha^2 f'^2(x))$ više nije zanemariv i ne možemo garantirati da će funkcija uvijek imati padajuću vrijednost.

Sada kada razumijemo gradijentni spust u jednoj dimenziji, možemo to generalizirati na više dimenzija. Pretpostavimo da imamo n ulaznim vrijednosti $x = [x_1, x_2, \dots, x_n]$ te funkciju $f: \mathbb{R}^n \to \mathbb{R}$. Obzirom da imamo više dimenzija, dobijemo da je gradijent u više dimenzija, vektor koji se sastoji od n parcijalnih derivacija:

$$\nabla f(x) = \left[\frac{\partial f(x)}{\partial x_1}, \frac{\partial f(x)}{\partial x_2}, \dots, \frac{\partial f(x)}{\partial x_n}\right] \tag{1.17}$$

Svaka parcijalna derivacija $\frac{\partial f(x)}{\partial x_i}$ označava stopu promjene funkcije u odnosu na ulaz x_i. Sukladno gradijentnom spustu u jednoj dimenziji, Taylorovom ekspanzijom u više dimenzija možemo doći do zaključka da do minimuma funkcije dolazimo pomakom u smjeru negativnog gradijenta:

$$x \leftarrow x - \alpha \nabla f(x) \tag{1.18}$$

Izborom faktora učenja $\alpha > 0$ dolazimo do osnovne formule algoritma gradijentnog spusta.

1.4.2. Stohastički gradijentni spust

Stohastički gradijentni spust je algoritam nastao u pokušaju da se riješe neki od problema sa algoritmom gradijentnog spusta koji ograničavaju treniranje zahtjevnijih modela sa velikim brojem podataka. Algoritam gradijentnog spusta ima 2 glavna nedostatka:

- Računanje gradijenta za cijeli skup učenja je vremenski zahtjevno
- Potreba za memorijom je jednaka veličini skupa za učenje

Obzirom da se u algoritmu gradijentnog spusta u jednoj iteraciji gradijenti se računaju za sve podate za treniranje, kako se skup podataka povećava, tako raste i računska složenost i vrijeme svake iteracije se povećava. Da bi efikasno riješili spomenute probleme nastao je algoritam stohastičkog gradijentnog spusta.

Stohastički gradijentni spust je vjerojatnosna aproksimacija gradijentnog spusta. To je aproksimacija jer u svakom koraku algoritam izračunava gradijent za samo jedan nasumični ulazni podatak umjesto računanja za cijeli skup ulaznih podataka. Ovime postižemo značajno poboljšanje u performansama kada skup podataka ima veliki broj podataka koji se može brojiti u milijunima. Zbog ovog pojednostavljenja dolazi i do nekih negativnih strana. Obzirom da računamo gradijent samo za jedan ulazni podataka u svakoj iteraciji dolazi do toga da ažuriranja imaju veću varijancu. Zbog toga funkcija gubitka više fluktuira kroz svaku iteraciju što otežava konvergiranje algoritma u usporedni s gradijentnim spustom.

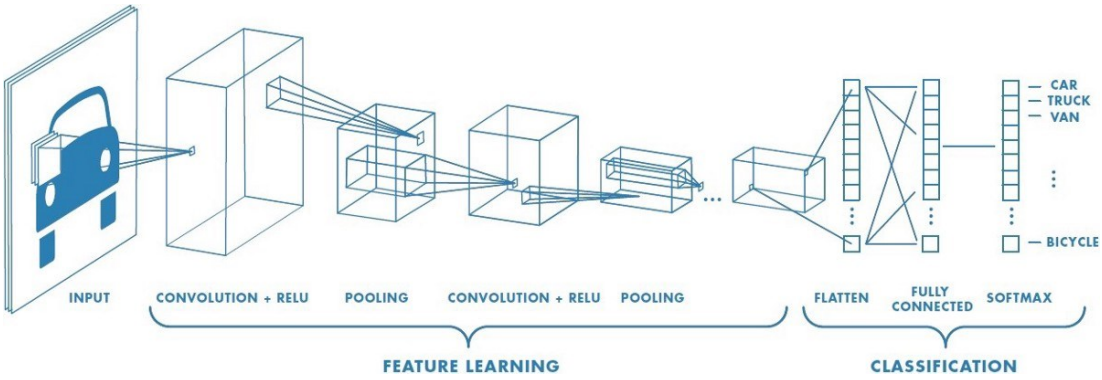
Nove varijante stohastičkog gradijentnog spusta su razvijene da dobijemo prednosti i od gradijentnog spusta i od stohastičkog gradijentnog spusta. Danas najčešće korištena varijanta stohastičkog gradijentnog spusta je mini-batch stohastički gradijentni spust gdje se u svakom koraku računa gradijent za n ulaznih podataka u odnosu na prije samo jedan čime postižemo bolju aproksimaciju i ujedno bolju konvergenciju.

Danas se kao optimizacijski algoritmi koriste proširenja opisanog algoritma stohastičkog gradijentnog spusta koji mogu značajno ubrzati vrijeme učenja korištenjem optimizacijskih metoda kao korištenjem zasebne stope učenja za svaku težinu u mreži koja se kroz učenje ažurira i slične metode.

1.5. Konvolucijske neuronske mreže

Pojavom konvolucijskih neuronskih mreža dolazi do velikog napretka u području računalnog vida. Arhitektura konvolucijskih neuronskih mreža inspirirana je organizacijom vizualnog korteksa. Pojedinačni neuroni reagiraju na podražaje samo u ograničenom području vidnog polja koje nazivamo receptivnim poljem. Konvolucijska neuronska mreža je algoritam dubokog učenja koji ima sposobnost za

neku ulaznu sliku važnost različitim objektima na slici i biti sposoban razlikovati jedan od drugih. Dok su se prije filtri koji prepoznaju određenje oblike ili uzorke na slici bili ručno konstruirani, uz dovoljno učenja konvolucijske neuronske mreže imaju sposobnost naučiti te iste filtere, najčešće i bolje. Konvolucijske neuronske mreže za probleme računalnog vida su puno bolje rješenje od potpuno povezanih neuronskih mreža jer imaju mogućnost odrediti prostorne i vremenske ovisnosti na slici primjenom relevantnih filtera. Obzirom da slike na ulazu mogu imati veliki broj podataka potpuno povezane mreže bi bile vrlo računalno zahtjevne zbog velikog broja potrebnih parametara. Konvolucijske neuronske mreže imaju prednost jer se koristi puno manji broj parametara u filtrima i oni se mogu ponovno upotrijebiti za sve dijelove slike. Kako slika napreduje kroz mrežu, filtri su u mogućnosti naučiti kompleksnije uzorke. Na primjer ako u prvom sloju imamo naučen filter koji prepoznaje boju ili rubove, u drugom sloju već možemo prepoznati kompleksnije oblike sastavljene od rubova i boja primjenjujući novi filter na rezultat prijašnjeg.



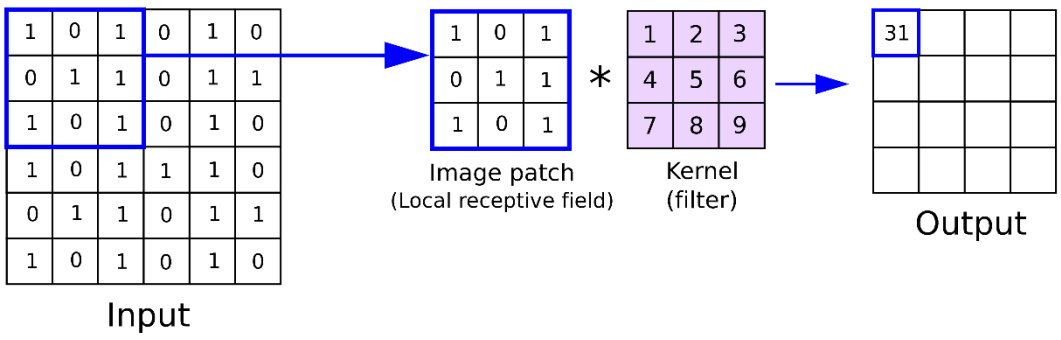
Slika 1.8: Konvolucijska neuronska mreža

Konvolucijske neuronske mreže se sastoje od tri glavna tipa slojeva:

- Konvolucijski slojevi
- Slojevi sažimanja
- Potpuno povezani slojevi

1.5.1. Konvolucijski sloj

Konvolucijski sloj temeljni je građevni bok konvolucijskih neuronskih mreža i na njemu se odvija većina izračuna. Za konvolucijske slojeve važni su nam pojmovi ulaznih podataka, filtera te mapi značajki. Ako pretpostavimo da je ulaz RGB slika u boji, to znači da je ulaz konstruiran od trodimenzionalne matrice vrijednost piksela. Konvolucijski sloj se također sastoji od detektora značajki kojeg najčešće nazivamo filtar koji funkcioniraju tako da se kreću kroz receptivno polje slike provjeravajući da li je značajka prisutna u određenom dijelu slike. Opisani proces nazivamo proces konvolucije. Filter je obično dvodimenzionalni niz težina kojim predstavljamo dio slike za koji pretražujemo određenu značajku. Iako filteri mogu varirati u veličini, za dimenziju se najčešće uzima matrica manjih dimenzija što ujedno određuje i dimenziju receptivnog polja, na primjer matrica dimenzije 3x3. Nakon što smo odredili dimenziju filtera, tada ga primjenjujemo na područja slike, te za svako područje računamo izlaz koji se računa najčešće kao umnožak odgovarajućih vrijednosti područja slike sa vrijednosti filtera. Filter pomičemo korak po korak kroz ulaznu sliku, ponavljajući proces dok filter ne prijeđe preko cijele slike. Konačni izlaz svih pomaka poznat je kao mapa značajki, gdje nam ime govori da se radi o izlazu koji predstavlja vrijednost za pronađenu značajku u svakom dijelu slike.



Slika 1.9: Konvolucija nad receptivnim poljem

Kao što možemo vidjeti, svaka izlazna vrijednost u mapi značajki ne povezuje se sa svakom vrijednošću piksela na ulaznoj slici, već samo s receptivnim poljem filtera. To

nam je važno jer tako efektivno smanjujemo broj potrebnih parametara u mreži i to nam omogućuje lokalnu ekstrakciju značajki u slici. Tijekom pomicanja filtera kroz ulaz vrijednosti težina filtera ostaju fiksne te se procesom gradijentnog spusta prilagođavaju u nove vrijednosti za koje očekujemo da ćemo dobiti bolje rezultate na izlazu u idućoj iteraciji što znači da će bolje predstavljati filter koji detektira one značajke koje će pomoći u donošenju finalne odluke. Prije učenja konvolucijskih slojeva, moramo odrediti vrijednosti hiperparametara koji određuju kako će se konvolucijski sloj ponašati. Osnovni i najznačajniji hiperparametri konvolucijskih slojeva su:

- Broj korištenih filtera – utječe na dimenziju izlaza mapi značajki
- Dimenzija filtera
- Korak pomaka filtera kroz ulaz – broj piksela kojim se filter pomiče kroz ulaznu matricu
- Ispunjavanje okolnih vrijednosti – obično se radi popunjavanje nulama. Ovime efektivno povećavamo dimenziju slike i može biti korisno ako želimo da ulazna vrijednost za svaku konvoluciju bude u sredini filtera, što neće inicijalno vrijediti za rubne elemente bez korištenja ispunjavanja.

Odabrani hiperparametri određuju dimenziju izlaznih mapi značajki:

$$H_{out} = \left\lfloor \frac{H_{in} + 2 * ispunjenje[0] - filterDimenzija[0] - 2}{pomak[0]} + 1 \right\rfloor \tag{1.19}$$

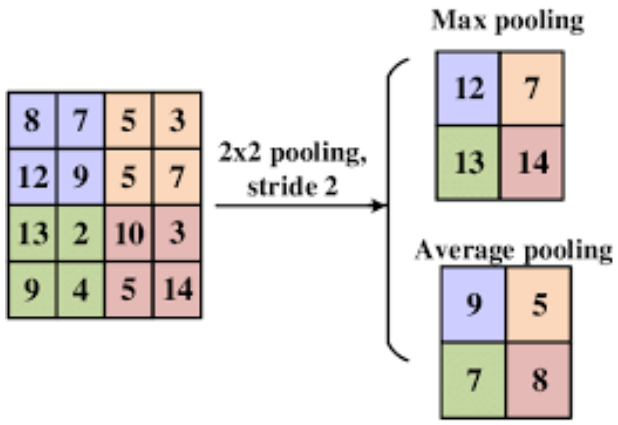
$$W_{out} = \left\lfloor \frac{W_{in} + 2 * ispunjenje[1] - filterDimenzija[1] - 2}{pomak[1]} + 1 \right\rfloor \tag{1.20}$$

Nakon konvolucijskog sloja primjenjujemo aktivacijsku funkciju da uvedemo nelinearnost u mrežu, najčešće ReLU. Kao što smo spomenuli, više konvolucijskih slojeva može biti iza inicijalnog sloja te tako efikasno možemo detektirati kompleksnije značajke. Ako uzmemo primjer auta, konvolucijski slojevi početnih slojeva će prepoznati jednostavne oblike kao linije i krugove dok će u kasnijim slojevima moći prepoznati kotač, oblik auta i slično.

1.5.2. Sloj sažimanja

Sloj sažimanja ima ulogu da provede smanjenje dimenzionalnosti tako da smanji broj parametara na ulazu. Slično konvolucijskom sloju, operacija sažimanja provodi filter kroz matricu ulaza korak po korak, ali velika razlika je u tome što filteri u sloju sažimanja nemaju težine i nemaju parametre koji se uče. Filteri u sloju sažimanja uvijek rade istu operaciju agregacije unutar receptivnog polja slike. U praski se najčešće koriste dvije vrste agregacijske funkcije:

- Sažimanje maksimalnom vrijednosti – uzimamo maksimalnu vrijednost iz receptivnog polja filtera i šaljemo rezultat na izlaz
- Sažimanje prosječnom vrijednosti – uzimamo prosječnu vrijednost



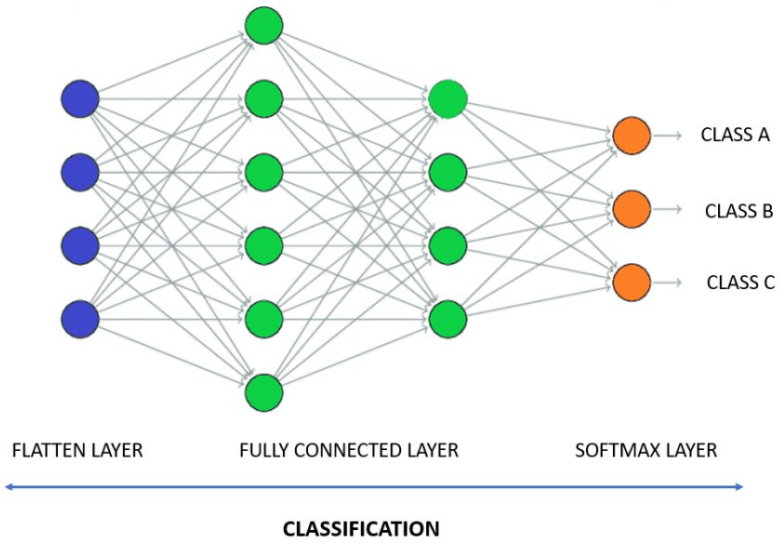
Slika 1.10: Dvije vrste sažimanja

Možemo primijetiti da se puno informacije izgubi u sloju sažimanja, ali ovaj sloj ujedno pomaže smanjiti složenost modela, poboljšava učinkovitost i pomaže da izbjegnemo prenaučenost modela.

1.5.3. Potpuno povezani sloj

Ovaj sloj ima identičnu arhitekturu neuronskim mrežama koje smo opisali prethodno u umjetnim neuronskim mrežama sa svojstvom da je svaki neuron prethodnog sloja

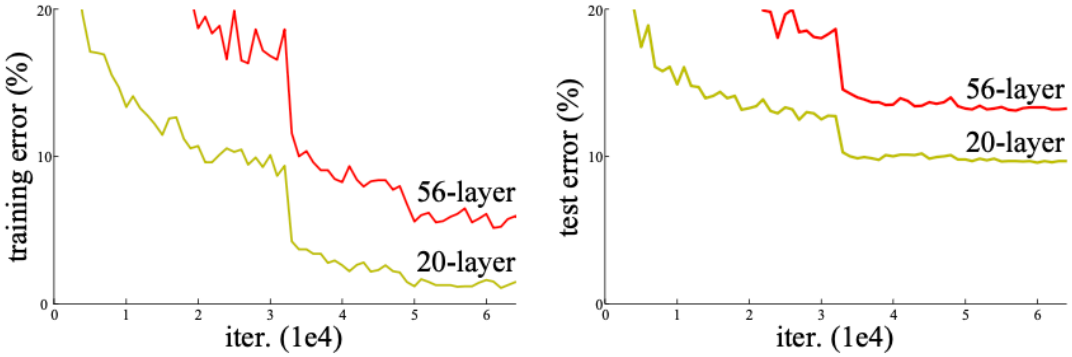
povezan sa svakim u idućem sloju. Ulaz u potpuno povezani sloj dobivamo tako da sve mape značajki „izravnamo“ u jednu dimenziju pogodnu za ovakvu mrežu. Funkcija ovog sloja je donošenje klasifikacijske odluke na temelju značajki koje smo izvukli u prethodnim dijelovima mreže. Kao aktivacijsku funkciju izlaznog sloja obično koristimo softmax da dobijemo vjerojatnosnu razdiobu za svaki razred klasifikacije.



Slika 1.11: Klasifikacija u potpuno povezanom sloju

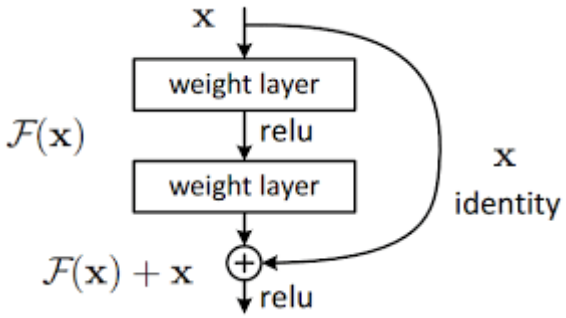
1.5.4. Arhitektura ResNet

Do pojave ResNet mreža kod svih popularnih arhitektura vrijedilo je pravilo da su performanse modela proporcionalne dubini modela (broju slojeva u mreži). Možemo se zapitati da li uvijek vrijedi da više slojeva znači da ćemo dobiti bolje rezultate što nije uvijek točno. Zbog fenomena nestajućeg gradijenta u dubokim modelima povećanje broje slojeva može naštetiti konvergenciji modela od početka.



Slika 1.12: Model sa 56 slojeva pokazuje lošije rezultate od onog sa 20 slojeva zbog problema nestajućeg gradijenta

Ova slika nam pokazuje kako model sa 56 slojeva pokazuje lošije rezultate od onog sa 20 upravo zbog problema nestajućeg gradijenta. U pokušaju da se izbjegne ovaj problem, predložena je duboka konvolucijska neuronska mreža nazvana ResNet. Ova mreža je konstruirana od većeg broja rezidualnih blokova. U mrežama sa rezidualnim blokovima, svaki sloj ulazi u idući sloj i izravno u slojeve udaljena 2-3 sloja.



Slika 1.13: Rezidualni blok

Rezidualni blokovi omogućuju protok memorije (informacije) od početnog do zadnjeg sloja te tako omogućuju puno dublje modele nego inicijalno bez tog svojstva. Neka od svojstava ResNet modela:

- Svi filteri unutar konvolucijskih slojeva imaju dimenziju 3x3
- Broj filtera raste sa dubinom mreže, od 64 pa sve do 2048 u nekim verzijama dubokih modela

- Koristi se samo jedan sloj sažimanja sa maksimalnom vrijednošću
- Na kraju modela koristi se sloj sažimanja srednjom vrijednošću kao zamjena za potpuno povezani sloj. Prednosti zamjene su smanjena kompleksnost modela jer nema parametara za učenje u završnom sloju. Još jedna prednost bi bila bolje uspostavljanje korespondencije između mapa značajki i izlaza modela.

2. Generativni modeli

Cilj generativnih modela je naučiti reprezentaciju vjerojatnosne distribucije $\chi \varepsilon \mathbb{R}^n$, gdje je n obično velik i gdje je distribucija kompleksna. U tu svrhu možemo koristiti potencijalno velik, ali tipično ograničen broj neovisnih uzorka iz χ koje nazivamo podacima za učenje. Za razliku od standardnog statističkog zaključivanja gdje se traži matematički izraz za vjerojatnost, cilj je dobiti generator:

$$g: \mathbb{R}^q \rightarrow \mathbb{R}^n \tag{2.1}$$

koji mapira uzorke iz kontrolirane distribucije $\mathbb{Z}$ generirane iz $\mathbb{R}^q$ do uzoraka iz $\mathbb{R}^n$ koji sliče danim podacima za učenje. Pretpostavljamo da za svaki uzorak $x \sim \chi$ postoji bar jedna točka $z \sim \mathbb{Z}$ tako da vrijedi $g(z) \approx x$. Generator koji može preslikavati točke iz jednostavne distribucije $\mathbb{Z}$ u kompleksnu distribuciju χ, omogućuje nam generiranje uzoraka iz χ što je poželjno svojstvo u mnogim primjenama. Obzirom da vektor z koji se koristi za generiranje vektora x obično nije poznat, obično ga nazivamo latentnom varijablom i $\mathbb{Z}$ latentnim prostorom. Za $\mathbb{Z}$ možemo uzeti bilo koju kontroliranu distribuciju, ali obično se uzima Normalna distribucija u $\mathbb{R}^q$. Od distribucije zahtijevamo da možemo generirati uzorke od te distribucije i u nekim slučajevima želimo mogućnost izračuna vjerojatnosti $p_{\mathbb{Z}}(z)$. Obično dimenzija latentnog prostora q će biti drugačija od dimenzije podataka n. U mnogim primjenama generativnih modela jedini cilj je generiranje novih uzoraka. Osim toga, generator također može biti korišten za izračun vjerojatnosti određenog uzorka x:

$$p_{\chi}(x) = \int p_g(x|z) p_{\mathbb{Z}}(z) dz \tag{2.2}$$

gdje $p_g(x|z)$ mjeri sličnost između $g(z)$ i x. Točno izračunavanje vjerojatnosti uzorka x općenito je vrlo teško moguće zbog visoke dimenzionalnosti integrala. Izbor vjerojatnosne funkcije $p_g(x|z)$ ovisi o svojstvima podataka među kojima najčešći izbor su normalna i Bernoullijeva razdioba. Izvođenje generatora g neizvedivo je za većinu skupova podataka od interesa. Zbog toga je posljednjih godina postalo uobičajeno koristiti funkcije aproksimacije kao što su duboke neuronske mreže. To je temeljni koncept kod dubokih generativnih modela, gdje je generator modeliran sa

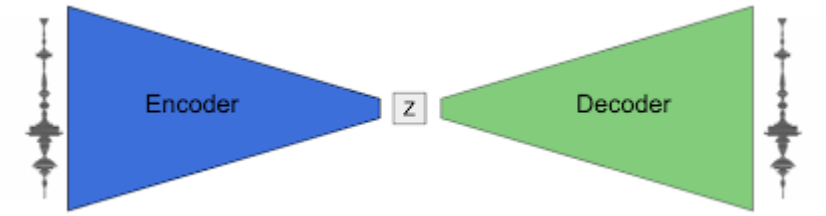
dubokom neuronskom mrežom. Prednosti ovakvih mreža su efikasna mogućnost aproksimacije funkcija u visokoj dimenziji. Ovakve generatore označavamo sa g_θ i njegove težine sa $\theta \, \varepsilon \, \mathbb{R}^{N_\theta}$. Predstavit ćemo tri glavna pristupa učenju dubokog generativnog modela g_θ koristeći primjere iz χ. Cilj dubokih generativnih modela je naučiti parametre mreže θ tako da generirani uzorci $g_\theta(z)$ se statistički ne razlikuju od primjera za učenje modela. Drugim riječima, trenirano model tako da mapira latentnu vjerojatnosnu distribuciju $Ƶ$ u vjerojatnosnu distribuciju podataka χ. Postoje razni načini u modeliranju generativnih modela od kojih ćemo spomenuti tri vrlo poznata pristupa:

- Varijacijski autoenkoder
- Model zasnovan na suparničkom gubitku
- Model s normalizirajućim vjerojatnosnim tokom

2.1. Varijacijski autoenkoder

Varijacijski autoenkoder je autoenkoder čija se distribucija kodiranja regularizira tijekom učenja kako bi se osiguralo da njegov latentni prostor ima dobra svojstva koja nam omogućuju generiranje novih podataka. Izraz varijacijski dolazi iz bliske povezanosti koja postoji između regularizacije i metode varijacijskog zaključivanja u statistici. Da bismo shvatili kako funkcioniraju varijacijski autoenkoderi prvo ćemo objasniti što je to pojam smanjenja dimenzionalnosti. Smanjenje dimenzionalnosti je proces smanjenja broja značajki koje opisuju određene podatke. Smanjenje broja značajki se ostvaruje odabirom (sačuvaju se samo neke postojeće značajke) ili ekstrakcijom (smanjeni broj novih značajki se kreira na temelju starih značajki) i može biti korisno u mnogim situacijama koje zahtijevaju podatke niže dimenzije. Proces koji proizvodi nove značajke iz značajki podataka zovemo enkoder, a dekoder obrnuti proces. Sada kada to znamo na smanjenje dimenzionalnosti možemo gledati kao na kompresiju podataka iz početnih podataka u kodirani prostor, kojeg također nazivamo latentnim prostorom, koristeći enkoder. Dekoder je obrnuti proces pokušaja dekodiranja latentnog prostora u originalne podatke. Kažemo pokušaj dekodiranja jer ovisno o početnoj distribuciji podataka, dimenziji latentnog prostora i definiciji

enkodera kompresija može biti s gubitkom što znači da se dio informacija gubi tijekom procesa kodiranja i ne može se oporaviti tijekom procesa dekodiranja.



Slika 2.1: Enkoder kodira u latentnu varijablu z, dekoder pokušava rekonstruira z u ulaznu vrijednost

Glavna svrha metode smanjenja dimenzionalnosti je pronaći najbolji par kodera-dekodera među mogućim parovima. Tražimo onaj par koji zadržava maksimum informacija prilikom kodiranja i samim time ima najmanju pogrešku rekonstrukcije tijekom dekodiranja. Problem smanjenja dimenzionalnosti možemo pisati kao:

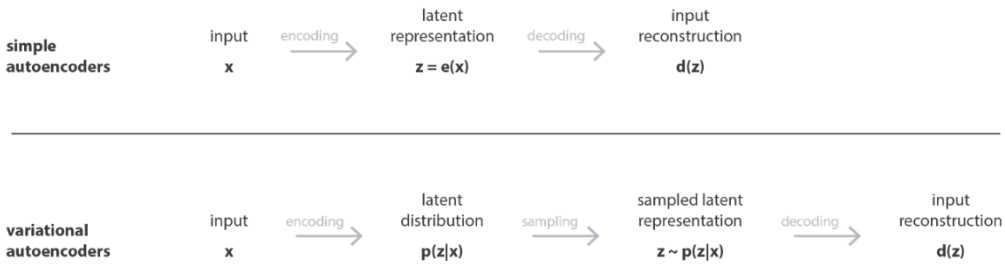
$$(e^*, d^*) = \underset{(e,d)\varepsilon ExD}{\operatorname{argmin}}\ error(x, d(e(x))) \tag{2.3}$$

gdje $error(x, d(e(x)))$ označava pogrešku rekonstrukcije podataka između ulaznih podataka i enkodirano-dekodiranih podataka. Sada kada znamo kako funkcionira problem smanjenja dimenzionalnosti možemo definirati što je to autoenkoder te kako se neuronske mreže koriste za smanjenje dimenzionalnosti. Ideja autoenkodera je prilično jednostavna i sastoji se od modeliranja enkodera i dekodera pomoću neuronskih mreža i učenja optimalnog načina kodiranja i dekodiranja iterativnim procesom optimizacije. U svakoj iteraciji učenja podatke kodiramo i dekodiramo, računamo grešku usporedbom dobivene vrijednosti sa originalnom očekivanom vrijednosti i optimiziramo parametre obje mreže propagacijom dobivene pogreške unatrag. Intuitivno možemo vidjeti da obje mreže rade zajedno kako bi pronašli glavni dio informacija koji opisuju ulazne podatke. Pretpostavimo prvo da modeli enkodera i dekodera imaju samo jedan sloj bez nelinearnosti. Tada se enkoder i dekoder ponašaju kao matrice koje optimalno preslikavaju u neki prostor niže dimenzije sa što manje gubitka informacije. Ako pretpostavimo da modeli enkodera i dekodera su duboke neuronske mreže sa nelinearnošću tada što je arhitektura složenija to autoenkoder može povećati redukciju dimenzionalnosti uz zadržavanje niskog gubitka

rekonstrukcije. Uz dovoljno složene modele autoenkodera moguće je svesti ulaz u latentni prostor s jednom dimenzijom. Važno je napomenuti da smanjenje dimenzionalnosti bez gubitka rekonstrukcije ima svoju cijenu koja se očituje kao nedostatak interpretabilnih i iskoristivih struktura u latentnom prostoru. Također uglavnom konačni cilj smanjenja dimenzionalnosti nije samo smanjiti broj dimenzija podataka, već smanjiti broj dimenzija uz uvjet da se pritom zadrži osnovni dio informacija o strukturi podataka. Iz ovih razloga dimenzija latentnog prostora i složenost autoenkodera moraju se prilagođavati ovisno o konačnom cilju smanjenja dimenzionalnosti.

Sada kada smo objasnili osnovne pojmove vezane uz autoenkoder možemo pokazati što je to varijacijski autoenkoder i kako nam on pomaže kod generiranja novih primjera. Problem s autoenkoderom je što kada optimiziramo parametre enkodera i dekodera čak ni tada nemamo način za generirati podatke koji će imati sličnu razdiobu kao ulazni podaci. Na prvi pogled možemo pomisliti ako je latentni prostor dovoljno organiziran od strane enkodera tijekom obuke da bi tada mogli uzeti nasumičnu točku u prostoru i koristeći dekoder generirati novi sadržaj. Međutim, pravilnost latentnog prostora za autoenkodere je kompleksna jer ovisi o distribuciji podataka na ulazu, dimenzionalnosti latentnog prostora i složenosti autoenkodera. Vrlo je nevjerojatno očekivati da će enkoder organizirati na pravilan način kompatibilan s generativnim procesom. Nedostatak strukture među kodiranim podacima u latentnom prostoru prilično je normalan zbog toga što ništa u zadatku za koji je autoenkoder zadužen ne zahtjeva od njega takvu organizaciju, jedini zadatak mu je enkodiranje i dekodiranje uz što manji gubitak informacije. Ako nismo pažljivi oko organizacije arhitekture prirodno je da tijekom učenja mreža iskorištava mogućnost prekomjernog prilagođavanja ulaznim podacima osim ako to eksplicitno ne reguliramo. Da bismo mogli koristiti dekoder u generativne svrhe moramo biti sigurni da je latentni prostor dovoljno pravilan. Jedno od mogućih rješenja za postizanje pravilnosti je uvođenje eksplicitne regularizacije tijekom učenja. Varijacijski autoenkoder možemo definirati kao autoenkoder čije je učenje regularizirano kako bi se osiguralo da latentni prostor ima poželjna svojstva za generiranje novih podataka. Kako bi uveli regularizaciju u latentni prostor, umjesto kodiranja ulaza u jednu točku latentnog prostora, varijacijski

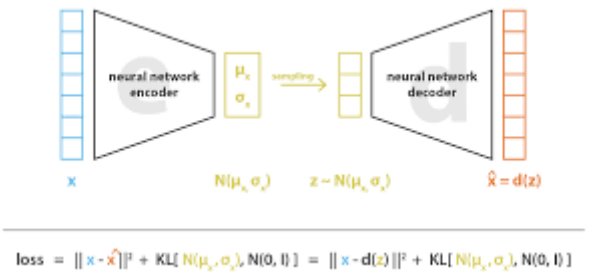
autenkoderi kodiraju ulaz u distribuciju nad latentnim prostorom, nakon toga se uzima jedan uzorak iz te distribucije i dekodira.



Slika 2.2: Razlika autoenkodera i varijacijskog autoenkodera je u latentnoj reprezentaciji

U praksi se najčešće kao distribucija latentnog prostora odabire normalna razdioba tako da se enkoder uči da kodira ulaz u vektor srednjih vrijednosti i vektor kovarijance. Razlog zbog kojeg kodiramo ulaz kao distribuciju je što želimo prisiliti model da generira distribuciju latentnog prostora što bliži normalnoj razdiobi te na taj način uvodimo lokalnu i globalnu regularizaciju latentnog prostora. Kod varijacijskih autoenkodera funkcija gubitka se ne računa samo kao gubitak rekonstrukcije jer sada želimo uvesti regularizacijski faktor koji će osigurati da dobivena distribucija enkodera je što bliža predefiniranoj distribuciji, najčešće normalnoj razdiobi. Regularizacijski faktor je uveden kao Kullback-Leibler divergencija između dobivene distribucije enkodera i normalne razdiobe. Novu funkciju gubitka možemo pisati kao:

$$loss = \|x - \hat{x}\|^2 + KL[N(\mu_x, \sigma_x), N(0,1)] \tag{2.4}$$

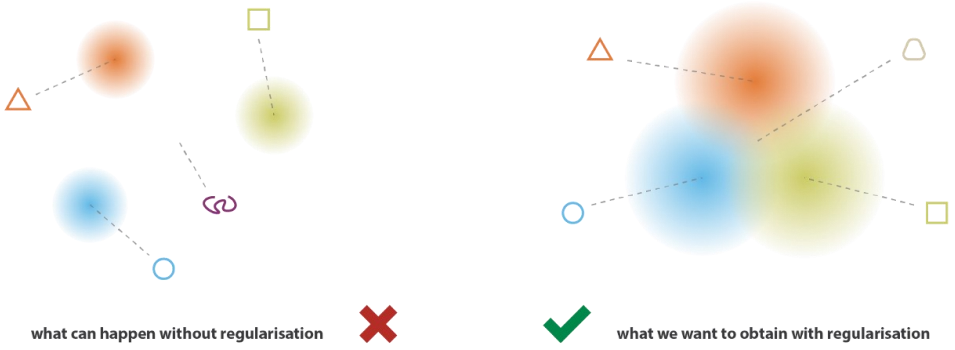


Slika 2.3: Varijacijski autoenkoder

Pravilnost koja se očekuje od latentnog prostora kako bi se omogućilo generiranje podataka može se izraziti kroz dva glavna svojstva:

- Kontinuitet – dvije bliske točke u latentnom prostoru ne bi trebale davati dva potpuno različita primjera nakon dekodiranja
- Cjelovitost – za odabranu distribuciju, točka uzorkovana iz latentnog prostora trebala bi dati smislen sadržaj nakon dekodiranja

Činjenica da varijacijski autoenkoder kodira ulaze kao distribucije u latentnom prostoru umjesto jednostavnih točaka nije dovoljna da osigura kontinuitet i cjelovitost. Bez definiranog regularizacijskog izraza, model može naučiti kako bi smanjio pogrešku rekonstrukcije ignorirajući činjenicu da je distribucija na izlazu enkodera i ponašajući se kao klasični autoenkoder. Kako bismo izbjegli ovo da se dogodi moramo regularizirati i matricu kovarijance i srednju vrijednost distribucija koje na izlazu daje enkoder. U praksi se ova regularizacija provodi prisiljavanjem distribucija da budu bliske standardnoj normalnoj distribuciji koja je centrirana i smanjena u volumenu.



Slika 2.4: Utjecaj regularizacije gubitka varijacijskog autoenkodera

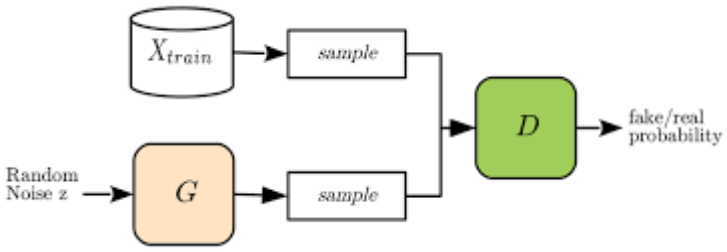
Regularizacijom sprječavamo model da kodira podatke daleko jedan od drugog u latentnom prostoru i potičemo preklapanja zadovoljavajući na taj način kontinuitet i potpunost.

2.2. Model zasnovan na suparničkom gubitku

U modelima zasnovanim na suparničkom gubitku generativni model suprotstavljen je protivniku, diskriminativnom modelu koji uči odrediti da li je uzorak iz distribucije generativnog modela ili distribucije podatka. Za razumjeti koncept na generativni model možemo gledati kao na tim krivotvoritelja koji pokušavaju proizvesti lažnu poznatu sliku i pokušati je prodati bez da ih otkriju, dok je diskriminirajući model policija koja pokušava otkriti krivotvorene slike. Natjecanje među sobom tjera oba tima da poboljšaju svoje metode do tog dijela kada se krivotvorene slike ne budu mogle razlikovati od originala. Generativni model u ovom slučaju obično kao ulaz prima latentnu varijablu koja se generira kao slučajni šum. Da bismo naučit distribuciju generatora $p_g(x)$, definiramo apriori na ulaznim varijablama šuma $p_z(z)$, zatim definiramo preslikavanje u podatkovni prostor kao $g_{\theta_g}(z)$, gdje je g diferencijabilna funkcija predstavljena višeslojnim perceptronom s parametrima θ_g. Također definiramo drugi višeslojni perceptron $d_{\theta_d}(x)$ koji preslikava iz podatkovnog prostora u jednu skalarnu vrijednost koja predstavlja vjerojatnost da x dolazi od distribucije ulaznih podataka, a ne od p_g. Cilj učenja diskriminativnog modela d je maksimiziranje vjerojatnosti dodjeljivanja točne procjene za stvarne podatke i za generirane podatke od generatora g. Paralelno tome učimo generator g da minimizira $\log(1 - d(g(z)))$, što znači da želimo da generator uspješno uspije zavarati diskriminator u pogrešnu procjenu. Drugim riječima, diskriminator i generator igraju minimax igru za dva igrača s funkcijom vrijednosti $V(g, d)$:

$$\min_g \max_d V(g,d) = E_{x\sim p_{data}(x)} log[d(x)] + E_{z\sim p_z(z)}[\log(1 - d(g(z)))] \quad (2.5)$$

U praksi, tu igru moramo implementirati pomoću iterativnog pristupa. Optimiziranje diskriminatora do kraja na ograničenom setu podataka bi rezultiralo u pretjeranom prilagođavanju na podatke i otežalo bi učenje generatora. Umjesto toga, obično se izmjenjuje par koraka optimiziranja diskriminatora sa jednim korakom optimiziranja generatora. To rezultira da se diskriminator drži blizu optimuma dok god se generator mijenja dovoljno sporo.



Slika 2.5: Model zasnovan na suparničkom gubitku

Generator implicitno definira vjerojatnosnu distribuciju p_g kao distribuciju uzoraka $g(z)$ dobivenih kada $z \sim p_z$. Kao cilj učenja želimo da algoritam konvergira u dobar procjenitelj stvarne distribucije podataka p_{data} ako mu damo dovoljno kapaciteta i vremena.

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**
  for k steps **do**
    • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
    • Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
    • Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for
  • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  • Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Slika 2.6: Učenje modela zasnovanih na suparničkom gubitku

Za neki dani generator, optimalni diskriminator računamo kao:

$$d_g^* = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)} \tag{2.6}$$

Dokaz. Kriterij učenja za diskriminator je maksimizacija $V(d,g)$:

$$V(d,g) = \int_x p_{data}(x)\log\big(d(x)\big)\,dx + \int_z p_z(z)\log\Big(1-d\big(g(z)\big)\Big)\,dz \qquad (2.7)$$
$$= \int_x p_{data}(x)\log\big(d(x)\big) + p_g(x)\log\big(1-d(x)\big)\,dx$$

Za bilo koji $(a,b)\ \varepsilon\ \mathbb{R}^2 \setminus \{0,0\}$, funkcija $y \to alog(y) + b(1-\log(y))$ dostiže svoj maksimum u $[0,1]$ u $\frac{a}{a+b}$.

Ako uvrstimo dobiveni optimum u minimax formulu dobivamo kriterij učenja:

$$C(g) = \max_d V(d,g) \qquad (2.8)$$
$$= E_{x\sim P_{data}}\big[log d_g^*(x)\big] + E_{z\sim p_z}\Big[\log\Big(1-d_g^*\big(g(z)\big)\Big)\Big]$$
$$= E_{x\sim P_{data}}\big[log d_g^*(x)\big] + E_{x\sim p_g}\Big[\log\Big(1-d_g^*(x)\Big)\Big]$$
$$= E_{x\sim p_{data}}\left[log\frac{p_{data}(x)}{p_{data}(x)+p_g(x)}\right] + E_{x\sim p_g}\left[log\frac{p_g(x)}{p_{data}(x)+p_g(x)}\right]$$

Ravnoteža je uspostavljena kada je distribucija generiranih podataka jednaka distribuciji stvarnih podataka. Globalni minimum kriterija učenja $C(g)$ je dostignut samo ako je uspostavljena ravnoteža $p_g = p_{data}$.

Dokaz. Za $p_g = p_{data}, d_g^*(x) = \frac{1}{2}$ dobivamo:

$$C(g) = log\frac{1}{2} + log\frac{1}{2} = -log4 \qquad (2.9)$$

Kako bismo vidjeli da je ovo najbolja vrijednost od $C(g)$, koja se postiže samo za $p_g = p_{data}$ možemo vidjeti da

$$E_{x\sim p_{data}}[-log2] + E_{x\sim p_g}[-log2] = -log4 \qquad (2.10)$$

Oduzimanjem ovog izraza od $C(g) = V\big(d_g^*, g\big)$ dobivamo:

$$C(g) = -log4 + KL(p_{data}\|\frac{p_{data}+p_g}{2}) + KL(p_{data}\|\frac{p_{data}+p_g}{2}) \qquad (2.11)$$

gdje KL predstavlja Kullback-Leibler divergence. U izrazu možemo prepoznati Jensen-Shannon divergenciju između distribucije stvarnih podataka i distribucije generiranih podataka generatora:

$$C(g) = -\log(4) + 2 * JSD(p_{data} \| p_g) \tag{2.12}$$

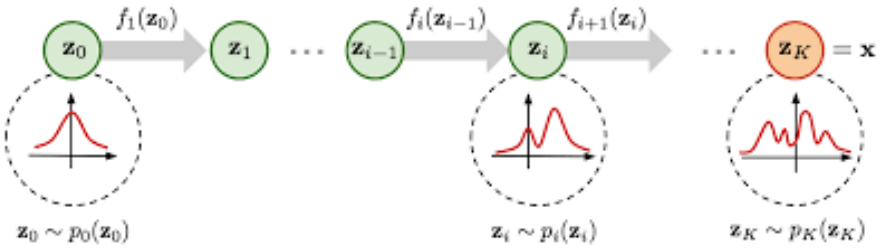
Obzirom da je Jensen-Shannon divergencija između dvije distribucije je uvijek pozitivna ili nula kada su distribucije jednake i iz tog možemo zaključiti da $C^* = -log4$ je globalni minimum od $C(g)$ i da je jedino rješenje $p_g = p_{data}$ kada generator idealno replicira proces generiranja podataka.

U praksi, modeli zasnovani na suparničkom gubitku predstavljaju ograničenu obitelj distribucija p_g preko funkcija g_θ i optimiziramo θ, a ne direktno distribuciju p_g. Koristeći višeslojni perceptron da definiramo generator uvodi više kritičnih područja u području parametara. Iako, dobre performanse višeslojnih perceptrona u praksi znači da su dobar izbor za generator unatoč nedostatku teoretskih garancija.

2.3. Model s normalizirajućim vjerojatnosnim tokom

Modeli s normalizirajućim vjerojatnosnim tokom su najnovija metoda kreiranja generativnih modela koja efektivno rješava problem učenja vjerojatnosne funkcije gustoće vjerojatnosti stvarnih podataka. Problem s generativnim modelima koji koriste latentne varijable kao ulaz je izračunavanje $p(x) = \int p(x|z)p(z)dz$ zbog toga jer praktički nije moguće obići sve moguće vrijednosti latentne varijable z. Modeli s normalizirajućim vjerojatnosnim tokovima rješavaju ovaj teški problem uz pomoć normalizirajućih tokova, moćnim alatom statistike za procjenu gustoće. Dobra procjena $p(x)$ omogućava nam da efikasno izvršavanje mnogih važnih zadataka kao što su generiranje novih, nikad ne viđenih, ali realističnih podataka, predviđanje vjerojatnosti budućih događaja, dobivanje latentnih varijabli, popunjavanje nepotpunih podataka i slično. Način na koji modeli s normalizirajućim vjerojatnosnim tokovima ovo postižu je tako da su konstruirani kao niz invertibilnih transformacija i za razliku od varijacijskih autoenkodera i modela zasnovanih na suparničkom gubitku ovaj model eksplicitno uči distribuciju podatka $p(x)$ i zbog toga je funkcija gubitka

jednostavno negativna log-vjerojatnost. Jedan od uvjeta da bi invertibilnost bila moguća bez gubitka informacija je da latentna varijabla bude jednake dimenzije kao dimenzionalnost podataka.



Slika 2.7: Normalizirajući tok

Prije nego krenemo detaljnije kako model funkcionira, opisat ćemo par važnih pojmova linearne algebre koji su u srži ovih modela: Jakobijan matrica i determinanta te teorem o zamjeni varijabli.

Ako postoji funkcija koja mapira n-dimenzionalni ulazni vektor x u m-dimenzionalni izlazni vektor:

$$f: \mathbb{R}^n \rightarrow \mathbb{R}^m \tag{2.13}$$

tada matrica svih parcijalnih derivacija prvog reda ove funkcije se naziva Jakobijan matrica J gdje vrijedi da vrijednost i-tog reda i j-tog stupca je jednaka $J_{ij} = \frac{\partial f_i}{\partial x_j}$.

$$\mathbf{J} = \begin{bmatrix} \dfrac{\partial \mathbf{f}}{\partial x_1} & \cdots & \dfrac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \nabla^{\mathrm{T}} f_1 \\ \vdots \\ \nabla^{\mathrm{T}} f_m \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f_1}{\partial x_1} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial f_m}{\partial x_1} & \cdots & \dfrac{\partial f_m}{\partial x_n} \end{bmatrix}$$

Slika 2.8: Računanje jakobijana funkcije

Determinanta matrice je jedan realan broj izračunat kao funkcija svih elemenata u kvadratnoj matrici. Važno svojstvo je da matrica bude kvadratna jer inače determinanta ne postoji. Na apsolutnu vrijednost determinante možemo gledati kao na

mjeru koja kaže u kolikoj mjeri će se prostor skupiti ili raširiti množenjem s tom matricom. Determinanta kvadratne matrice također nam daje informaciju o tome da li je matrica invertibilna ili ne. Ako je determinanta matrice jednaka nuli onda se smatra da matrica nije invertibilna, inače ako je determinanta različita od nule možemo reći da je matrica invertibilna. Determinanta umnoška matrica je jednaka umnošku determinanti matrica, $\det(AB) = \det(A)\det(B)$.

Sada ćemo objasniti teorem o promjeni varijable u kontekstu procjene gustoće vjerojatnosti, počevši od slučaja jedne varijable. Ako je dana neka slučajna varijabla z i njena poznata funkcija gustoće vjerojatnosti $z \sim \pi(z)$, želimo kreirati novi slučajnu varijablu preko 1 na 1 funkcijom mapiranja $x = f(z)$. Funkcija f je invertibilna što znači da vrijedi $z = f^{-1}(x)$. Sada se postavlja pitanje kako zaključiti nepoznatu funkciju gustoće vjerojatnosti $p(x)$ od nove varijable. Vrijedi:

$$\int p(x)dx = \int \pi(z)dz = 1 \tag{2.14}$$

$$p(x) = \pi(z)\left|\frac{dz}{dx}\right| = \pi\left(f^{-1}(x)\right)\left|\frac{df^{-1}}{dx}\right| = \pi(f^{-1}(x))|(f^{-1})'(x)| \tag{2.15}$$

Po definiciji, integral $\int \pi(z)dz$ je zbroj beskonačnog broja pravokutnika infinitezimalne širine Δz. Visina takvog pravokutnika na poziciji z je vrijednost funkcije gustoće $\pi(z)$. Kada zamijenimo varijablu, $z = f^{-1}(x)$ dobivamo:

$$\frac{\Delta z}{\Delta x} = (f^{-1}(x))' \tag{2.16}$$

$$\Delta z = (f^{-1}(x))'\Delta x \tag{2.17}$$

Možemo vidjeti da $|(f^{-1}(x))'|$ označava omjer između površina pravokutnika definiranih u dvije različite koordinate varijabli z i x.

Verzija sa više varijabli ima sličnu formulu:

$$z \sim \pi(z), x = f(z), z = f^{-1}(x) \tag{2.18}$$

$$p(x) = \pi(z)\left|det\frac{dz}{dx}\right| = \pi(f^{-1}(x))\left|det\frac{df^{-1}}{dx}\right| \tag{2.19}$$

gdje $det\frac{\partial f}{\partial z}$ predstavlja determinantu Jakobijana funkcije f.

Ukratko ćemo opisati 4 različita modela zasnovana na normalizirajućim vjerojatnosnim tokovima koji su povezani na način da je svaki idući nadogradnja na prijašnji sa boljim pokazanim rezultatima: NICE, RealNVP, GLOW i DenseFlow.

2.3.1. NICE

NICE ili punim imenom procjena nelinearnih nezavisnih komponenti je model koji pokušava naučiti transformaciju $h = f(x)$ podataka u novi prostor tako da rezultirajuća distribucija faktorizira, to jest da su komponente h_d nezavisne:

$$p_H(h) = \prod_d p_{H_d}(h_d) \tag{2.20}$$

Predloženi kriterij učenja izravno se izvodi iz log-vjerojatnosti. Točnije razmatramo promjenu varijabli $h = f(x)$, koja pretpostavlja da je funkcija f invertibilna i dimenzionalnost od h je jednaka dimenzionalnosti od x kako bi odgovarala distribuciji p_H. Pravilom promjene varijabli dobivamo:

$$p_X(x) = p_H(f(x)) \left| det \frac{\partial f(x)}{\partial x} \right| \tag{2.21}$$

gdje je $\frac{\partial f(x)}{\partial x}$ Jakobijan matrica funkcije f u x. U modelu odabiru funkciju f tako da determinanta Jakobijana se može jednostavno izračunati. Također, inverz funkcije je također jednostavno dobiti, što omogućuje jednostavno uzorkovanje iz $p_X(x)$:

$$h \sim p_H(h) \tag{2.22}$$

$$x = f^{-1}(h) \tag{2.23}$$

Najveći doprinos ovog modela je dizajn takvih transformacija f koje nam omogućuju dva važna svojstva: jednostavno izračunavanje determinante Jakobijana i jednostavno dobivanje inverza funkcije dok nam omogućuje da imamo što veći kapacitet kako bi naučili složene transformacije. Temeljna ideja iza ovoga je da možemo podijeliti x na dva dijela $(x1, x2)$ i primijeniti transformaciju u $(y1, y2)$ pomoću:

$$\begin{aligned} y1 &= x1 \\ y2 &= x2 + m(x1) \end{aligned} \tag{2.24}$$

gdje je m proizvoljno složena funkcija. Ovaj gradivni blok ima jediničnu Jakobijanovu determinantu za bilo koji m i jednostavno je dobiti inverz jer vrijedi:

$$x1 = y1$$
$$x2 = y2 - m(y1) \tag{2.25}$$

Razmatramo problem učenja gustoće vjerojatnosti iz parametarske obitelji gustoća nad ograničenim skupom podataka dimenzije D koji dolaze iz distribucije podataka χ. Model pokušava naučiti kontinuiranu, skoro svugdje diferencijabilnu nelinearnu transformaciju f distribucije podataka u jednostavniju distribuciju preko maksimalne vjerojatnosti korištenjem formule za promjenu varijabli:

$$\log\big(p_X(x)\big) = \log\Big(p_H\big(f(x)\big)\Big) + \log\left(det\,\frac{\partial f(x)}{\partial x}\right) \tag{2.26}$$

gdje $p_H(h)$ će biti predefinirana funkcija gustoće. Ako je funkcija s neovisnim dimenzijama, tada dobivamo NICE kriterij koji predstavlja najveću vjerojatnost podataka nad generativnim modelom kao determinističke transformacije distribucije s neovisnim dimenzijama:

$$\log\big(p_X(x)\big) = \sum_{d=1}^{D} \log\left(p_{H_d}(f_d(x))\right) + \log\left(det\,\frac{\partial f(x)}{\partial x}\right) \tag{2.27}$$

gdje $f(x) = (f_d(x))_{d \le D}$. Na NICE kriterij možemo gledati kao učenje invertibilne transformacije skupa podataka. Invertibilna transformacija može proizvoljno povećati vjerojatnost jednostavnim skupljanjem podataka na manji volumen. Koristimo kriterij zamjene varijabli da izbjegnemo ovaj fenomen i potičemo model da otkrije značajke strukture u skupu podataka. Pomoću tog kriterija determinanta Jakobijana transformacije f kažnjava skupljanje i potiče širenje u područjima visoke gustoće. Kao usporedbu sa varijacijskim autoenkoderom možemo gledati na funkciju f kao enkoder, dok na njen inverz f^{-1} gledamo kao na dekoder kojeg možemo koristiti za generiranje novih podataka iz distribucije.

Ako koristimo slojevitu ili složenu transformaciju $f = f_L \circ \ldots \circ f_2 \circ f_1$, transformacije prema naprijed i nazad sastavljene su od transformacija slojeva, a Jakobijanova determinanta umnožak je pripadnih Jakobijanovih determinanti slojeva. Kao izbor

osnovne transformacije afine transformacije sa triangularnim Jakobijanom su dobar izbor zbog svoje jednostavnosti izračuna determinante kao umnožak elemenata dijagonale i ujedno zbog jednostavnosti pronalaska inverza. Kao transformacije koje imaju ova poželjna svojstva izabiru se opći spojni slojevi. Ako je $x \sim \chi$ i I_1, I_2 su particije od x kada ga podjelimo u nekoj od dimenzija $[1, D]$ tako da vrijedi $d = |I_1|$ i m je funkcija definirana u $\mathbb{R}^d$, onda možemo definirati $y = (y_{I1}, y_{I2})$ gdje vrijedi:

$$\begin{aligned} y_{I1} &= x_{I1} \\ y_{I2} &= g(x_{I2}; m(x_{I1})) \end{aligned} \tag{2.28}$$

gdje $g : \mathbb{R}^{D-d} \, x \, m(\mathbb{R}^d) \rightarrow \mathbb{R}^{D-d}$ nazivamo zakon spajanja. Jakobijan ove funkcije računamo kao:

$$\frac{\partial y}{\partial x} = \begin{bmatrix} I_d & 0 \\ \dfrac{\partial y_{I2}}{\partial x_{I1}} & \dfrac{\partial y_{I2}}{\partial x_{I2}} \end{bmatrix} \tag{2.29}$$

gdje je I_d jedinična matrica dimenzije d što znači da $det \frac{\partial y}{\partial x} = \frac{\partial y_{I2}}{\partial x_{I2}}$.

Inverznu transformaciju računamo preko:

$$\begin{aligned} x_{I1} &= y_{I1} \\ x_{I2} &= g^{-1}(y_{I2}; m(y_{I1})) \end{aligned} \tag{2.30}$$

Kao primjer spojne funkcije pokazat ćemo kako funkcionira spojni sloj zbrajanjem gdje je $g(a; b) = a + b$:

$$\begin{aligned} y_{I2} &= x_{I2} + m(x_{I1}) \\ x_{I2} &= y_{I2} - m(y_{I1}) \end{aligned} \tag{2.31}$$

Determinanta Jakobijana ove transformacije će uvijek biti konstanta jedan. Možemo primijetiti da funkcija m ne zahtjeva svojstvo da treba postojati inverz i zbog toga za nju nemamo restrikcija pri izboru. Na primjer funkciju možemo predstavljati klasičnom neuronskom mrežom. Kao spojna funkcija umjesto zbrajanja možemo koristiti proizvoljni izraz uz uvjet da računanje determinante i inverza bude jednostavno. Možemo koristiti više spojnih slojeva u nizu kako bismo dobili složeniju slojevitu transformaciju. Obzirom da transformacija u sloju ostavlja dio ulaza nepromijenjen, moramo razmijeniti redoslijed particija u izmjeničnim slojevima tako

da sastav više spojnih slojeva modificira svaku dimenziju. Pokazalo se da je potrebno imati barem tri spojna sloja da bi svaka dimenzija podataka utjecala jedna na drugu.

2.3.2. RealNVP

Glavni doprinos RealNVP modela u odnosu na prijašnji model je u izboru spojne funkcije koja uvodi skaliranje kako bi bolje mogao naučiti distribuciju podataka. Prijašnji model je radio samo pomak podataka po dimenzijama bez skaliranja što možemo zaključiti jer Jakobijan transformacije zbrajanjem je uvijek bio konstantan i jednak jedan. U ovom modelu se kao spojne funkcije koriste afine transformacije gdje za spojnu funkciju vrijedi:

$$g(a; b) = a \circ b_1 + b_2 \tag{2.32}$$

gdje je b_2 ekvivalentan prijašnjoj funkciji pomaka iz spojne funkcije zbrajanjem, dok b_1 predstavlja funkciju koja će skalirati volumen podataka. $\circ$ predstavlja Hadamardov umnožak matrica što znači da se element na pojedinom indeksu matrice množi sa elementom sa istim indeksom u drugoj matrici. Kao izlaz predstavljene afine transformacije dobivamo:

$$\begin{aligned} y_{I1} &= x_{I1} \\ y_{I2} &= x_{I2} \circ \exp\big(s(x_{I1})\big) + t(x_{I1}) \end{aligned} \tag{2.33}$$

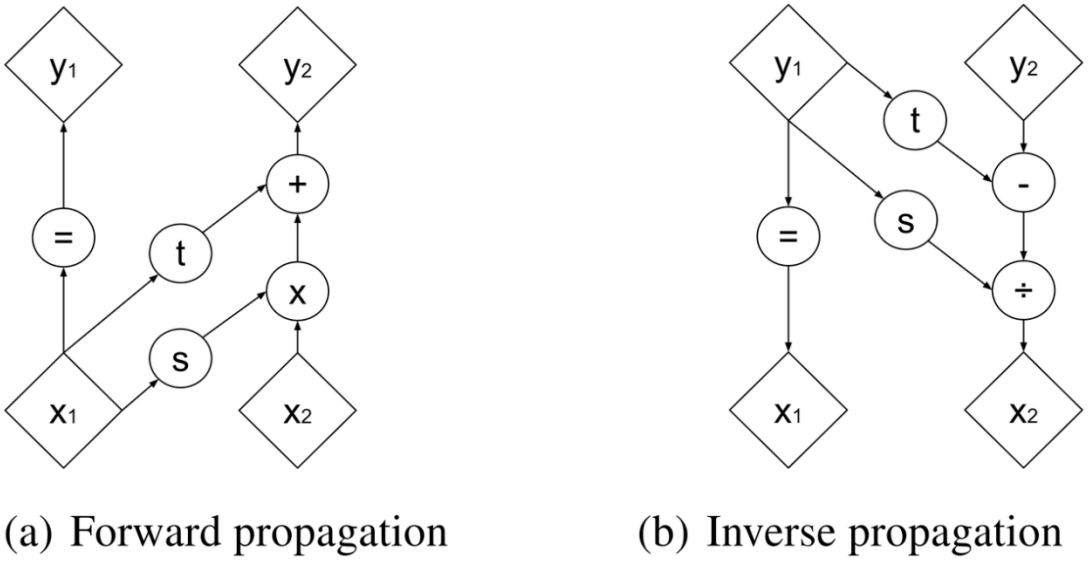
Jakobijan ovaj transformacije jednak je:

$$\frac{\partial y}{\partial x} = \begin{bmatrix} I_d & 0 \\ \dfrac{\partial y_{I2}}{\partial x_{I1}} & diag(\exp\,(s(x_{I1}))) \end{bmatrix} \tag{2.34}$$

Determinantu ovog Jakobijana predstavlja $\exp\,(\sum_j s(x_{I1})_j)$ što predstavlja faktor skaliranja. Inverz ove transformacija je jednak:

$$\begin{aligned} x_{I1} &= y_{I1} \\ x_{I2} &= (y_{I2} - t(y_{I1})) \circ \exp\,(-s(y_{I1})) \end{aligned} \tag{2.35}$$

Ponovno možemo vidjeti da računanje inverza od funkcija s i t nije nužan uvjet za njihovo kreiranja što znači da nemamo restrikcija kod njihovog modeliranja.

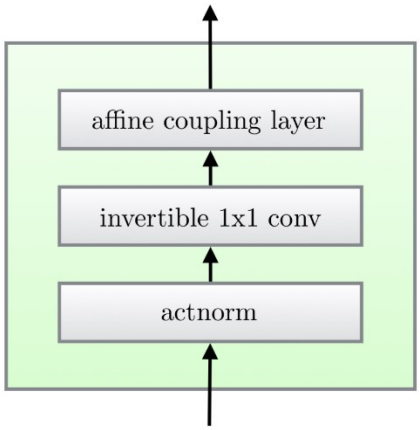


Slika 2.9: Afina transformacija skaliranjem i pomakom

2.3.3. GLOW

Glavni doprinos GLOW modela u odnosu na prijašnji je uvođenje tri operacije koje se izvode u svakom sloju, dok se prije sastojao samo od spojnog sloja i eventualno razmjena redoslijeda particija prije izvođenja sloja koja je sada u potpunosti izbačena i zamijenjena invertibilnom konvolucijom dimenzije 1x1 koja permutira podatke prije izvršavanja spojnog sloja. Novi predloženi redoslijed izvođenja operacija unutar jednog koraka transformacije je sljedeći:

1. Actnorm
2. Invertibilna 1x1 konvolucija (umjesto dodavanja promjene redoslijeda particija)
3. Spojni sloj



Slika 2.10: Korak transformacije od 3 invertibilne operacije

Actnorm ili normalizacija aktivacija izvodi afinu transformaciju aktivacija pomoću parametara skaliranja i pristranosti po svakom ulaznom kanalu slično kao kod batch normalizacije. Parametri su inicijalizirani tako da nakon aktivacije po kanalu imaju srednju vrijednost nula i jediničnu varijancu u odnosu na početni minibatch podataka. Nakon inicijalizacije, skaliranje i pristranost se tretiraju kao parametri koji se uče kroz iteracije i nisu ovisni o podacima. Actnorm transformaciju, njen inverz i log-determinantu možemo pisati kao:

$$y = s \circ x + b \tag{2.36}$$
$$x = \frac{y - b}{s}$$
$$\log\left(\frac{\partial y}{\partial x}\right) = h * w * sum(log|s|)$$

Problem kojim se dio ulaza ne mijenja se prije ovog modela rješavao tako da zamijenimo redoslijed kanala te tako osiguramo da se svaki ulaz transformira u jednom od slojeva. U ovom module se uvodi učena invertibilna 1x1 konvolucija. Konvolucija 1x1 s jednakim brojem ulaznih i izlaznih kanala generalizacija je operacije permutacije. Invertibilnu 1x1 konvoluciju, njen inverz i log-determinantu možemo pisati kao:

$$y = Wx$$
$$x = W^{-1}y \tag{2.37}$$
$$\log\left(\frac{\partial y}{\partial x}\right) = h * w * log|\det(W)|$$

2.3.4. DenseFlow

Glavni doprinosi ovog modela u odnosu na prijašnje je dodavanje inkrementalnog povećanja latentnih varijabli, odnosno međurezultata inverznih transformacija. Također predlaže se korištenje samo-opažanja unutar afinih spojnih slojeva. Standardna formulacija normalizirajućih tokova može se proširiti povećanjem ulaza varijablom šuma e_i. Taj šum je definiran nekom poznatom distribucijom $p^*_{e_i}$ koja na primjer može biti normalna distribucija. U ovom modelu se postupno ulančava šum na svaku latentnu varijablu z_i koja predstavlja među rezultat transformacija:

$$x \overset{f_1}{\leftrightarrow} z_1 \overset{f_2}{\leftrightarrow} \dots \overset{f_{n-1}}{\longleftrightarrow} z_{n-1} \overset{f_n}{\leftrightarrow} y \tag{2.38}$$

Formulacija ove ideje može se dobiti izračunavanjem donje granice vjerojatnosti $p(z_i)$ korištenjem Monte Carlo uzorkovanja e_i:

$$\ln p(z_i) \geq E_{e_i \sim p^*(e)}[\ln p(z_i, e_i) - \ln p^*(e_i)] \tag{2.39}$$

Naučena distribucija $p(z_i, e_i)$ aproksimira umnožak ciljnih distribucija $p^*(z_i)$ i $p^*(e_i)$. Transformira se uvedeni šum sa afinom transformacijom po svakom elementu. Parametri ove transformacije se računaju preko učene nelinearne transformacije $g_i(z_{<i})$ prijašnjih latentnih reprezentacija $z_{<i} = [z_0, z_1, \dots, z_{i-1}]$. Rezultirajuća funkcija se može definirati:

$$z_i^{(aug)} = h_i(z_i, e_i, z_{<i}) = [z_i, e_i \circ \sigma + \mu]$$
$$(\mu, \sigma) = g_i(z_{<i}) \tag{2.40}$$

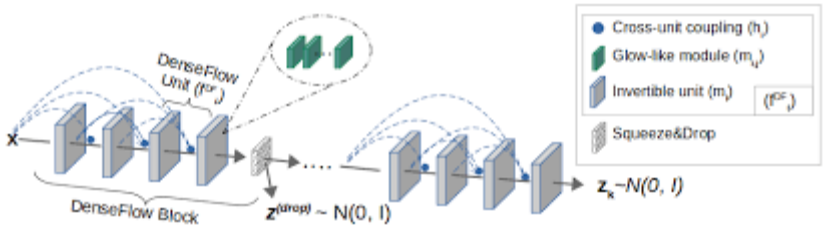
Jakobijan računamo preko formule:

$$\frac{\partial z_i^{(aug)}}{\partial [z_i, e_i]} = \begin{bmatrix} I & 0 \\ 0 & diag(\sigma) \end{bmatrix} \tag{2.41}$$

Iz toga možemo iskoristiti teorem promjene varijable:

$$\ln p(z_i, e_i) = \ln p\left(z_i^{(aug)}\right) + \ln|\det diag(\sigma)| \tag{2.42}$$
$$\ln p(z_i) \geq E_{e_i \sim p^*(e)}\left[\ln p\left(z_i^{(aug)}\right) - \ln p^*(e_i) + \ln|\det diag(\sigma)|\right]$$

Transformaciju h_i nazivamo unakrsni spojni sloj zato jer se ponaša kao spojni sloj nad prijašnjim invertibilnim slojevima. Latentni dio se propagira bez promjene dok se šum linearno transformira. Na linearnu transformaciju možemo gledati kao na reparametrizaciju distribucije iz koje se šum uzrokuje. Tijekom generiranja novih podataka važno nam je da možemo dobiti z_i iz $z_i^{(aug)}$, a to možemo učiniti uklanjanjem šuma.



Slika 2.11: DenseFlow model

3. Implementacija i rezultati

Na temeljnoj razini, ideja je naučiti generativni mode koji minimizira neki pojam divergencije obzirom na distribuciju podataka. Minimiziranje Kullback-Lieblerove razlike između distribucije podataka i modela na primjer ekvivalentno je izvođenju procjene maksimalne vjerojatnosti na promatranim podacima. Procjenitelji maksimalne vjerojatnosti asimptotski su statistički učinkoviti i služe kao prirodni ciljevi za učenje takvih generativnih modela. U ovu skupinu generativnih modela možemo svrstati prethodno opisane varijacijske autoenkodere i modele s normalizirajućim vjerojatnosnim tokovima. Iako ovi modeli imaju mnogo dobrih karakteristika, još uvijek u praksi bolji vizualni rezultati se postižu modelima temeljenima na suparničkom gubitku.

Nasuprot tome, alternativni način učenja generativnih modela temelji se na suparničkom gubitku gdje je cilj generirati podatke koji se ne razlikuju od podataka za učenje. Ovakvi modeli mogu zaobići određivanje eksplicitne gustoće za bilo koju točku podataka i pripadaju razredu implicitnih generativnih modela. U ovu skupinu generativnih modela možemo svrstati prethodno opisane modele zasnovane na suparničkom gubitku. Nedostatak karakterizacije eksplicitne gustoće problematičan je iz dva razloga. Neke primjene dubokih generativnih modela oslanja se na procjenu gustoće. Kao drugo, to čini kvantitativnu procjenu izvedbe generalizacije takvih modela izazovnom. Tipični kriterij ocjenjivanja kod ovakvih tipova modela nekada generiraju dobre uzorke memoriranjem podataka za učenje ili propuštanjem važnih načina distribucija.

Kako bi izbjegli ove nedostatke, u ovom radu pokušaj je koristiti model s normalizirajućim vjerojatnosnim tokom kao generator i regularizirati ga suparničkim gubitkom kako bi pokušali ostvariti kompromis između ova dva modela tako da dobijemo u određenoj razini dobre strane od obje vrste generativnih modela. Generator izgrađen kao model s normalizirajućim tokovima transformira prethodnu gustoću šuma u gustoću modela kroz niz invertibilnih transformacija. Korištenjem invertibilnog generatora, omogućeno nam je izračunati točnu vjerojatnosnu distribuciju korištenjem teorema za promjenu varijabli i izvođenje točnog posteriornog

zaključivanja nad latentnim varijablama, poželjna svojstva koja modeli sa suparničkim gubitkom obično no omogućuju, dok u isto vrijeme dobivamo bolje uzorkovanje regulariziranjem gubitka suparničkim gubitkom. Model zbog svojeg poželjnog svojstva invertibilnosti koje se koristi za generiranje podataka jednostavno se može koristiti kao generator za modele sa suparničkim gubitkom i upravo na taj način se uvodi regulariziranje suparničkim gubitkom. Uvodimo novi faktor $\alpha > 0$ koji regularizira koliki omjer gubitka uzimamo u obzir suparničkim gubitkom u odnosu na gubitak unakrsne entropije koji se koristi kod modela s normalizirajućim tokovima. Obzirom na taj omjer biramo omjer koji želimo za kvalitetu generiranih podataka vizualno u odnosu na log-vjerojatnost distribucije podataka.

U ovom radu izabrani model sa normalizirajućim tokom je DenseFlow zbog prednosti brzine učenja te postignutom kvalitetom rezultata. Kao model za diskriminator koji će se koristiti za regulariziranje suparničkim gubitkom korištene su dvije verzije: diskriminator od DCGAN te Resnet18 rezidualna mreža.

3.1. Korišteni skup podataka

Korišteni skup podataka za eksperimente u sklopu ovog rada je CIFAR-10. Ovaj skup podataka sastoji se od 60000 slika u boji 32x32 smještenih u deset kategorija sa 6000 slika po svakoj kategoriji. Skup podataka je podijeljen na 50000 slika za učenje i 10000 slika za testiranje. Kategorije međusobno nisu povezane.

Slika 3.1: CIFAR-10 skup podataka

3.2. Implementacija

Sva implementacija je nadodana na već postojeću implementaciju DenseFlow modela s normalizirajućim vjerojatnosnim tokovima koji će se ujedno koristiti kao model generatora za dodavanje regularizacije suparničkim gubitkom.

3.2.1. Implementacija diskriminatora

Koriste se dvije vrste diskriminatora u eksperimentima:

- DCGAN diskriminator

```
class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()
        self.main = nn.Sequential(
            # input is (nc) x 64 x 64
            nn.Conv2d(3, 32, 4, 2, 1, bias=False),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf) x 32 x 32
            nn.Conv2d(32, 64, 4, 2, 1, bias=False),
            nn.BatchNorm2d(64),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf*2) x 16 x 16
            nn.Conv2d(64, 128, 4, 2, 1, bias=False),
            nn.BatchNorm2d(128),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf*8) x 4 x 4
            nn.Conv2d(128, 1, 4, 1, 0, bias=False),
            nn.Sigmoid()
        )

    def forward(self, input):
        return self.main(input)
```

Slika 3.2: Implementacija DCGAN diskriminatora

- ResNet18 korišten kao diskriminator

```
class BasicBlock(nn.Module):
    expansion = 1

    def __init__(self, in_planes, planes, stride=1):
        super(BasicBlock, self).__init__()
        self.conv1 = nn.Conv2d(
            in_planes, planes, kernel_size=3, stride=stride, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(planes)
        self.conv2 = nn.Conv2d(planes, planes, kernel_size=3,
                               stride=1, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(planes)

        self.shortcut = nn.Sequential()
        if stride != 1 or in_planes != self.expansion*planes:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_planes, self.expansion*planes,
                          kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(self.expansion*planes)
            )

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        out += self.shortcut(x)
        out = F.relu(out)
        return out

class ResNet(nn.Module):
    def __init__(self, block, num_blocks, num_classes=1):
        super(ResNet, self).__init__()
        self.in_planes = 64

        self.conv1 = nn.Conv2d(3, 64, kernel_size=3,
                               stride=1, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(64)
        self.layer1 = self._make_layer(block, 64, num_blocks[0], stride=1)
        self.layer2 = self._make_layer(block, 128, num_blocks[1], stride=2)
        self.layer3 = self._make_layer(block, 256, num_blocks[2], stride=2)
        self.layer4 = self._make_layer(block, 512, num_blocks[3], stride=2)
        self.linear = nn.Linear(512*block.expansion, num_classes)
        self.sigm = nn.Sigmoid()

    def _make_layer(self, block, planes, num_blocks, stride):
        strides = [stride] + [1]*(num_blocks-1)
        layers = []
        for stride in strides:
            layers.append(block(self.in_planes, planes, stride))
            self.in_planes = planes * block.expansion
        return nn.Sequential(*layers)

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.layer1(out)
        out = self.layer2(out)
        out = self.layer3(out)
        out = self.layer4(out)
        out = F.avg_pool2d(out, 4)
        out = out.view(out.size(0), -1)
        out = self.linear(out)
        out = self.sigm(out)
        return out

def ResNet18():
    return ResNet(BasicBlock, [2, 2, 2, 2])
```

Slika 3.3: Implementacija ResNet-18 diskriminatora

3.2.2. Funkcija učenja

Učenje hibridnog modela kroz svaki batch podataka funkcionira na sljedeći način:

1. Računamo standardni gubitak za DenseFlow koristeći ulazni batch podataka

```
loss = elbo_bpd(self.model, x)
```

2. Generiramo novi podatak koristeći DenseFlow generator radeći inverznu propagaciju kroz model

```
fake_x = self.model.sample(batch).to(device=self.args.device, dtype=torch.float)
```

3. Računamo suparnički gubitak za generator

```
label = torch.full((batch,), self.real_label, dtype=torch.float, device=self.args.device)
output = self.model_discriminator(fake_x).view(-1)
errG = self.discriminator_criterion(output, label)
```

4. Računamo kompletni gubitak za generator u odnosu na izabrani faktor koji definira u kojem omjeru želimo regularizirati suparničkim gubitkom

```
G_complete_loss = self.args.alpha*loss+(1.0-self.args.alpha)*errG
```

5. Propagiramo pogrešku kroz generator i ažuriramo parametre

```
G_complete_loss.backward()
self.optimizer.step()
```

6. Radimo iteraciju učenja diskriminatora koja se provodi na standardan način

```
self.optimizer_discriminator.zero_grad()
output = self.model_discriminator(x).view(-1)
label.fill_(self.real_label)
label = label*(1.-self.args.label_smoothing)
errD_real = self.discriminator_criterion(output, label)
errD_real.backward()

label.fill_(self.fake_label)
label = label*(1.-self.args.label_smoothing/2)
output2 = self.model_discriminator(fake_x).view(-1)
errD_fake = self.discriminator_criterion(output2, label)
errD_fake.backward()

self.optimizer_discriminator.step()
```

3.3. Rezultati

Kao metrike koje pokazuju koliko dobro model uči određene karakteristike koje smatramo važnim koristi se BPD (*eng. bits per dimension*) te FID (*eng. Frechet Inception Distance*).

BPD je sličan negativnoj log vjerojatnosti, ali ovisi i o dimenziji ulaza tako da manje ovisi o rezoluciji ulazne slike. Što je vrijednost metrike niža, to znači da distribucija podataka je bliža distribuciji podataka za učenje.

FID je metrika koja se koristi za procjenu kvalitete slika stvorenih generativnim modelom. Za razliku od IS (*eng. Inception Score*) koji ocjenjuje samo distribuciju generiranih slika, FID uspoređuje distribuciju generiranih slika s distribucijom stvarnih slika koje su korištene za učenje generativnog modela. Što je vrijednost metrike niža, to znači da kvaliteta generiranih slika je bliža kvaliteti podataka za učenje.

Tablica 1: Rezultati učenja modela ovisno o regularizacijskom faktoru i vrsti diskriminatora

	Regularizacijski faktor	BPD	FID
-	0	3.05	44
DCGAN	0.5	3.35	41.5
ResNet18	0.5	3.3	40
ResNet18	1	6.85	31

Slika 3.4: Rezultati učenja modela sa regularizacijskim faktorom 0, 0.5 i 1 s lijeva na desno

4. Zaključak

Generiranje novih slika u računalnom vidu složen je problem sa raznim vrstama pristupa u rješavanju problema ovisno o rezultatu i mogućnostima koje modeli omogućuju. Modeli s normalizirajućim tokovima nam omogućuju računanje izglednosti podataka što nam je važno svojstvo u određenim primjenama. S druge strane modeli zasnovani na suparničkom gubitku pokazuju najbolje vizualne rezultate na generiranim slikama što je vrlo često glavni cilj u generiranju slika, ali s druge strane skloni su kolapsirati modove i nema mogućnosti da računamo izglednost podatka na jednostavan način.

U ovom radu pokušavamo dobiti kompromis između ove dvije vrste modela u pokušaju da dobijemo kvalitetu generiranih podataka blisku modelima zasnovanim na suparničkom gubitku dok u isto vrijeme omogućimo računanje izglednosti podataka. Jedan od uvjeta da bi to bilo moguće je da model generatora podataka bude invertibilan tako da možemo jednostavno računati izglednost podataka. Način na koji bismo dobili prednosti oba modela je tako da uvedemo parametar koji će regulirati utjecaj određenog gubitka koji nam donosi svojstva modela koja želimo dobiti. Na jednoj strani imamo gubitak BPD koji nam pokazuje koliko dobro distribucija generatora aproksimira distribuciju podataka, dok na drugoj strani imamo klasični gubitak modela zasnovanih na suparničkom gubitku za generator koji nam pokazuje koliko dobro generator uspijeva prevariti diskriminator u detektiranju generiranih podataka.

Možemo vidjeti da povećanjem omjera suparničkog gubitka dolazi do poboljšanja FID metrike, tj. dolazi do bolje vizualne kvalitete generiranih podataka, dok u isto vrijeme dolazi do povećanja BPD vrijednosti što znači da model generira podatke koji slabije aproksimiraju distribuciju podataka za učenje.

Literatura

[1] Ed Burns, *machine learning*. Poveznica: https://www.techtarget.com/searchenterpriseai/definition/machine-learning-ML

[2] Toma Petrač, *Polunadzirana semantička segmentacija utemeljena na pseudooznačavanju.* Diplomski rad. Sveučilište u Zagrebu Fakultet elektrotehnike i računarstva, 2021.

[3] Jay Salig, *What is Machine Learning? A Definition.* expert.ai (2022, ožujak). Poveznica: https://www.expert.ai/blog/machine-learning-definition/

[4] IBM Cloud Education, *Neural Networks*, IBM (2020, kolovoz). Poveznica: https://www.ibm.com/cloud/learn/neural-networks

[5] Jason Brownlee, *How to Choose an Activation Function for Deep Learning,* Machine Learning Mastery (2021, siječanj). Poveznica: https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/

[6] Ravindra Parmar, *Common Loss functions in machine learning*, Towards Data Science (2018, rujan). Poveznica: https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23

[7] Seb, *An Introduction to Neural Network Loss Functions*, Programmathically (2021, rujan). Poveznica: https://programmathically.com/an-introduction-to-neural-network-loss-functions/

[8] Aston Zhang, Zachary C. Lipton, Mu Li, Alexander J. Smola. *Dive Into Deep Learning*. Poveznica: https://d2l.ai/index.html

[9] Jason Brownlee, *Gentle Introduction to the Adam Optimization Algorithm for Deep Learning*, Machine Learning Mastery (2017, srpanj). Poveznica: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

[10] Sumit Saha, *A Coprehensive Guide to Convolutional Neural Networks – the ELI5 way*, Towards Dana Science (2018, prosinac). Poveznica: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[11] IBM Cloud Education, *Convolutional Neural Networks*, IBM (2020, listopad). Poveznica: https://www.ibm.com/cloud/learn/convolutional-neural-networks

[12] Vincent Feng, *An Overview of ResNet and its Variants,* Towards Data Science (2017, srpanj). Poveznica: https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035

[13] Lars Ruthotto, Eldad Haber. *An Introduction to Deep Generative Modeling,* CoRR, abs/2103.05180 (2021, travanj). Poveznica: https://arxiv.org/abs/2103.00265

[14] Joseph Rocca, *Understanding Variational Autoencoders (VAEs),* Towards Data Science (2019, rujan). Poveznica: https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

[15] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Benglo. *Generative Adversial Networks,* CoRR, abs/1406.2661 (2014, lipanj). Poveznica: https://arxiv.org/abs/1406.2661

[16] Lilian Weng, *Flow-based Deep Generative Models*, Lil'Log (2018, listopad). Poveznica: https://lilianweng.github.io/posts/2018-10-13-flow-models/

[17] Laurent Dinh, David Krueger, Yoshua Bengio. *NICE: Non-linear Independent Components Estimation*, CoRR, abs/1410.8516 (2015, travanj). Poveznica: https://arxiv.org/abs/1410.8516

[18] Laurent Dinh, Jascha Sohl-Dickstein, Samy Bengio. *Density estimation using Real NVP*, CoRR, abs/1605.08803 (2016, svibanj). Poveznica: https://arxiv.org/abs/1605.08803

[19] Diederik P. Kingma, Prafulla Dhariwal. *GLOW: Generative Flow with Invertible 1x1 Convolutions*, CoRR, abs/1807.03039 (2018, srpanj). Poveznica: https://arxiv.org/abs/1807.03039

[20] Matej Grcić, Ivan Grubišić, Siniša Šegvić. *Densely connected normalizing flows,* CoRR, abs/2106.04627 (2021, lipanj). Poveznica: https://arxiv.org/abs/2106.04627

[21] Aditya Grover, Manik Dhar, Stefano Ermon. *Flow-GAN: Combining Maximum Likelihood and Adversial Learning in Generative Models,* CoRR, abs/1705.08868 (2017, svibanj). Poveznica: https://arxiv.org/abs/1705.08868