SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1160

Algoritmi za brzo učenje na neprijateljskim primjerima

Dominik Jambrović

Zagreb, svibanj 2023.

Zahvaljujem svojoj obitelji radi podrške tijekom studiranja, kao i prof. dr. sc. Siniši Šegviću te mag. ing. Ivanu Grubišiću na pomoći tijekom izrade završnog rada.

SADRŽAJ

1. Uvod

Velik broj problema s kojima se danas susrećemo takve su prirode da ne znamo kako ih riješiti koristeći klasičan, algoritamski pristup. Razlog tome često leži u činjenici da ne znamo ni kako mi sami rješavamo te probleme, a jedan od najčešćih primjera za to je raspoznavanje tj. klasifikacija slika. Jedno od mogućih rješenja takvih problema je korištenje umjetnih neuronskih mreža - mreža sastavljenih od velikog broja povezanih jedinica (neurona) koje obavljaju veoma jednostavne operacije.

Razvojem dubokih neuronskih mreža došlo je do ubrzanog napretka u području računalnog vida. Računalni vid područje je umjetne inteligencije koje se bavi problemima poput klasifikacije 2D slika. Sve većom popularizacijom i korištenjem dubokih modela u sustavima različitih namjena, u pitanje se dovodi sigurnost takvih modela - ako model želimo koristiti u automobilima s ciljem detekcije pješaka i vozila, model mora moći dobro generalizirati, kao i biti robustan. Takav model ne bi smio mijenjati svoje odluke na temelju veoma malih promjena na ulazu - na primjeru prometa, želimo da model točno detektira pješaka, bez obzira nosi li on kapu ili ne.

Kako bi ostvarili robusnost modela, predložene su brojne tehnike, a jedna od najpopularnijih je robusno učenje tj. učenje na neprijateljskim primjerima. Kada su u pitanju modeli koji brzo uče, robusno učenje prihvatljivo je rješenje za postizanje robusnih modela otpornih na napade. Ipak, kada su u pitanju veći modeli za koje učenje traje veoma dugo, obično robusno učenje često je neprihvatljivo. U tu svrhu, razvijene su metode koje ubrzavaju robusno učenje. U ovome radu, razmatrat ćemo tri takve metode: besplatno robusno učenje [11], brzo robusno učenje [12] i nadogradnju na brzo robusno učenje (FastAdv+ i FastAdvW, [7]).

Uz to, razmatrat ćemo i otpornost naučenih modela na zatrovane podatke. Zatrovani podatci ulazi su izmijenjeni s ciljem navođenja modela na neočekivano ponašanje. Uvođenjem takve ranjivosti u model, napadači mogu neprimijećeno postići proizvoljne ciljeve poput izbjegavanja detekcije ili pogrešne klasifikacije.

2. Neuronske mreže

2.1. Općenito o neuronskim mrežama

Umjetne neuronske mreže veoma su popularan alat za rješavanje kompleksnih problema za koje je teško modelirati ili formalizirati znanje. Predstavnik su konektivističkog pristupa umjetnoj inteligenciji [1] koji se zasniva na oblikovanju sustava inspiriranih građom mozga.

Problemi koje rješavamo umjetnim neuronskim mrežama svrstavaju se u dvije glavne kategorije:

1. klasifikacija
2. regresija

Kada su u pitanju klasifikacijski problemi, cilj nam je svrstati ulaz u jedan od mogućih razreda. Pritom je na izlazu često korišteno jednojedinično kodiranje - ako ulaze svrstavamo u 10 razreda, u izlaznom sloju mreže bit će 10 neurona, a aktivacija jednog od njih predstavljat će rezultat klasifikacije.

S druge strane, kod regresijskih problema cilj nam je što bolje aproksimirati neku, modelu nepoznatu, funkciju. Za ovakve probleme, često nam je dovoljan jedan neuron u izlaznom sloju. Taj neuron na izlazu bi trebao davati predviđenu vrijednost funkcije za neki, do sada neviđeni, ulaz.

Da bi neuronska mreža mogla rješavati takve probleme, važno nam je da može učiti na temelju predočenih podataka. Učenje neuronske mreže odvija se izmjenom težina pojedinih neurona (time znanje implicitno ugrađujemo u našu mrežu). Kako bismo detaljnije mogli govoriti o učenju i arhitekturama neuronskih mreža, važno je ukratko opisati neuron - osnovnu gradivnu jedinicu svake mreže.

2.2. Umjetni neuron

Umjetni neuroni predstavljaju jednostavne procesne jedinice koje modeliraju ponašanje prirodnih neurona. Osnovni neuron akumulira vrijednosti na ulazu pomnožene težinama, akumuliranoj vrijednosti dodaje pomak te na kraju istu propušta kroz prijenosnu (aktivacijsku) funkciju. Ponašanje jednog neurona možemo modelirati jednadžbom:

$$o = f(\sum_{i=1}^{n} x_i \cdot w_i + b) \tag{2.1}$$

pri čemu x označava pojedine ulaze, w težine na pripadnim ulazima, b pomak te f prijenosnu funkciju.

2.3. Prijenosne funkcije

Neki od najranijih modela umjetnog neurona kao prijenosnu funkciju koristili su funkciju identiteta (ADELINE-neuron) te funkciju skoka (TLU-perceptron). S vremenom su korištene i razvijene brojne druge prijenosne funkcije poput sigmoidalne funkcije definirane kao:

$$sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

Sigmoidalna funkcija značajna je zbog svojstva derivabilnosti nad cijelom svojom domenom. Ovo svojstvo važno je za brojne optimizacijske postupke. Uz nju, danas je veoma značajna i zglobnica (ReLU - engl. *Rectified Linear Unit*) koju možemo prikazati na sljedeći način:

$$relu(x) = max(0, x) \tag{2.3}$$

Osim navedenih prijenosnih funkcija, korištena je i softmax funkcija koju jednadžbom možemo prikazati kao:

$$softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{2.4}$$

Softmax funkcija poopćenje je sigmoidalne funkcije, a često se koristi kao zadnja prijenosna funkcija u neuronskim mrežama korištenim za klasifikaciju. Korištenjem te funkcije u zadnjem sloju, na izlazu mreže imat ćemo vjerojatnosti klasifikacije u pojedini od razreda. Ovo svojstvo korisno nam je kada kao funkciju gubitka koristimo unakrsnu entropiju.

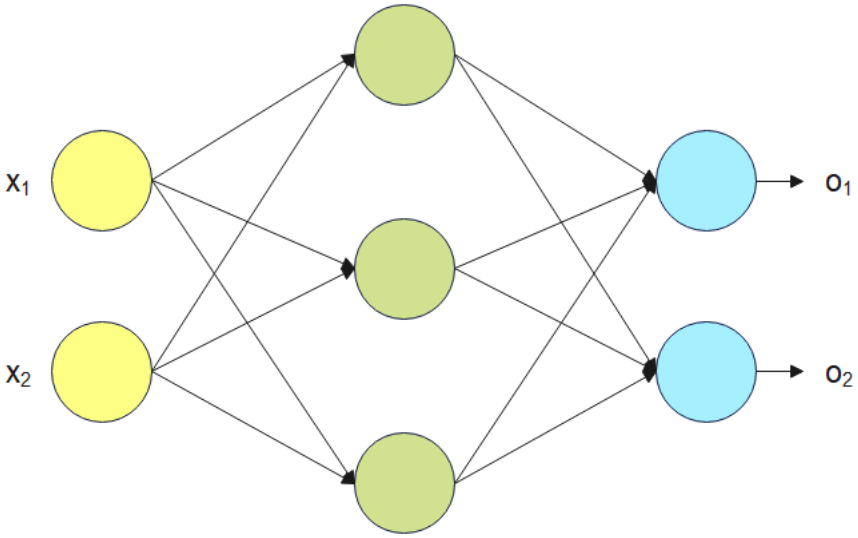
Važno je primijetiti da ako je prijenosna funkcija linearna, cijeli neuron može postići isključivo linearnu transformaciju. Kako bi umjetnim neuronima mogli modelirati kompleksnije funkcije, koristimo nelinearne prijenosne funkcije poput sigmoidalne funkcije i zglobnice. Pritom je za duboke neuronske mreže s velikim brojem slojeva često korištena upravo zglobnica - sigmoidalna funkcija za takve mreže nije prikladna zbog problema nestajućeg gradijenta (engl. *vanishing gradient problem*) koji nastaje tijekom učenja temeljenog na gradijentnim metodama.

2.4. Arhitektura umjetne neuronske mreže

Kada je za neki problem potrebno koristiti više od jednog osnovnog neurona, neurone povezujemo u mrežu. Pritom kažemo da se neuronska mreža sastoji od nekolicine slojeva:

1. ulazni sloj
2. skriveni slojevi
3. izlazni sloj

Iako ulazni sloj predstavljamo neuronima, oni, za razliku od neurona u skrivenim slojevima i izlaznom sloju, ne obavljaju nikakve transformacije - možemo reći da predstavljaju ulazni podatak. Veličina ulaznog sloja govori nam o dimenzijama ulaznih podataka, a veličina izlaznog sloja u slučaju problema klasifikacije često nam govori o broju razreda u koje klasificiramo ulaz.



Slika 2.1: 2x3x2 arhitektura umjetne neuronske mreže

Na slici 2.1 moguće je vidjeti primjer arhitekture umjetne neuronske mreže. Neuroni označeni žutom bojom predstavljaju ulazni sloj, neuroni označeni zelenom bojom skriveni sloj, a neuroni označeni plavom bojom izlazni sloj. Ovakvu arhitekturu mreže skraćeno možemo označiti kao 2x3x2 neuronsku mrežu. Pritom brojke označavaju broj neurona u pojedinom sloju (ulazni sloj je prvi sloj mreže).

Za ovakvu mrežu kažemo da je unaprijedna potpuno-povezana mreža. Pojam unaprijedna mreža označava to da ne postoje veze iz dubljih slojeva prema plićim slojevima, a pojam potpuno-povezana mreža označava to da svaki neuron ima vezu sa svakim neuronom iz prethodnog sloja. Uz to, svi neuroni imaju i dodatnu težinu zvanu pomak (nije prikazano na slici 2.1). Djelovanje jednog sloja mreže sažeto možemo prikazati kao:

$$\boldsymbol{h}_i = f(\boldsymbol{W}_i \cdot \boldsymbol{h}_{i-1} + \boldsymbol{b}_i) \tag{2.5}$$

pri čemu W_i predstavlja težine trenutnog sloja, b_i pomake trenutnog sloja, h_{i-1} izlaz iz prethodnog sloja, f prijenosnu funkciju primijenjenu na svaki element te h_i izlaz iz trenutnog sloja. Korištenjem takve formule za svaki sloj mreže, na kraju ćemo dobiti izlaz mreže za neki proizvoljni ulaz. Ovo nazivamo unaprijednim prolazom.

2.5. Učenje neuronske mreže

Kako bismo mogli koristiti proizvoljnu mrežu za probleme klasifikacije ili regresije, potrebno ju je prvo naučiti. Kao što je već prethodno rečeno, učenje neuronske mreže odgovara izmjeni težina pojedinih neurona, a najčešće se postiže algoritmom propagacije pogreške unatrag [3]. Da bismo mogli znati kako trebamo izmijeniti težine neurona, prvo trebamo znati koliko naš model griješi. Mjera greške naziva se gubitak, a računa se na temelju izlaza modela i očekivanog (točnog) izlaza. Za izračun gubitka često je korištena unakrsna entropija koju možemo definirati kao:

$$H(P^*|P) = -\sum_i P^*(i) \cdot \log P(i) \tag{2.6}$$

pri čemu $P^*(i)$ označava distribuciju očekivanog izlaza, a $P(i)$ distribuciju izlaza modela. Jednom kada znamo iznos gubitka, koristeći optimizatore poput stohastičkog gradijentnog spusta (SGD) ili Adam optimizatora [6] možemo poboljšati naš model. Pritom nam je za optimizacijski postupak veoma često potreban gradijent funkcije gubitka po parametrima modela, a izračunavamo ga uzastopnom primjenom pravila ulančavanja koje u svojem najjednostavnijem obliku možemo definirati kao:

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx} \tag{2.7}$$

U slučaju stohastičkog gradijentnog spusta, iterativno ažuriramo težine modela na temelju iznosa gradijenta funkcije gubitka, kao i stope učenja. Sažeto postupak stohastičkog gradijentnog spusta možemo prikazati jednadžbom:

$$w_{i+1} := w_i - \eta \cdot \nabla_w \cdot L(w_i, x) \tag{2.8}$$

pri čemu w_i označava jednu od težina modela u trenutnoj iteraciji, w_{i+1} istu težinu u sljedećoj iteraciji, $L(w_i, x)$ funkciju gubitka, a η stopu učenja. Stopa učenja mali je pozitivni broj pomoću kojeg možemo kontrolirati iznos promjene težina po iteracijama. Iterativnim ponavljanjem ovakvog postupka minimiziramo iznos funkcije gubitka, time dobivajući što bolju mrežu. Kod stohastičkog gradijentnog spusta, iteracije mogu biti predstavljene pojedinačnim ulazima ili mini-grupama. Zbog činjenice da SGD pohranjuje gradijent za male ulaze, ovaj optimizacijski postupak zahtijeva malo memorije.

Nakon učenja mreže na skupu za učenje, evaluirat ćemo performanse mreže na neviđenom skupu zvanom skup za testiranje (engl. *test set*). Često korištena mjera za kvalitetu modela je točnost definirana kao:

$$accuracy = \frac{correct}{total} \tag{2.9}$$

pri čemu $correct$ označava broj točno klasificiranih primjera, $total$ ukupan broj primjera, a $accuracy$ točnost. Uz točnost, postoje i brojne druge mjere kvalitete modela. Neke od njih su preciznost (engl. *precision*), odziv (engl. *recall*) i matrica zabune (engl. *confusion matrix*).

2.6. Duboke neuronske mreže

Ako želimo rješavati složenije probleme koristeći umjetne neuronske mreže sa samo jednim skrivenim slojem, suočit ćemo se s problemom - broj neurona potreban kako bi umjetna neuronska mreža mogla obavljati svoju zadaću bit će prevelik. Uz to, korištenjem širokog modela s velikim brojem neurona u skrivenom sloju teško ćemo postići svojstvo generalizacije jer će se model lako prenaučiti i zapamtiti ulazne podatke.

Zbog tih razloga, veoma su popularne duboke neuronske mreže [3]. Duboke neuronske mreže, za razliku od mreža sa samo jednim skrivenim slojem, imaju nekolicinu skrivenih slojeva. Pritom za rješavanje složenijih problema duboke mreže trebaju imati značajno manje neurona po sloju naspram mreže sa samo jednim skrivenim slojem. Zasebni slojevi mreže naučit će prepoznavati zasebne značajke ulaza, a njihovom kombinacijom mreža će moći postići uspješnu klasifikaciju.

Ipak, postoje i određene mane dubokih neuronskih mreža. Jedna od mana činjenica je da je za propagaciju pogreške unazad kod ovakvih mreža potrebno množiti gradijente. U slučaju da kao prijenosnu funkciju koristimo sigmoidalnu funkciju, ovo lako vodi do problema nestajućeg gradijenata zbog kojega težine neurona u plitkim slojevima nećemo moći ispravno izmijeniti. Uz problem nestajućeg gradijenta, postoji i problem eksplodirajućeg gradijenta (engl. *exploding gradient problem*) koji se pojavljuje kod nekih drugih prijenosnih funkcija od kojih je najpoznatija upravo zglobnica (ReLU). Još jedna mana dubokih neuronskih mreža činjenica je da kako bismo kvalitetno naučili duboku mrežu moramo imati veoma velik skup podataka.

2.7. Konvolucijske mreže

Za probleme s velikim dimenzijama ulaza, potpuno-povezane mreže imaju izuzetno velik broj parametara tj. težina neurona. Učenje ovakvih modela zahtijeva veliku količinu memorije, a i općenito je sporo. Uz to, potpuno-povezane mreže osjetljive su na translaciju ulaza: ako učimo mrežu da klasificira slike vozila te nakon učenja mreži predočimo translatiranu sliku iz skupa za učenje, mreža tu sliku neće nužno moći ispravno klasificirati jer je za nju to potpuno novi podatak. Ovo svojstvo proizlazi iz činjenice da je svaki neuron ovisan o svakom neuronu iz prethodnog sloja.

Kako bi doskočili ovim problemima, uvedene su konvolucijske mreže [9]. Za razliku od potpuno-povezanih mreža, ovdje aktivacija neurona ne ovisi o svim neuronima iz prethodnog sloja, već samo o malom rasponu neurona iz prethodnog sloja. Time konvolucijska mreža ima značajno manje parametara naspram potpuno-povezane mreže iste dubine, a postiže i svojstvo translacijske invarijantnosti. Ova svojstva konvolucijska mreža postiže zamjenom standardnog množenja matrica operacijom konvolucije s jezgrom (engl. *kernel*). Općenito govoreći, operaciju konvolucije za dvije funkcije f, g možemo definirati kao:

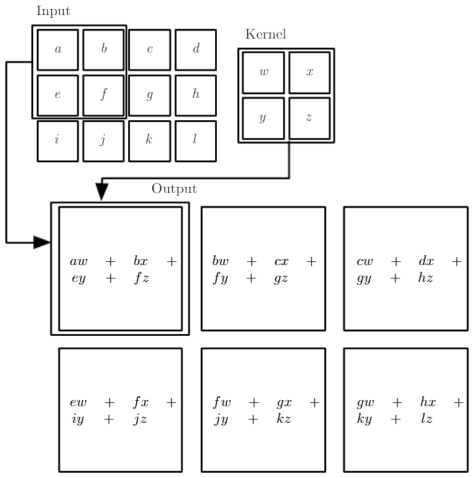
$$(f * g)(t) := \int_{-\infty}^{\infty} f(\tau) \cdot g(t - \tau) d\tau \tag{2.10}$$

Operacija konvolucije opisuje nam kako se izgled jedne funkcije mijenja pod utjecajem druge funkcije, a definirana je kao integral umnoška funkcija nakon što je jedna reflektirana i translatirana. Uz operaciju konvolucije postoji i unakrsna korelacija definirana kao:

$$(f \star g)(t) := \int_{-\infty}^{\infty} f(\tau) \cdot g(t + \tau) d\tau \tag{2.11}$$

Važno je primijetiti da je glavna razlika između te dvije operacije izostajanje reflektiranja jedne od funkcija u slučaju operacije unakrsne korelacije. Kada kod konvolucijskih mreža govorimo o konvoluciji, gotovo uvijek se zapravo misli na unakrsnu korelaciju.

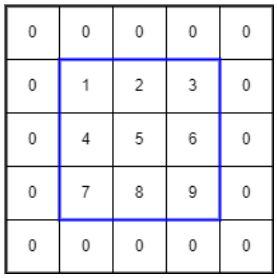
Kako bismo što jednostavnije objasnili konvoluciju, koristit ćemo primjer s 2D konvolucijom. U tom slučaju, jezgra s kojom se provodi konvolucija mala je kvadratna matrica s težinama koje učenjem izmjenjujemo. Skalarnim produktom dijelova ulazne matrice i jezgre dobivamo izlaz konvolucijskog sloja. Pritom se jezgra pomiče po ulaznoj matrici, a rezultati skalarnog produkta zapisuju se u novu matricu koju zovemo mapa značajki.



Slika 2.2: Primjer 2D konvolucije. Preuzeto iz [3]

Na slici 2.2 možemo vidjeti rezultat 2D konvolucije s ulaznom matricom veličine 3x4 i jezgrom dimenzija 2x2. Mapa značajki nastala kao rezultat ove konvolucije dimenzija je 2x3. Primijetimo da će izlaz konvolucijskog sloja uvijek biti manjih dimenzija naspram ulaza. Uz to, vrijednosti na rubovima matrice ulaza manje će doprinijeti rezultatu naspram vrijednosti koje su dalje od rubova.

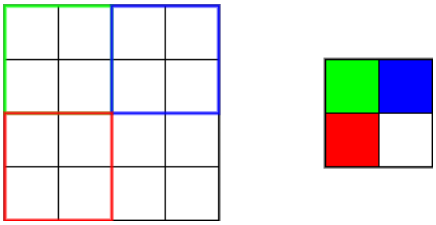
Kako bismo imali veću kontrolu nad dimenzijama izlaza, kao i utjecajem vrijednosti na rubovima matrice ulaza, često koristimo nadopunjavanje nulama (engl. *zero padding*).



Slika 2.3: Primjer nadopunjavanja 3x3 matrice nulama

Na slici 2.3 moguće je vidjeti 3x3 matricu nadopunjenu nulama. U slučaju da nad takvom matricom primijenimo konvoluciju s jezgrom dimenzija 2x2, mapa značajki na izlazu bila bi dimenzija 4x4. Da smo konvoluciju primijenili nad matricom bez nadopunjavanja, mapa značajki bila bi dimenzija 2x2, a vrijednosti pri rubovima matrice manje bi doprinosile istoj. Možemo reći da nule čine okvir oko originalne matrice, time osiguravajući veće dimenzije izlaza.

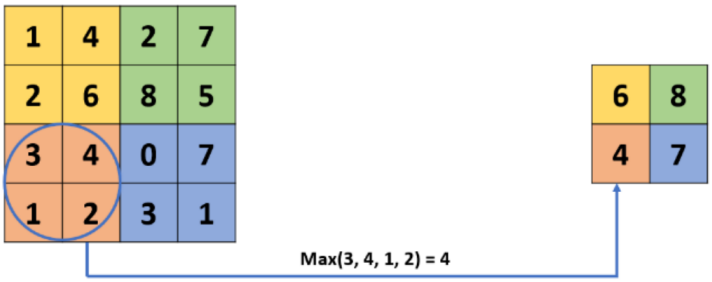
Uz nadopunjavanje nulama, kod konvolucijskih slojeva često se mijenja i korak (engl. *stride*). Na slici 2.2 korak je 1, a definira za koliko se ćelija horizontalno i vertikalno pomiče jezgra. U slučaju da povećamo korak, izlaz konvolucije bio bi manjih dimenzija, a time bi se i ubrzao postupak računanja izlaza.



Slika 2.4: Primjer konvolucije s korakom 2

Na slici 2.4 moguće je vidjeti konvoluciju s korakom 2. Za razliku od standardnog kretanja jezgre, ovdje se jezgra nakon svakog izračuna pomiče za 2 ćelije.

Osim samih konvolucijskih slojeva, konvolucijske mreže u sebi sadrže i slojeve sažimanja. Najčešći razlog za upotrebu slojeva sažimanja je smanjivanje dimenzija podataka, a time i skraćivanje vremena potrebnog za učenje mreže.

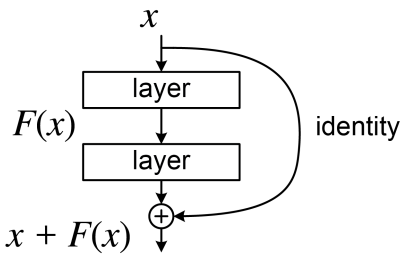


Slika 2.5: Primjer sažimanja maksimalnom vrijednošću. Preuzeto iz [2]

Na slici 2.5 moguće je vidjeti primjer korištenja sloja sažimanja maksimalnom vrijednošću. Mapa značajki dimenzija 4x4 korištenjem sažimanja maksimalnom vrijednošću 2x2 (engl. *2x2 max pooling*) smanjena je na dimenzije 2x2, efektivno smanjujući dimenzije ulaza za faktor 2. Uz sažimanje maksimalnom vrijednošću, veoma je popularno i sažimanje usrednjavanjem, no postoje i brojne druge varijante koje se koriste za slojeve sažimanja. Kao i operacija konvolucije, i slojevi sažimanja doprinose translacijskoj invarijantnosti, osiguravajući da mreža lako može prepoznati neku značajku bez obzira na njenu točnu lokaciju.

2.8. Rezidualne mreže

Rezidualne mreže [5] vrsta su dubokih neuronskih mreža koje koriste rezidualne blokove. Općenito govoreći, blok u kontekstu neuronskih mreža označava niz od nekoliko slojeva. Pojedini blokovi mogu se kombinirati kako bi sačinjavali složeniju mrežu. Rezidualni blokovi najčešće se sastoje od nekoliko konvolucijskih slojeva, a njihova glavna karakteristika postojanje je preskočnih veza.



Slika 2.6: Primjer rezidualnog bloka. Preuzeto iz [5]

Na slici 2.6 možemo vidjeti rezidualni blok sačinjen od dva sloja. Preskočna veza ulaz u blok bez ikakvih transformacija prenosi na izlaz. Ovakvim strukturiranjem mreže, cilj mreže postaje modelirati rezidualnu funkciju $F(x)$ koja mjeri razliku izlaza naspram ulaza. Rezidualni blok možemo prikazati jednadžbom:

$$f(\boldsymbol{x}) = F(\boldsymbol{x}) + \boldsymbol{x} \tag{2.12}$$

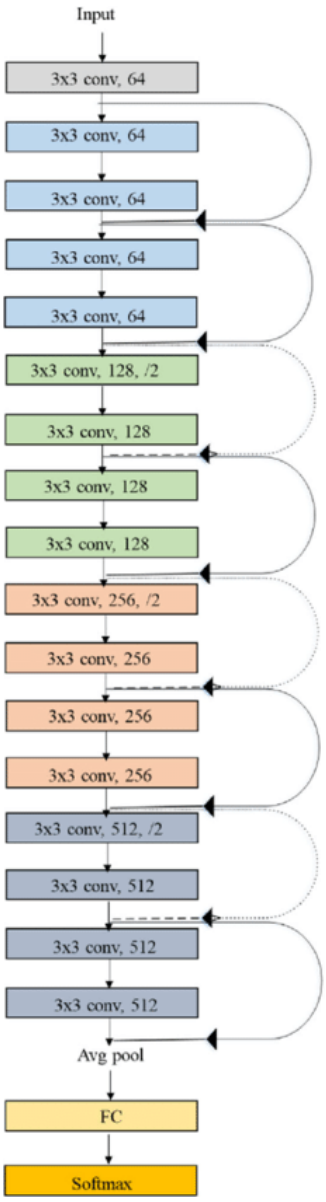
pri čemu $F(x)$ predstavlja rezidualnu funkciju, x prenošenje ulaza na izlaz preskočnom vezom, a $f(x)$ izlaz bloka. U slučaju da slojevi između mijenjaju dimenzije podataka, preskočna veza morat će raditi linearnu projekciju kako bi podatci pri zbrajanju bili istih dimenzija. Tada rezidualni blok možemo prikazati na sljedeći način:

$$f(\boldsymbol{x}) = F(\boldsymbol{x}) + \boldsymbol{W}_s \cdot \boldsymbol{x} \tag{2.13}$$

pri čemu W_s označava matricu korištenu za linearnu projekciju ulaza x. Korištenje slojeva s preskočnim vezama motivirano je željom za učenjem funkcije identiteta. Klasične duboke mreže s velikim brojem slojeva teško će naučiti funkciju identiteta, dok korištenje preskočnih veza dubokim mrežama značajno olakšava učenje te funkcije. Ovo potječe od činjenice da je za uspješno modeliranje funkcije identiteta $f(x)$ za rezidualnu funkciju $F(x)$ potrebno samo postaviti težine jezgri na 0.

2.9. ResNet18

U okviru ovog rada, za provođenje svih eksperimenata koristit ćemo ResNet18 mrežu kako bismo što bolje mogli usporediti učinak različitih eksperimenata. ResNet18 rezidualna je neuronska mreža, a sastoji se od ukupno 18 slojeva.



Slika 2.7: Arhitektura ResNet18. Preuzeto iz [10]

Kao što možemo vidjeti na slici 2.7, ResNet18 sastoji se od 8 rezidualnih blokova. Svaki od tih blokova sastoji se od 2 konvolucijska sloja i jedne preskočne veze. Rezidualnim blokovima prethodi jedan konvolucijski sloj, a nakon njih dolazi jedan potpuno-povezani sloj sa softmax prijenosnom (aktivacijskom) funkcijom.

3. Brzo učenje s neprijateljskim primjerima

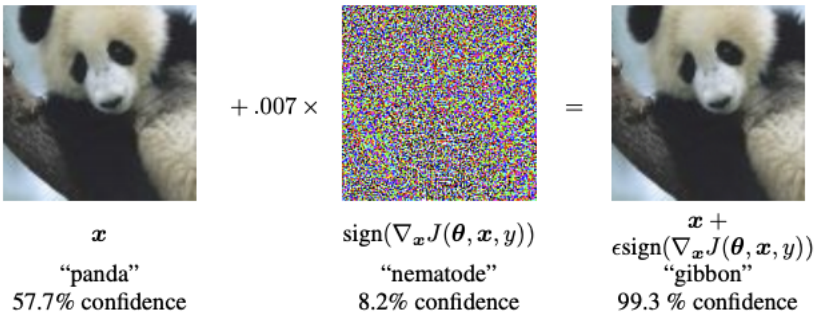
3.1. Neprijateljski primjeri

Kada učimo modele da rješavaju razne probleme, cilj nam je da modeli pokažu svojstvo generalizacije. U slučaju problema klasifikacije, to znači da bi modeli trebali dobro klasificirati i do sada neviđene primjere. Mogućnost generalizacije određenog modela najčešće provjeravamo koristeći unakrsnu provjeru (engl. *cross-validation*). Pritom skup podataka dijelimo na 2 ili 3 skupa:

1. skup za učenje (engl. *train set*)
2. skup za provjeru (engl. *validation set*)
3. skup za testiranje (engl. *test set*)

Skup za učenje modelu predočavamo kako bi na temelju njega mogao poboljšavati svoje parametre tj. učiti. S druge strane, skup za provjeru koristimo za poboljšavanje hiperparametara modela (npr. stopa učenja modela). U slučaju da smo sigurni u dobar odabir hiperparametara modela, korištenje ovog skupa nije nužno. Nakon postupka učenja, koristeći skup za testiranje evaluiramo model.

Općenito govoreći, model će na skupu za testiranje postizati lošije rezultate naspram rezultata na skupu za učenje. Ako model dobro generalizira, razlika između rezultata na ta dva skupa neće biti veoma značajna. Ipak, u zadnjih desetak godina otkriveni su ulazi koje i modeli koji inače pokazuju mogućnost generalizacije loše klasificiraju. Takve ulaze zvat ćemo neprijateljski primjeri [4].



Slika 3.1: Primjer neprijateljskog primjera. Preuzeto iz [4]

Na slici 3.1 moguće je vidjeti jedan neprijateljski primjer. Originalni ulaz kojeg model ispravno klasificira izmijenjen je za iznos perturbacije. U ovom slučaju iznos perturbacije dobiven je kao umnožak malog pozitivnog koeficijenta ϵ i predznaka gradijenta gubitka po ulazu. Iako je izmijenjeni ulaz ljudima gotovo identičan, model ovaj ulaz krivo klasificira, a pritom je veoma siguran u svoju odluku. Ovo svojstvo iskazali su brojni modeli te je zbog toga u zadnjih desetak godina porastao interes za proučavanjem neprijateljskih primjera.

3.2. Načini generiranja neprijateljskih primjera

Neprijateljske primjere moguće je generirati na različite načine. Načine generiranja neprijateljskih primjera zovemo napadima, a možemo ih podijeliti na usmjerene i neusmjerene napade. Usmjereni napadi kao cilj imaju generiranje neprijateljskih primjera koje će napadnuti model klasificirati u točno jedan ciljni razred, dok neusmjereni napadi kao cilj imaju generiranje neprijateljskih primjera koje će napadnuti model što lošije klasificirati, neovisno o konkretnom razredu. Pronalaženje neprijateljskih primjera zapravo je optimizacijski problem koji možemo prikazati sljedećim izrazom:

$$\underset{\delta \in [-\epsilon, \epsilon]}{max} L(x + \delta, y; \theta) \tag{3.1}$$

pri čemu δ označava iznos perturbacije u intervalu određenim s granicama $[-\epsilon, \epsilon]$, $L(x, y; \theta)$ funkciju gubitka, a θ parametre modela. Drugim riječima, cilj nam je naći perturbaciju omeđenu određenim iznosom takvu da je iznos funkcije gubitka za ulaz izmijenjen tom perturbacijom maksimalan. Iznos perturbacije najčešće je omeđen L_∞-normom, no isti može biti omeđen i L_2-normom ili L_1-normom.

Postoje brojni načini generiranja neprijateljskih primjera. Neki od najpoznatijih su FGSM (engl. *Fast Gradient Sign Method*) [4] i PGD (engl. *Projected Gradient Descent*) [8] napadi. Primjer sa slike 3.1 zapravo je neprijateljski primjer generiran koristeći FGSM napad, a neprijateljske primjere generirane koristeći taj napad sažeto možemo opisati jednadžbom:

$$x := x + \epsilon \cdot sign(\nabla_x L(x, y; \theta)) \tag{3.2}$$

pri čemu ϵ predstavlja mali pozitivan broj koji određuje veličinu perturbacije, a $sign(x)$ predznak od x. FGSM napad originalnom ulazu pridodaje iznos istog predznaka kao i gradijent funkcije gubitka, time osiguravajući najveći mogući rast iznosa funkcije gubitka. U slučaju neusmjerenog napada, napad bi pokušavao što više smanjiti iznos funkcije gubitka s obzirom na proizvoljan razred.

Uz FGSM napad, postoji i napad koji iterativno stvara značajno učinkovitije neprijateljske primjere. Tu vrstu napada zovemo PGD napad, a zasniva se na istoj ideji kao i FGSM napad. Za razliku od FGSM-a, PGD napad iterativno ponavlja postupak, time pokušavajući odrediti najbolje rješenje optimizacijskog problema 3.1. Uz to, PGD napad tijekom generiranja neprijateljskih primjera osigurava da ukupna perturbacija nije veća od unaprijed definiranog koeficijenta ϵ.

Algorithm 1 Generiranje neprijateljskih primjera koristeći PGD napad

Ulaz: x – ulazne slike, ϵ – ograničenje perturbacije, α – veličina koraka po iteraciji, K – broj iteracija za generiranje neprijateljskih primjera

Izlaz: x_{adv} – neprijateljski primjeri generirani PGD napadom

---

$\delta = U(-\epsilon, \epsilon)$

for $(i = 1 \dots K)$ **do**

  $grad = \nabla_\delta L(x + \delta, y; \theta)$

  $\delta = \delta + \alpha \cdot sign(g)$

  $\delta = clamp(\delta, -\epsilon, \epsilon)$

end for

$x_{adv} = x + \delta$

return x_{adv}

Pseudokod 1 prikazuje način generiranja neprijateljskih primjera koristeći PGD napad. Perturbacija δ na početku je inicijalizirana koristeći uniformnu distribuciju s rasponom $[-\epsilon, \epsilon]$, a na kraju svake iteracije iznos perturbacije ograničava se koristeći funkciju *clamp*. Naravno, postoje brojni načini inicijalizacije iznosa perturbacije, no nasumična inicijalizacija pokazala se dosta uspješnom. Važno je istaknuti da pseudokod za ograničavanje iznosa perturbacije koristi L_∞-normu. PGD napad s K iteracija za generiranje neprijateljskih primjera zvat ćemo K-PGD (npr. 20-PGD).

3.3. Učenje s neprijateljskim primjerima

Ako želimo da modeli koje učimo budu otporni na neprijateljske primjere tj. ispravno ih klasificiraju, ne možemo se ograničiti na klasično (prirodno) učenje modela. Svojstvo otpornosti na neprijateljske primjere zvat ćemo robusnost, a jedan od mogućih načina za postizanje robusnih modela učenje je s neprijateljskim primjerima (robusno učenje). Kada govorimo o učenju s neprijateljskim primjerima, općenita ideja je da se tijekom učenja generiraju neprijateljski primjeri prilagođeni samome modelu. Model tada ne uči na originalnim ulaznim podacima, već isključivo na generiranim neprijateljskim primjerima. U tom slučaju neprijateljske primjere možemo smatrati načinom augmentacije ulaznih podataka - na temelju originalnih ulaznih podataka stvaramo nove podatke koje predočavamo modelu.

Robusnim učenjem možemo postići puno višu točnost modela na skupu za testiranje sačinjenom od neprijateljskih primjera naspram modela učenih prirodnim učenjem. Nažalost, robusno učenje ima svoju cijenu. Robusni modeli gotovo uvijek će na originalnom skupu za testiranje imati nižu točnost naspram modela učenih prirodnim učenjem, a i učit će se dulje. Ako za generiranje neprijateljskih primjera tijekom robusnog učenja koristimo FGSM metodu, učenje će trajati otprilike dvostruko dulje naspram prirodnog učenja jer je za generiranje neprijateljskih primjera FGSM metodom potreban dodatan unaprijedni prolaz kroz mrežu. Ako za generiranje neprijateljskih primjera koristimo PGD metodu, duljina trajanja učenja ovisit će o broju iteracija korištenih za generiranje neprijateljskih primjera. Koristimo li 10 iteracija za generiranje neprijateljskih primjera, učenje će trajati otprilike 11 puta dulje naspram prirodnog učenja. Općenito govoreći, kako bismo model učili koristeći neprijateljske primjere generirane K-PGD metodom, učenje će trajati otprilike K + 1 puta dulje naspram prirodnog učenja [11].

Algorithm 2 Učenje s neprijateljskim primjerima

Ulaz: x – ulazne slike, N – broj epoha, M – veličina skupa za učenje, η – stopa učenja, ϵ – ograničenje perturbacije, α – veličina koraka po iteraciji, K – broj iteracija za generiranje neprijateljskih primjera

---

$\theta = initializeModelParams()$
for $(ep = 1 \dots N)$ **do**
  for $(i = 1 \dots M)$ **do**
    $x_{adv} = generateAdversarial(x_i, \epsilon, \alpha, K)$
    $grad = \nabla_\theta L(x_{adv}, y; \theta)$
    $\theta = \theta - \eta \cdot grad$
  end for
end for

Pseudokod 2 prikazuje općeniti algoritam za učenje s neprijateljskim primjerima. Pritom inicijalizaciju parametara modela θ prikazujemo pozivom funkcije *initializeModelParams*. Radi općenitosti, generiranje neprijateljskih primjera prikazano je pozivom funkcije *generateAdversarial*. Ta funkcija mogla bi generirati neprijateljske primjere koristeći FGSM metodu, PGD metodu ili neki alternativan način, no to nam za općeniti prikaz nije važno.

3.4. Brzo učenje s neprijateljskim primjerima

U slučaju da model koji učimo nije veoma složen, obično učenje s neprijateljskim primjerima prihvatljiv je način za postizanje robusnih modela otpornih na napade. Ipak, ako učimo složene modele na velikim skupovima podataka, obično učenje s neprijateljskim primjerima presporo je. Uz to, rijetko tko uopće posjeduje dovoljnu količinu računalnih resursa za robusno učenje složenijih modela. Kako bismo riješili ili barem umanjili ovaj problem, predložena su brojna rješenja čijom bi se primjenom ubrzalo robusno učenje. Razmotrit ćemo tri takva rješenja.

3.4.1. Besplatno učenje

Jedno od prvih predloženih rješenja za problem duljine trajanja učenja robusnih modela naziva se besplatno učenje (engl. *free adversarial training*) [11]. Uočeno je da modeli učeni koristeći neprijateljske primjere generirane PGD metodom imaju visoku robusnost, no, kao što je već rečeno, zahtijevaju puno vremena za učenje.

Osnovna ideja besplatnog učenja je sljedeća: unatražnim prolazom osim iznosa gradijenta po parametrima modela izračunamo i iznos gradijenta po ulazu potreban za generiranje neprijateljskih primjera. Kako bi se neprijateljski primjeri iterativno izmjenjivali, na svakoj mini-grupi model uči nekoliko puta zaredom. Ovaj hiperparametar zovemo ponavljanje (engl. *replay*). Važno je uočiti da, kako bismo i dalje trebali otprilike jednako vremena za učenje modela, moramo smanjiti broj epoha za faktor ponavljanja. Ako svaku mini-grupu unutar jedne epohe ponavljamo 8 puta, broj epoha bit će 8 puta manji naspram broja epoha kod modela učenog prirodnim učenjem. Nažalost, povećavanjem broja ponavljanja dolazi do degradacije performansi modela na običnom testnom skupu - u pitanju je kompromis između željene razine robusnosti i točnosti na običnom testnom skupu.

Algorithm 3 Besplatno učenje s neprijateljskim primjerima. Prilagođeno iz [12]

Ulaz: x – ulazne slike, N – broj epoha, M – veličina skupa za učenje, K – broj ponavljanja (engl. *replay*), η – stopa učenja, ϵ – ograničenje perturbacije

---

$\theta = initializeModelParams()$
$\delta = 0$
for $(ep = 1 \dots N/K)$ **do**
  for $(i = 1 \dots M)$ **do**
    for $(j = 1 \dots K)$ **do**
      $gradAdv, grad = \nabla L(x_i + \delta, y; \theta)$
      $\delta = \delta + \epsilon \cdot sign(gradAdv)$
      $\delta = clamp(\delta, -\epsilon, \epsilon)$
      $\theta = \theta - \eta \cdot grad$
    end for
  end for
end for

Pseudokod 3 prikazuje algoritam za besplatno učenje s neprijateljskim primjerima. Kao i u prethodnom algoritmu, parametri modela θ inicijaliziraju se pozivom funkcije *initializeModelParams*. Važno je uočiti da se iznosi gradijenata potrebni za generiranje neprijateljskih primjera, ali i izmjenu parametara modela izračunavaju u istom unatražnom prolazu, time osiguravajući da generiranje neprijateljskih primjera ne usporava postupak učenja. Uz to, perturbacija δ inicijalizira se na 0 samo na početku postupka učenja - kada započinje izračun perturbacije za sljedeći ulaz, prethodni iznos služi kao "iznos za zagrijavanje" (engl. *warmup*).

Ovakvim načinom robusnog učenja, duljina trajanja učenja modela otprilike je jednaka kao i duljina trajanja učenja modela prirodnim učenjem. Uz to, rezultati rada [11] pokazuju da je robusnost naučenih modela usporediva s robusnošću modela učenih koristeći neprijateljske primjere generirane PGD metodom. Kao što je već rečeno, visoku robusnost modela učenih koristeći besplatno učenje možemo postići izmjenom broja ponavljanja, ali po cijeni smanjenja performansi modela na običnom testnom skupu.

3.4.2. Brzo učenje

4. Zaključak

Zaključak.

LITERATURA

[1] Bojana Dalbelo Bašić, Marko Čupić, i Jan Šnajder. Umjetne neuronske mreže, prezentacija, 2020.

[2] Hossein Gholamalinezhad i Hossein Khosravi. Pooling methods in deep neural networks, a review. *arXiv preprint arXiv:2009.07485*, 2020.

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

[4] Ian J Goodfellow, Jonathon Shlens, i Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, i Jian Sun. Deep residual learning for image recognition. U *Proceedings of the IEEE conference on computer vision and pattern recognition*, stranice 770–778, 2016.

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

[7] Bai Li, Shiqi Wang, Suman Jana, i Lawrence Carin. Towards understanding fast adversarial training. *arXiv preprint arXiv:2006.03089*, 2020.

[8] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, i Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[9] Keiron O'Shea i Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

[10] Farheen Ramzan, Muhammad Usman Khan, Asim Rehmat, Sajid Iqbal, Tanzila Saba, Amjad Rehman, i Zahid Mehmood. A deep learning approach for automated diagnosis and multi-class classification of alzheimer's disease stages using

resting-state fmri and residual neural networks. *Journal of Medical Systems*, 44, 12 2019. doi: 10.1007/s10916-019-1475-2.

[11] Ali Shafahi, Mahyar Najibi, Mohammad Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, i Tom Goldstein. Adversarial training for free! *Advances in Neural Information Processing Systems*, 32, 2019.

[12] Eric Wong, Leslie Rice, i J Zico Kolter. Fast is better than free: Revisiting adversarial training. *arXiv preprint arXiv:2001.03994*, 2020.

Algoritmi za brzo učenje na neprijateljskim primjerima

Sažetak

Sažetak na hrvatskom jeziku.

Ključne riječi: Ključne riječi, odvojene zarezima.

Algorithms for fast robust training on adversarial examples

Abstract

Abstract.

Keywords: Keywords.