	UNIVERZITET „UNION“ **RAČUNARSKI FAKULTET** Knez Mihailova 6/VI 11000 BEOGRAD	Broj: Datum: 11.09.2018.

UNIVERZITET UNION
RAČUNARSKI FAKULTET
BEOGRAD
Računarske nauke

Sigurnost modela dubokog učenja

DIPLOMSKI RAD

Kandidat: Stanko Kovačević
Broj indeksa: RN 29/13

Mentor: prof. Dr Dragan Urošević

Beograd, 18.12.2016.

Sadržaj

1. Apstrakt

Tema diplomskog rada je istraživanje i razvoj napadačkih i odbrambenih tehnika (eng. *Adversarial Attacks*) nad modelima dubokog učenja (eng. *Deep Learning Models*).

Veliki deo današnje tehnologije zasniva se na sistemima koji su sposobni da donesu odluke, i reše instance problema za koje prethodno nisu eksplicitno programirani. Tipičan primer je prepoznavanje objekta na slici. Ne postoji tačno eksplicitno pravilo prema kome možemo da napišemo egzaktan deterministički algoritam koji može da klasifikuje sadržaj slike. Čak i kada bi bilo moguće razviti takvo pravilo, postojao bi drugi praktični problem zbog toga sto je problem klasifikacije šablona NP-hard. Danas se ove praktične poteškoće prevazilaze upotrebom mašinskog učenja (eng. *Machine Learning*), a najpopularniji modeli su duboke veštačke neuralne mreže (eng. *Deep Neural Networks*), koje pomoću statističke estimacije i matematičke optimizacije omogućavaju kreiranje algoritama koji iz velikog skupa podataka mogu sami da nauče *fuzzy* pravila za rešavanje problema koji su veoma značajni u praksi.

Problem sa pristupom mašinskog učenja je što skoro svi modeli uče distribuciju podataka, tj. uče karakteristike stohastičkog procesa koji generiše podatke, pa nad tim karakteristikama grade *fuzzy* semantičku reprezentaciju, a ne semantički model znanja direktno (npr. model će da nauči da ako se na slici pojavljuju nos, usta, oči, i uši zajedno, da je to slika lica sa verovatnoćom od npr. 0.99, ali model neće da nauči da lice ima usta, oči, uši, nos, itd). Zbog toga je moguće modifikovati ulaz u ove modele kako bi se distribucija iskrivila i tako omeo rad modela i model doterao u stanje koje odgovara napadaču. Ovo predstavlja veliki problem zato što su inteligentni sistemi svuda oko nas (banke, automobili bez vozača, medicina, sigurnost računarskih sistema, itd).

Cilj diplomskog rada jeste predstavljanje nekih od osnovnih ali veoma moćnih tehnika napada na modele dubokog učenja zasnovanje na metodi gradijenta poput klasične propagacije unazad po slici, FGSM (eng. *Fast Gradient Sign Method*). Diplomski rad će također sadržati i tehnike za odbranu od ovih napada, i pokazati koliko je ovo težak problem. Tehnike za odbranu od ovakvih napada su otvoren problem u računarskim naukama, i ne postoji metod koji povoljno rešava i štiti od bilo kog napada. Također će biti prikazano kako se praktično izvodi ovaj napad pomoću aproksimacije modela, i svojstva prenosa protivničkih podprostora (eng. *Adversarial Subspace Transfer Property*).

Diplomski rad će sadržati i aplikacije napisane u programskom jeziku Pajton, koje će služiti kao primeri teorije koja se bude izlagala. U tu svrhu će da koriste biblioteke *TensorFlow* (biblioteka koja omogućava lako i jednostavno konstruisanje dubokih modela koji se pokreću na grafičkoj kartici), kao i raznih statističkih i grafičkih biblioteka za generisanje izveštaja i prikaza potrebnih za ilustracije ideja.

2. Oznake

Skup: Nakrivljeno veliko slovo, npr. A

Skalar: Nakrivljeno malo slovo, npr. x, y, z

Vektor: Nakrivljeno podebljano malo slovo, npr. $\boldsymbol{x}, \boldsymbol{y}$. i-ti element vektora $\boldsymbol{x}$ se označava sa podskriptom, npr. x_i

Matrica: Nakrivljeno podebljano veliko slovo, npr. $\boldsymbol{\Theta}, \boldsymbol{X}, \boldsymbol{Y}$. Elementi matrice se označavaju duplom podskriptom. Element u i-tom redu i j-toj koloni matrice $\boldsymbol{X}$ se označava sa $X_{i,j}$

Vektorski prostor matrica: Skup $\mathcal{M}(n, m)$ predstavlja skup svih matrica koje imaju n redova i m kolona.

Tensor: Podebljano veliko slovo, npr. $\boldsymbol{\Theta}$. U ovom radu se koriste tensori 3 reda, tj. niz matrica. l-ta matrica tensora $\boldsymbol{\Theta}$ se označava sa $\boldsymbol{\Theta}^l$, a konkretan element tensora u i-tom redu, j-toj koloni matrice l sa $\Theta^l_{i,j}$

Skalarni (unutrašnji) proizvod: Ako su $\boldsymbol{x}, \boldsymbol{y}$ vektori iz istog vektorskog prostora, njihov skalarni proizvod se definiše kao $r = \langle \boldsymbol{x}, \boldsymbol{y} \rangle$, gde $r = \sum_i x_i * y_i$

Tensor (vanjski) proizvod: Ako su $\boldsymbol{x}, \boldsymbol{y}$ vektori, njihov vanjski proizvod je matrica $\boldsymbol{Z} = \boldsymbol{x} \otimes \boldsymbol{y}$ **gde** $Z_{i,j} = x_i * y_j$

Hadamardov proizvod: Ako su $\boldsymbol{x}, \boldsymbol{y}$ vektori, njihov Hadamardov proizvod se definise kao $\boldsymbol{z} = \boldsymbol{x} \circ \boldsymbol{y}$ gde $z_i = x_i * y_i$

Funkcija: Oznake funkcija se uvode na mestu na kome se prvi put pojavljuju. Funkcije mogu biti parametrizovane, a parametri se postavljaju posle znaka ;. Parametri su ulaz u funkciju koji je fiksiran, npr. $f(x; \boldsymbol{\theta})$.

Uslovna verovatnoća: U tekstu ćemo koristiti $p(\boldsymbol{y}|\boldsymbol{x})$ da označimo uslovnu verovatnoću realizacije vektora $\boldsymbol{y}$ ako znamo da se realizovao vektor $\boldsymbol{x}$

3. Osnove dubokog učenja

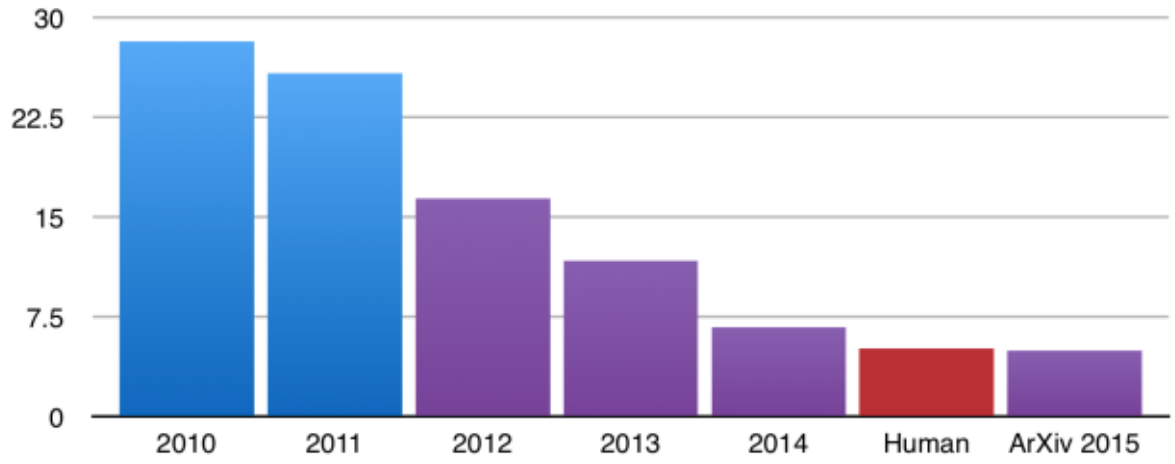
Još od pojave prvih mašina, ljudi su bili opsednuti time da nateraju mašine da budu autonomne, i da razmišljaju kao ljudi. Razlog tome je ljudska priroda da bude radoznao, ali i velika praktična primena u industriji i poboljšavanju kvaliteta života ljudi. Međutim, ispostavilo se da je ovakve mašine veoma teško napraviti iz dva prosta razloga. Prvi razlog je mehanički. Čovek ima veoma preciznu kontrolu nad svim svojim zglobovima, i veoma složen i precizan kinetički lanac, koji sa centralnim nervnim sistemom održava ravnotežu čoveka, koordiniše svim njegovim pokretima, pruža povratne informacije u vidu čula. Konstrukcija motora koji je precizan i energetski efikasan kao ljudski zglob je praktično zahtevna. Drugi, daleko složeniji problem je inteligencija. Ljudski centralni nervni sistem je povezan sa svim senzorima (čulima), aktivatorima (kinetičkim lancem i zglobovima), i omogućava njihovu koordinaciju. Međutim, ljudski mozak omogućava i daleko složenije stvari kao što su donošenje odluka, učenje, apstrakcija okruženja, rekonstrukcija okruženja iz nervnih impulsa poslatih iz organa za vid i organa za sluh.

Uzmimo na primer ljudski vid. Čovek genetski ima ugrađene određene sposobnosti i karakteristike čula vida koje su se razvile procesom evolucije, ka na primer strah od zmija i paukova [1], fokusiranje na lica, kao i čitav sistem koji rekonstruiše mentalnu sliku onog što ljudske oči vide. Međutim, da bi čovek funkcionisao, i preživeo u svom okruženju, on mora konstantno da uči i da povezuje te mentalne slike sa prethodnim iskustvom. Na primer, da bi čovek mogao da čita knjigu, on prvo mora da nauči kako knjiga izgleda. Ovaj proces izvlačenja informacije sa mentalne slike i generalizacija oblika knjige je veoma efikasan kod ljudi, i čovek veoma brzo generalizuje i uči da prepozna svakodnevne predmete.

3.1. Istorija i razvoj modela

Može se reći da mašinsko učenje potiče još od 1763 godina pojavom matematičke statistike i začetka Bajesove teoreme [2] na kojoj se zasniva većina algoritama mašinskog učenja, ali tek od 1930-tih godina se ova disciplina aktivno izučava kao oblast veštačke inteligencije. Istraživanja u neurologiji, Šenonova teorija informacija, i Alan Turingova teorija izračunljivosti su doprinele razvoju ideja prvih agenata veštačke inteligencije. MekKulok, i Pits [3] su prvi koji su pokušali da naprave model koji će kasnije postati popularan kao neuralne mreže. 1950 godine Alan Turing postavlja osnove veštačke inteligencije Turingovim testom [4], a već 1951 godine Marvin Minski konstruiše prvi neuralni računar *SNARC.* Svi dosadašnji modeli nisu imali sposobnost učenja, nego su samo opisivali algoritamski potrebne korake pod uslovom da su svi parametri već zadani. To se menja 1957 godine kada Frenk Rosenblat objavljuje algoritam učenja Perceptron. Iako su na početku kreirali veliko interesovanje u akademiji, 1970 godina mašinsko učenje ulazi u svoje mračno doba. Minski je pokazao da neuralne mreže imaju ograničenja, i da Perceptron ne može da nauči probleme koji su linearni, ali da ne može da nauči složenije funkcije poput XOR. Jedan od najznačajnijih događaja koji neuralne mreže vraća u život jeste razvoj algoritma učenja sa propagacijom

unazad [5], koji omogućava da se treniraju složeni neuralni modeli sposobni da nauče da aproksimiraju nelinearne funkcije. Ovaj algoritam se u različitim oblicima koristi i danas. U istom periodu se pojavljuju i razne arhitekture neuralnih mreža kao što su rekurentne neuralne mreže (Hopfild) [6], kao i konvolucione neuralne mreže (*Li Kun*) [7], a kasnije 1997 se pojavljuje i LSTM arhitektura rekurentnih neuralnih mreža (Hošreiter, Smithaber) [8]. Iako



Ilustracija 1: top-5 greška na ImagineNet skupu

su sve tehnike koje se koriste i danas razvijene mnogo ranije, ipak sve do 2012 godine neuralne mreže su bile potisnute od strane statističkih algoritama poput mašina vektora podrške (eng. *Support Vector Machines, SVM*), stabala odlučivanja, i slično. Razlog maloj popularnosti neuralnih mreža je bila složenost treninga, jer je za uspešnu konvergenciju potreban veliki skup trening podataka, kao i velika procesorska moć. Ranije nije bilo moguće prikupiti dovoljnu količinu podataka, ali poboljšanjem pristupačnosti tehnologije, postaje lakše prikupiti podatke. Također, tih godina počinje era primene grafičkih kartica u naučne svrhe. 2012 godina je prekretnica dubokog učenja kada na *ImageNet* takmičenju po prvi put pobjeđuje duboka neuralna mreža *AlexNet* [9].

3.2. Matematički model mašinskog učenja

Mašinsko učenje se može definisati na razne načine, međutim, opšte prihvaćena definicija je sledeća:

Definicija 1: Za program kažemo da uči da reši zadatak *T* ako se sa povećanjem iskustva *E*, povećavaju i performanse *P* programa.

Primer zadatka bi bio klasifikacija da li je na slici mačka ili ne, ili predikcija cene kuće na osnovu parametara kao što su površina, lokacija, starost kuće. Ovakvo učenje se naziva učenje nadgledanjem (eng. *Supervised Learning*). Iskustvo uglavnom predstavlja skup podataka koji se izvlači i prikuplja iz populacije, ili se simulira i meri kod učenja pojačavanjem (eng. *Reinforcement learning*). Primer skupa podataka za trening klasifikatora mačke bi bio

skup slika od polovina sadrži mačku, a druga polovina ne. Postoji čitava oblast o tome kako se prikupljaju podaci za test i trening, i u nastavku se nećemo baviti tim problemima i pretpostavićemo da su podaci iz trening skupa statistički nezavisni, i da imaju istu statističku raspodelo verovatnoće. Performanse programa predstavljaju meru koliko model dobro aproksimira podatke koje je video u toku treninga, tj. Koliko dobro aproksimira funkciju koja mu je zadana da je nauči. Konkretni detalji funkcije performanse zavise od problema koji se rešava, a najčešće je to srednja kvadratna greška (eng. *Mean Squared* Error, MSE, L2), preciznost, odziv, ROC kriva (eng. *Receiver Operator Characteristic*), i druge. U ovom radu ćemo se fokusirati na problem klasifikacije tako da će nam performansa *P* biti preciznost i odziv koje ćemo definisati kasnije. Sve tehnike opisane u nastavku ovog rada analogno se mogu primeniti i na druge funkcije *P* pod uslovom da su te funkcije diferencijalne skoro svuda. Ovakvo ograničavanje na specifičan problem klasifikacije omogućava da se iznesu ciljne ideje, i da se prije toga čitalac uvede u materiju bez toga da se mu se odvuče pažnja od primarnog cilja ovog rada, a to su napadi na duboke modele. Zadatak *T* će u daljem tekstu biti zadatak klasifikacije, tj. mapiranje ulaznog signala u vektor raspodele verovatnoća

$$\boldsymbol{y}' = f(\boldsymbol{x}; \boldsymbol{\theta}), y_j' > 0, \sum_j y_j' = 1$$

tako da se aproksimira stohastički proces $\boldsymbol{y} = \tau(\boldsymbol{x})$ koji generiše podatke, i koji nam je nepoznat. Koncept vektor $\boldsymbol{y}$ ima kategoričnu raspodelu, gde *i*-ti element ima vrednost 1 ako se u ulaznom signalu pojavljuje koncept *i,* i on je observabilan. *f* je familija funkcija kojom pokušavamo da aproksimiramo nepoznati model τ, a $\boldsymbol{\theta}$ je parametar koji definiše konkretnu funkciju iz familije *f*. Preciznije, koncept vektor definišemo na sledeći način.

Definicija 2: Neka je $\boldsymbol{y}$ diskretan *m* dimenzionalni slučajni vektor, čija podrška je

$$C_m = \left\{ \boldsymbol{y} \in \{0,1\}^m : \sum_{j=1}^{m} y_j = 1 \right\}$$

Neka su $p_1, p_2, \dots, p_m$ pozitivni brojevi tako da

$$\sum_{j=1}^{m} p_j = 1$$

Kažemo da je $\boldsymbol{y}$ koncept vektor ako je udružena marginalna raspodela:

$$p(y_1, y_2, \dots, y_m) = \begin{cases} \prod_{j=1}^{m} p_j^{y_j}, & (y_1, y_2, \dots, y_m) \in C_m \\ 0, & (y_1, y_2, \dots, y_m) \notin C_m \end{cases}$$

Primer: Neka imamo model $\boldsymbol{y}' = f(\boldsymbol{x}; \boldsymbol{\theta})$ koji klasifikuje da li se na slici nalazi mačka, pas, ili ptica. Vektor $\boldsymbol{x}$ bi u ovom slučaju predstavljao sliku, dok bi se koncept vektor mogao definisati na sledeći način: $\boldsymbol{y} = (1,0,0)$ ako je na slici mačka, $\boldsymbol{y} = (0,1,0)$ ako je na slici pas,

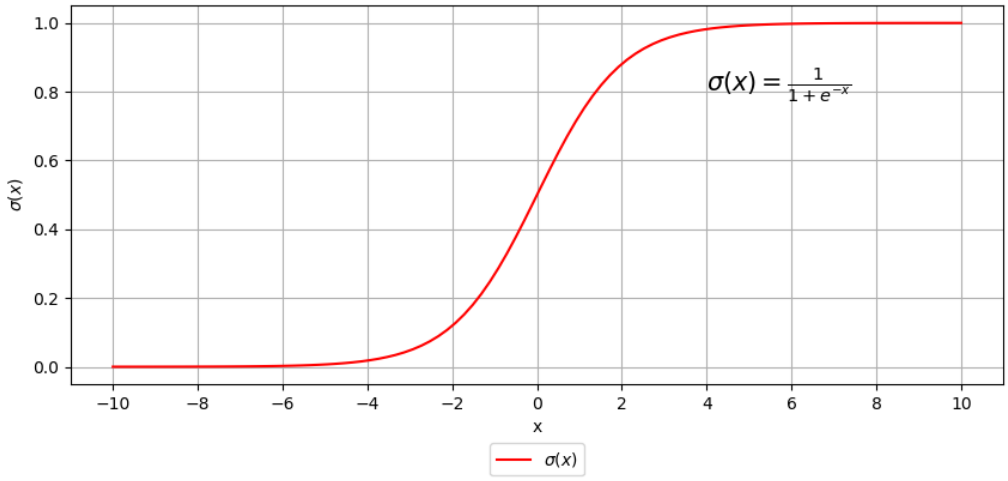
ili $\boldsymbol{y} = (0, 0, 1)$ ako je na slici ptica. Izlaz $\boldsymbol{y}'$ bi bio vektor koji aproksimira uslovnu raspodelu $p_f(\boldsymbol{y}\,|\boldsymbol{x})$**.** Iako je marginalna raspodela korisna kod ekploratorne analize, nas će više zanimati uslovna raspodela $p_h(\boldsymbol{y}|\boldsymbol{x})$, gde je h model koji opisuje generativni proces. U slučaju kada je $h = \tau$, tj. kod potpunog opisa, $p_h(\boldsymbol{y}|\boldsymbol{x}) = \boldsymbol{y}$, tj. za svaki ulazni vektor se tačno zna koji je koncept (pretpostavka je da je proces deterministički).

3.2.1. Osnovni gradivni blokovi dubokih modela

Glavno pitanje kod dizajna sistema zasnovanog na mašinskom učenju jeste koju funkciju izabrati za aproksimaciju. Za početak možemo se ograničiti na konkretnu familiju funkcija *f*, kao što je logistička funkcija

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

Ova funkcija modeluje verovatnoću $p_f(y = 1|\boldsymbol{x})$, gde je *y* skalarna vrednost, a pošto direktno modeluje raspodelu, spada u diskriminacione modele. U slučaju binarne klasifikacije koncept vektor $\boldsymbol{y}$ se obično modeluje kao skalarna vrednost i to $y = \begin{cases} 1, & \boldsymbol{y} = (1, 0) \\ 0, & \boldsymbol{y} = (0, 1) \end{cases}$



Ilustracija 2: Sigmoidalna funkcija

Kada uvrstimo ovu funkciju, naš model postaje

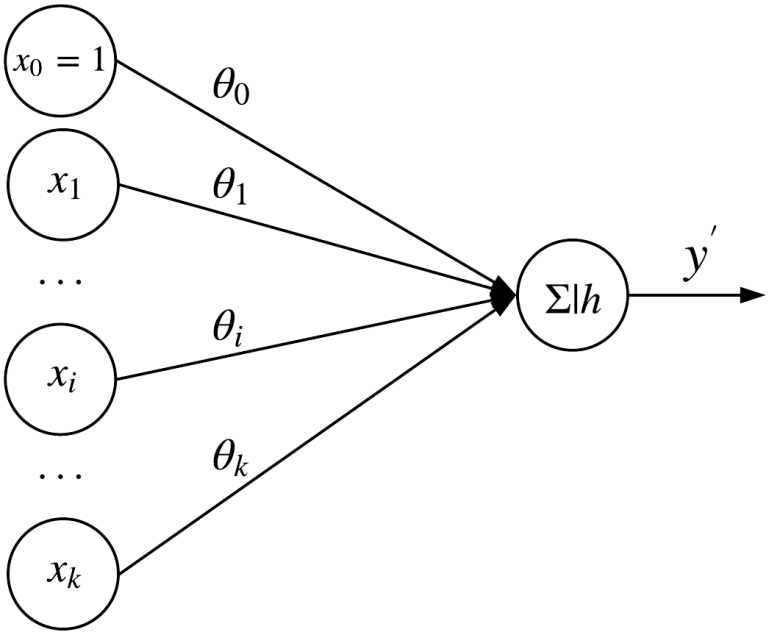
$$\boldsymbol{y}' = f(\boldsymbol{x}; \boldsymbol{\theta}) = \sigma(b + \langle \boldsymbol{x}, \boldsymbol{\theta} \rangle) = \frac{1}{1 + e^{-(b + x_1 * \theta_1 + x_2 * \theta_2 + \cdots + x_k * \theta_k)}}, \qquad f\colon \mathbb{R}^k \to (0, 1)$$

Ovim modelom modelujemo uslovnu verovatnoću da je na slici koncept 1, ako je ulaz $\boldsymbol{x}$**.**

Ako stavimo da je $x_0 = 1,\ b = \theta_0$, tada dobijamo kompaktniji način reprezentacije koji se može vektorizovati i efikasno izračunati na grafičkom procesoru. U nastavku ćemo

pretpostavljati da je odrađeno pre procesiranje i da se ulaznom vektoru $\boldsymbol{x}$, pridruzio elemenat $x_0 = 1$, i da je $\boldsymbol{\theta}$ k+1 dimenzionalni vektor parametara, pa logistička funkcija poprima oblik

$$f(\boldsymbol{x};\boldsymbol{\theta}) = \sigma(\langle \boldsymbol{x},\boldsymbol{\theta}\rangle) = \frac{1}{1+e^{-\langle \boldsymbol{x},\boldsymbol{\theta}\rangle}} = \frac{1}{1+e^{-(x_0*\theta_0+x_1*\theta_1+x_2*\theta_2+\cdots+x_k*\theta_k)}}, f\colon \mathbb{R}^{k+1} \to (0,1)$$

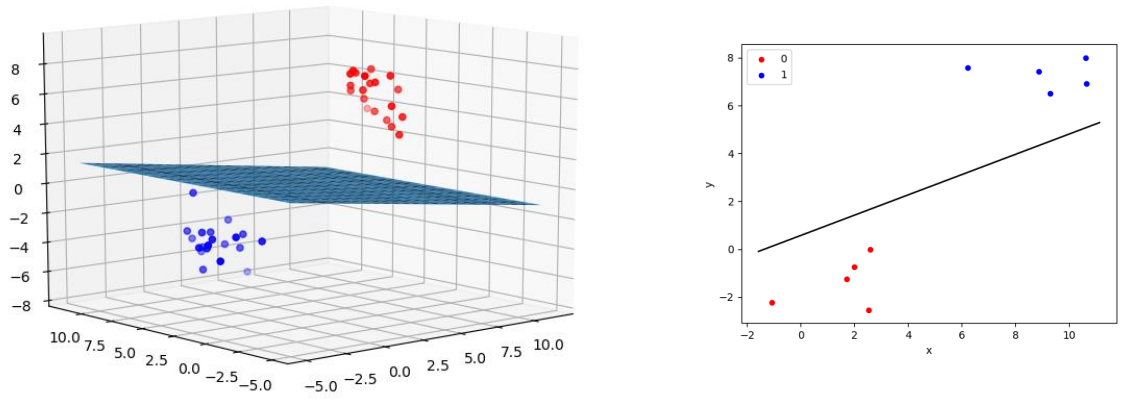


Ilustracija 3: Graf računa logističke regresije

Elementi ulaznog vektora $\boldsymbol{x}$ se množe sa elementima vektora parametara $\boldsymbol{\theta}$, zatim se sabiraju, i propuštaju kroz aktivacionu funkciju h, koja je u ovom slučaju σ. Ovaj blok iznad je osnovni gradivni blok neuralnih mreža kao što ćemo videti kasnije.

Ova klasa funkcija deli $\mathrm{k}+2$ dimenzionalni prostor na 2 dela sa $\mathrm{k}+1$ dimenzionalnom hiper-ravni (k dimenzionalan podprostor)

$$\langle \boldsymbol{x},\boldsymbol{\theta}\rangle = \sum_{j=0}^{k} \theta_j x_j = 0$$



Ilustracija 4: (Levo) Granica podele u 2d ravni. Sve tačke iznad linije sa klasifikuju kao plave, a ispod linije kao crvene. (Desno) Granica podele u 3d prostoru.

Logistička regresija je veoma bitan model zbog toga što predstavlja diferencijalnu verziju Hevisajdove funkcije koja je ključna kod modela neuralnih mreža, i predstavlja gradivni blok klasičnih neuralnih mreža. Logistička regresija se generalizuje na više kategoričnu klasifikaciju pomoću *softmax* funkcije

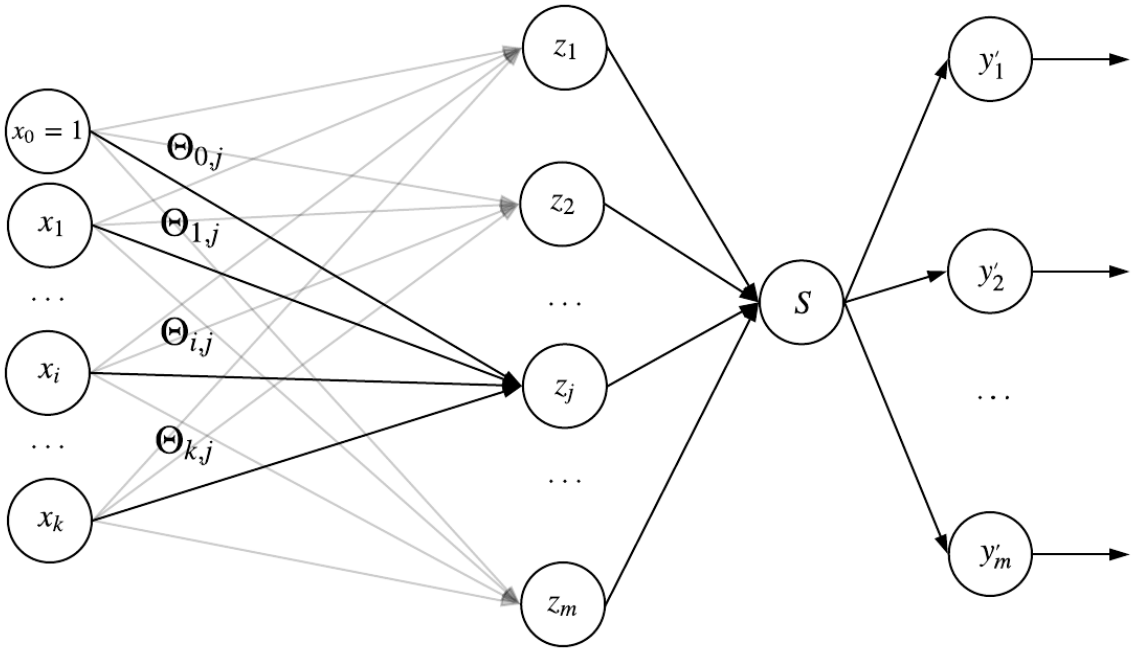
$$\boldsymbol{s} = \mathrm{S}(\mathbf{z}), s_j = \frac{e^{z_j}}{\sum_{p=1}^{k} e^{z_p}}, s_j > 0\,, \sum_{j=1}^{m} s_j = 1, \mathrm{S}\colon \mathbb{R}^k \to (0,1)^m$$

Softmax funkcija sabija vektor $\boldsymbol{z}$ u vektor verovatnoća i tako omogućava konstrukciju diskriminacionog modela $p_f(\boldsymbol{y}|\boldsymbol{x})$, gde je $\boldsymbol{x}$ ulazni vektor, a $\boldsymbol{y}$ koncept vektor**.** Umjesto vektora parametara $\boldsymbol{\theta}$, sada imamo matricu parametara $\boldsymbol{\Theta}$, gde svaka kolona $\boldsymbol{\Theta}_{:,i}$ predstavlja ugrađeni vektor parametara za i-ti koncept.

$$\boldsymbol{y}' = f(\boldsymbol{x}; \boldsymbol{\Theta}) = S(\boldsymbol{x}^{\boldsymbol{T}}\boldsymbol{\Theta}),$$

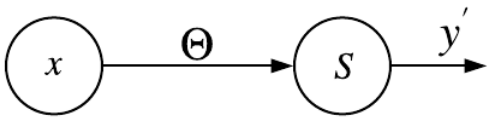
$$f\colon \mathbb{R}^{k+1} \to (0,1)^m$$

$$\boldsymbol{\Theta} \in \mathcal{M}(k+1, m)$$



Ilustracija 5: Graf softmax regresije.

Model *softmax* regresije je sposoban da klasifikuje ulazni signal kad postoje više od 2 koncepta. Drugi način da ilustrujemo *softmax* regresiju jeste da grupišemo elemente vektora u jedan čvor. Ovaj način će biti veoma koristan kasnije kada pređemo na duboke modele jer bi u suprotnom bilo veoma teško prestaviti toliko veliki broj čvorova.



Ilustracija 6: Kompaktan prikaz sofmax regresije

3.2.2. Statistička estimacija

Postavlja se pitanje kako izabrati parametre $\boldsymbol{\Theta}$ da bi model konvergirao i rezultirao dobrim performansama, tj. da bi model aproksimirao funkciju koja mu je data da nauči, a u našem slučaju, to je raspodela $p_\tau(\boldsymbol{y}|\boldsymbol{x})$. Za podatke koje smo već videli u trening skupu podataka tačno znamo raspodelu $p_\tau(\boldsymbol{y}|\boldsymbol{x})$, međutim, naš cilj je da naučimo raspodelu za čitavu populaciju, a ne samo mali uzorak koji nam je potpuno poznat. Kao što smo već rekli, uslovnu raspodelu $\boldsymbol{y} = p_\tau(\boldsymbol{y}|\boldsymbol{x})$ nad populacijom ćemo aproksimirati raspodelom

$$\boldsymbol{y}' = p_f(\boldsymbol{y}|\boldsymbol{x}) = f(\boldsymbol{x};\boldsymbol{\Theta})$$

Da bi našli parametre $\boldsymbol{\Theta}$, potrebno je da u svakom trenutku znamo koliko je naša trenutna aproksimacija dobra, tj. koliko je p_τ daleko od p_f. Drugim rečima, potrebna nam je metrika, tj. distanca između 2 slučajne raspodele. Jedna takva metrika (polu-metrika) je unakrsna entropija [10]

Definicija 3: Unakrsna entropija između dve slučajne promenjive p, i q se definiše kao $H(p,q) = E_p[-\log q] = -\sum_x p(x)\log q(x)$

Unakrsna entropija ima nekoliko interpretacija. Prva zasnovana na teoriji informacija pokazuje koliko bita u proseku treba da se kodiraju simboli izvora ako se koristi šema kodiranja optimalna za stohastički izvor sa distribucijom q koja može biti procenjena nekom metodom, umjesto prave distribucije p koja je nepoznata. Tačnije, ovo je entropija koja pokazuje prosečan broj bitova potreban ako se koristi „pogrešna“ distribucija.

Teorema 1: Unakrsna entropija $H(p,q)$ je uvek veća od entropije $H(p)$, ili formalno: $H(p,q) > H(p)\ \forall p,q\ \in\ \Omega$, gde je Ω skup svih raspodela verovatnoće.

Dokaz: Šenon [11] je pokazao da prosečan broj bita potreban da se bez gubitka kodira izvor ne može biti manji od njegove entropije $H(p)$, gde je p parametar stohastičkog izvora koji predstavlja raspodelu verovatnoće pojavljivanja simbola. Dakle, ako je naša procenjena distribucija „pogrešna“, kodiranje neće biti optimalno, tj. $H(p,q) > H(p)$.

Unakrsna entropija nam omogućava da izmerimo rastojanje između 2 slučajna vektora $\boldsymbol{y}$, i $\boldsymbol{y'}$, tj. za svaki primerak iz trening skupa $\boldsymbol{x}$, možemo da izmerimo koliko je naš model daleko od stvarnog. Ta mera se naziva *loss* funkcija

$$L(\boldsymbol{y},\boldsymbol{y'}) = L\left(p_\tau(\boldsymbol{y}|\boldsymbol{x}), p_f(\boldsymbol{y}|\boldsymbol{x})\right) = H\left(p_\tau(\boldsymbol{y}|\boldsymbol{x}), p_f(\boldsymbol{y}|\boldsymbol{x})\right)$$

$$L(\boldsymbol{y},\boldsymbol{y'}) = H(\boldsymbol{y},\boldsymbol{y'}) = -\sum_{i=1}^{m} y_i \log y_i'$$

Poslednji deo slagalice jeste da se *loss* funkcija minimizuje na čitavom trening skupu u proseku, a ne samo za jedan ulaz.

Definicija 4: Funkcija greške modela $\boldsymbol{y'} = f(\boldsymbol{x};\boldsymbol{\Theta})$, na trening skupu *E* se definiše na sledeći način:

$$C(\boldsymbol{\Theta}) = \frac{1}{|E|}\sum_{(\boldsymbol{x},\boldsymbol{y})\in\boldsymbol{E}} L(\boldsymbol{y}, f(\boldsymbol{x};\boldsymbol{\Theta}))$$

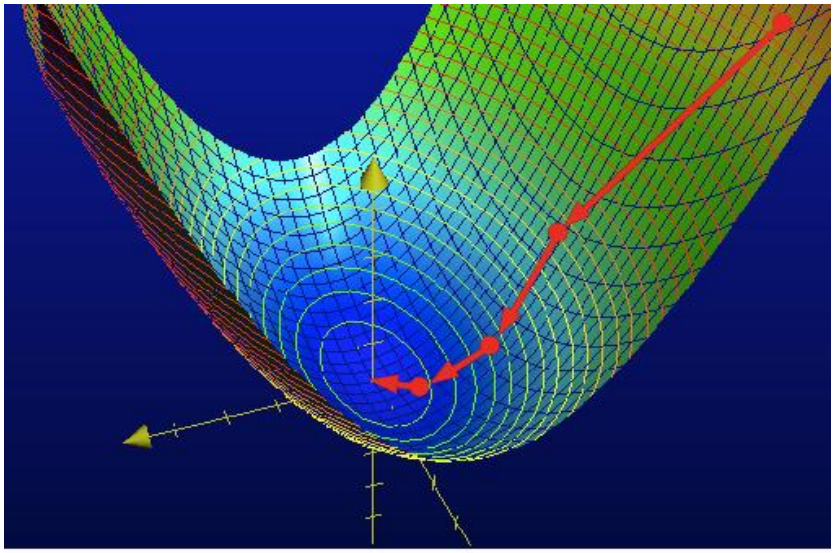
Pošto trening skup *E* predstavlja slučajan uzorak iz neke populacije, svaka estimacija treba da se generalizuje na čitavu populaciju, tj. pristrasnost $bias(f) = E(p_f) - p_\tau$ treba da bude što manja, kao varijansa modela $Var(f) = Var(p_f)$. Greška modela se može dekomponovati na

$$Err(f) = bias(f)^2 + Var(f) + \epsilon$$

gde je ϵ ne merljiva greška. Nažalost, tačna vrednost svih izraza gore nije rešiva, i mogu se raditi samo neke aproksimacije. Generalno pravilo je da kompleksni model spušta pristrasnost ali podiže varijansu modela, a manje kompleksan model podiže pristrasnost, ali spušta varijansu. Glavni problem u praksi kod rešenja sa mašinskim učenjem jest pronalaženje neke „zlatne sredine“ u kojoj imamo dovoljno kompleksan model koji ima kapacitet da nauči raspodelu, ali ne previše tako da se loše generalizuje na populaciju [12, Ch. 7].

3.2.3. Matematička optimizacija

Kada smo tačno definisali funkciju greške nad prostorom parametara, sve što je ostalo jeste da se ta funkcija minimizuje. Analitičko rešenje je veoma teško izvesti, pa se primenjuju iterativni metodi zasnovani na gradijentu funkcije. S toga se zahteva da svi modeli, budu diferencijabilni. Optimizacija se vrši tako što se određen broj puta pomeramo u prostoru parametara u smeru suprotnom gradijentu [12, Ch. 4]. Razlog je taj što gradijent



Ilustracija 7: Spust gradijentom. Algoritam počinje u gornjem uglu i iterativno se spušta do lokalnog maksimuma

funkcije uvek pokazuje u smeru u kome je porast funkcije najveći, pa je očigledno da ako se u toj tački pomerimo suprotno od gradijenta, da imamo najmanji porast, što je u suštini najveći spust.

Spust gradijentom uvek pronalazi rešenje ukoliko je funkcija konveksna. Na žalost, kod dubokih neuralnih mreža, funkcija greške nije konveksna ali to u praksi ne predstavlja problem. Pri svakoj iteraciji, primenjuje se sledeća jednačina

$$\boldsymbol{\Theta} = \boldsymbol{\Theta} - \alpha \boldsymbol{\nabla} C$$

Ovo pravilo se ponavlja određen broj puta ili dok model ne konvergira.

Na primeru ćemo pokazati kako se trenira *softmax* klasifikator. Za to nam je potrebno da znamo $\boldsymbol{\nabla} C = (\frac{\partial C}{\partial \Theta_{0,1}}, \frac{\partial C}{\partial \Theta_{1,1}}, \dots, \frac{\partial C}{\partial \Theta_{k,m}})$.

$$C(\boldsymbol{\theta}) = \frac{1}{|E|} \sum_{(\boldsymbol{x},\boldsymbol{y}) \in E} L\big(\boldsymbol{y}, f(\boldsymbol{x}; \boldsymbol{\theta})\big)$$

$$1. \frac{\partial C}{\partial \Theta_{i,j}} = \frac{1}{|E|} \sum_{(\boldsymbol{x},\boldsymbol{y}) \in E} \frac{\partial L}{\partial \Theta_{i,j}}$$

$$2. \frac{\partial L}{\partial \Theta_{i,j}} = \frac{\partial L}{\partial \boldsymbol{y}'} \frac{\partial \boldsymbol{y}'}{\partial \Theta_{i,j}} = \frac{\partial L}{\partial \boldsymbol{y}'} \frac{\partial \boldsymbol{y}'}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \Theta_{i,j}}$$

$$3. \ \frac{\partial L}{\partial \boldsymbol{y}'} = \boldsymbol{\nabla} L \text{ , gde } \boldsymbol{\nabla} L_k = \frac{\partial H(\boldsymbol{y}, \boldsymbol{y}')}{\partial y_k'} = \ -\frac{y_k}{{y_k}'}$$

$$4. \frac{\partial \boldsymbol{y}'}{\partial \boldsymbol{z}} = \frac{\partial S(\boldsymbol{z})}{\partial \boldsymbol{z}} = \boldsymbol{J}_S$$

$$\text{gde } \boldsymbol{z} = \boldsymbol{x}\boldsymbol{\Theta} = \big(\boldsymbol{x}\boldsymbol{\Theta}_{:,1}, \dots, \boldsymbol{x}\boldsymbol{\Theta}_{:,m}\big)$$

$J_{S_{i,j}} = \mathrm{S}(\boldsymbol{z})_{\mathrm{i}}\big(\delta_{\mathrm{ij}} - \mathrm{S}(\boldsymbol{z})_{\mathrm{j}}\big) = y_i'(\delta_{ij} - y_j')$ – *m*x*m* Džakobijeva matrica *softmax* funkcije

$$\delta_{ij} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}$$

$$5. \frac{\partial \boldsymbol{z}}{\partial \Theta_{i,j}} = \frac{\partial (\boldsymbol{x}\boldsymbol{\Theta})}{\partial \Theta_{i,j}} = \mathbf{Z} = \boldsymbol{x} \otimes \boldsymbol{I}$$

gde je $\boldsymbol{I}$ matrica identičnosti, $\mathbf{Z}$ tensor 3 reda i definisan je kao $Z_{l,j}^{i} = \delta_{l,j} x_i$

4,3 ->2:

$$6.\frac{\partial L}{\partial \Theta_{i,j}} = (\nabla L \boldsymbol{J_S})\frac{\partial \boldsymbol{z}}{\partial \Theta_{i,j}} = (\boldsymbol{y}' - \boldsymbol{y})\frac{\partial \boldsymbol{z}}{\partial \Theta_{i,j}}$$

5->6:

$$7.\frac{\partial L}{\partial \Theta_{i,j}} = \big(\boldsymbol{x} \otimes (\boldsymbol{y}' - \boldsymbol{y})\big)_{i,j} = \big(y_j' - y_j\big)x_i$$

Na kraju kad uvrstimo 7 u 1, dobijamo konačnu jednačinu:

$$8.\frac{\partial C}{\partial \Theta_{i,j}} = \frac{1}{|E|}\sum_{(\boldsymbol{x},\boldsymbol{y})\in \boldsymbol{E}} \big(y_j' - y_j\big)x_i$$

U praksi se ipak ovakav oblik jednačine 8 ne koristi jer u ovakvom obliku gubimo prednosti izvršavanja algoritma na grafičkoj kartici. Kako je grafička kartica veoma dobra u množenju matrica, potrebno je sve algoritme vektorizovati.

Ako sa $(\boldsymbol{x}^k, \boldsymbol{y}^k)$ označimo k-ti element iz trening skupa *E*, tada jednačina 8 postaje:

$$9.\frac{\partial C}{\partial \Theta_{i,j}} = \frac{1}{|E|}\sum_{k=1}^{|E|}\big(y_j'^k - y_j^k\big)x_i^k$$

Ako sve elemente trening skupa, spakujemo u $|\mathrm{E}|\mathrm{x}k$ matricu $\boldsymbol{X}$, a odgovarajuće koncept vektore u $|\mathrm{E}|\mathrm{x}m$ matricu $\boldsymbol{Y}$, možemo jednačinu 9 zapisati kao:

$$10.\frac{\partial C}{\partial \Theta_{i,j}} = \frac{1}{|E|}\sum_{k=1}^{|E|}\big(Y'_{k,j} - Y_{k,j}\big)X_{k,i} = \frac{1}{|E|}\sum_{k=1}^{|E|}(\boldsymbol{Y}' - \boldsymbol{Y})_{k,i}X_{k,j}$$

Jednačina iznad je (i,j) element matrice $\frac{\partial C}{\partial \boldsymbol{\Theta}} = \nabla C$, a ako malo drugačije zapišemo indekse iznad, lako dolazimo do konačne vektorske formule gradijenta funkcije *C*:

$$11.\nabla C = \boldsymbol{X^T}(\boldsymbol{Y}' - \boldsymbol{Y})$$

Sa jednačinom 11, možemo kompletno da izračunamo gradijent vektor $\nabla C = (\frac{\partial C}{\partial \Theta_{0,1}}, \frac{\partial C}{\partial \Theta_{1,1}}, \dots, \frac{\partial C}{\partial \Theta_{k,m}})$, i iterativno primenjivati opisanu metodu spusta gradijentom. Tehnika koju smo primenili iznad je osnovna ideja koja se krije iza algoritma propagacijom unazad (eng. *Backpropagation*). Ideja je da krenemo od zadnjeg čvora (funkcija greške), i računamo parcijalni izvod samo za taj sloj u odnosu na izlaz sloja ispod. Dobijeni rezultat propagiramo unazad.

Da bi opisani algoritam bio primenjiv u praksi, potrebno je izvršiti regularizaciju modela. Drugim rečima, potrebno je ograničiti model kako ne bi bio previše kompleksan u odnosu na trening podatke, i uveo veliku varijansu u model, tj. preveliku zavisnost od uzorka. Regularizacija prevazilazi obim ovog rada, tako da ćemo je izostaviti. Za više detalja o tehnikama regularizacije, pogledajte [12, Ch. 7]. Također, u praksi se ne vrši zbir nad čitavim skupom *E*, nego se iz tog skupa na slučajan način odabere manji uzorak *B*, i nad njim se radi spust gradijentom. Ovaj uzorak *B*, se ponovo uzorkuje pri svakoj iteraciji algoritma. Ova tehnika naziva se stohastički spust gradijentom [12, Ch. 8]

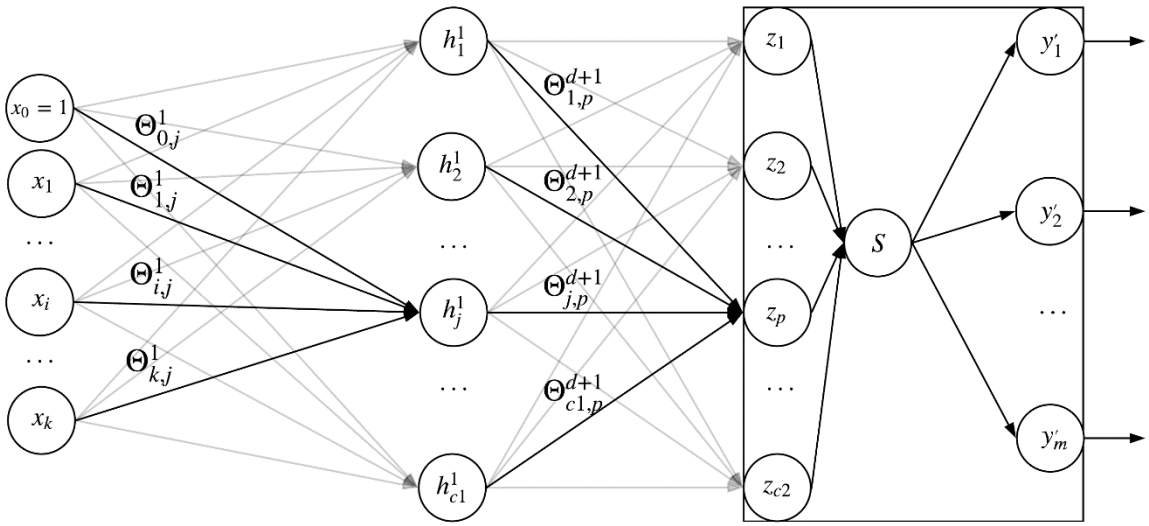
U ovakvom obliku, matrica $\boldsymbol{Y}'$ se dobija kao $S(\boldsymbol{X\Theta})$ gde se *softmax* primenjuje nad redovima matrice $\boldsymbol{X\Theta}$. Ovaj algoritam se naziva propagacija napred (eng. *forward pass*).

Primer koda koji implementira algoritam treniranja propagacijom unazad i algoritam evaluacije propagacijom napred:

```
def softmax(Z):
    return np.exp(Z) / np.sum(np.exp(Z), axis=1, keepdims=True)

def gradC(P, X, Y):
    n = X.shape[0]
    return (1 / n) * np.dot(X.T, (P - Y))

def L(Theta, X, Y):
    n = X.shape[0]
    prob = forward(X, Theta)
    loss = (-1 / n) * np.sum(Y * np.log(prob))
    return loss

def train(X, Y, learning_rate=0.1, iter=100):
    Theta = np.random.normal(scale=0.1, size=(X.shape[1], Y.shape[1]))

    for _ in range(iter):
        P = forward(X, Theta)
        Theta = Theta - learning_rate*gradC(P, X, Y)

    return Theta

def forward(X, Theta):
    Z = np.dot(X, Theta)
    S = softmax(Z)
    return S
```

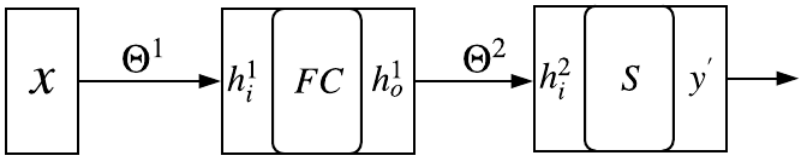
3.3. Duboki modeli – neuralne mreže

Neuralnu mrežu za zadatak klasifikacije dobijamo kada između čvorova ulaza i čvora *softmax* operatora ubacimo dodatne nelinearne čvorove. Ti čvorovi se nazivaju i skriveni čvorovi (eng. *hidden layers*).



Ilustracija 8: Neuralna mreža sa 1 skrivenim slojem.

Glavni deskriptivni parametar arhitekture neuralne mreže jeste dubina, tj. Broj skrivenih slojeva *d.* U primeru iznad, dubina je *d*=1. U tom slučaju, potrebne su nam 2 matrice, jedna za skriveni čvor $\boldsymbol{\Theta}^1$, i druga za izlazni čvor $\boldsymbol{\Theta}^2$**.** Sa c_i označavamo broj čvorova u i-tom skrivenom sloju**.**



Ilustracija 9: Kompaktni prikaz neuralne mreže. Ovakav prikaz je prikladniji za duboke neuralne mreže. Čitav sloj predstavljamo kao jedan čvor koji ima ulaz (levi odeljak), tip (srednji odeljak), i izlaz (desni odeljak). FC je skraćenica od eng. „Fully Connected“, što znači da je svaki neuron u sloju l povezan sa svakim neuronom iz sloja l-1 i sloja l+1.

Glavna ideja dubokih modela jeste da svaki sloj unosi nelinearnost u model tako što se ulaz u sloj l pušta kroz aktivacionu funkciju f_{hl} koja mora da bude diferencijabilna i nelinearna funkcija. Dve funkcije koje se često koriste u praksi jesu sigmoidalna i ReLU funkcija [13], [14]. U nastavku se nećemo detaljno baviti ovim funkcijama jer ne utiču na osnovne koncepte koje ovaj rad istražuje.

Već i za mreže sa samo jednim skrivenim slojem postaje veoma teško da se izvedu jednačine za gradijent i evaluaciju direktno, pa je potreban sistematičan pristup. Umesto da računamo direktno sve slojeve, kretaćemo se jedan po jedan sloj od ulaza $\boldsymbol{x}$ do izlaza prilikom evaluacije, a zatim od izlaza se vraćati ka ulazu računajući parcijalne izvode i koristeći pravilo ulančavanja izvoda složenih funkcija (ideja algoritma propagacijom unazad).

Jednačine algoritma evaluacije (eng. *forward pass*)

$$\boldsymbol{h}_o^1 = f_{h1}(\boldsymbol{h}_i^1)$$
$$\text{generalno: } \boldsymbol{h}_o^l = f_{hl}\left(\boldsymbol{h}_i^{l-1}\boldsymbol{\Theta}^l\right), l = 1, \dots, d$$
$$\boldsymbol{h}_i^l = \boldsymbol{h}_o^{l-1}\boldsymbol{\Theta}^l$$
$$\boldsymbol{h}_o^0 = \boldsymbol{x}$$
$$\boldsymbol{y}' = S\left(\boldsymbol{h}_i^2\right)$$

Matrica $\boldsymbol{\Theta}^l$ ima dimenzije $c_{l-1} \times c_l$

Ako vektorizujemo algoritam sa ulaznim matricama $\boldsymbol{X}$ i $\boldsymbol{Y}$ kao što smo ranije opisali, algoritam evaluacije se zapisuje u sledećem rekurentnom obliku:

$$\boldsymbol{H}_o^l = f_{hl}\left(\boldsymbol{H}_i^l\right) \text{ za } l = 1, 2, \dots, d$$
$$\boldsymbol{H}_i^l = \boldsymbol{H}_o^{l-1}\boldsymbol{\Theta}^l$$
$$\boldsymbol{H}^0 = \boldsymbol{X}$$
$$\boldsymbol{Y}' = S\left(\boldsymbol{H}_i^2\right)$$

Aktivaciona funkcija f_{hl} se primenjuje na svaki element vektora/matrice posebno.

Kod ispod ilustruje algoritam evaluacije propagacijom napred:

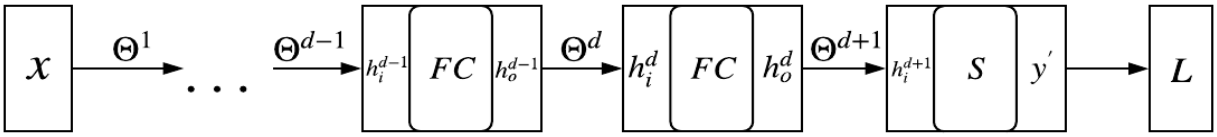
```
def forward(X, ThetaTensor):
    H = X
    d = len(ThetaTensor) - 1

    # Compute hidden layers
    for i in range(d + 1):
        H = sigmoid(H*ThetaTensor[i]) #Sigmoid applied per element

    # Compute output Softmax layer. Sofmax applied per row
    return softmax(H)
```

ThetaTensor je niz matrica $\boldsymbol{\Theta}$ gde ThetaTensor[i]= $\boldsymbol{\Theta^i}$**.** Ideja algoritma jeste da postepeno računamo vrednosti slojeva od ulaza ka izlazu, i tako propagiramo signal sve do izlaza mreže.

Za računanje gradijenta ∇C koristićemo sličnu ideju samo što ćemo se kretati od izlaza ka ulazu. Ideja je da koristimo pravilo ulančavanja izvoda složenih funkcija kako bi kombinovali i izračunali

$$\frac{\partial C}{\partial \Theta_{i,j}^{l}}$$

Krenućemo od izlaznog sloja, i izračunati $\frac{\partial C}{\partial \boldsymbol{\Theta}^{d+1}}$. Prilikom treniranja matrice, ubacuje se implicitan sloj koji računa unakrsnu entropiju pa treba i njega uzeti u obzir.



Ilustracija 10: Kompaktan prikaz duboke neuralne mreže. Oznake ispod strelice su izlaz iz i-tog sloja, a oznake iznad strelice predstavljaju matrice parametara. h_i^l označava ulazni vektor sloja l, a h_o^l označava izlazni vektor sloja l

$$C(\boldsymbol{\theta}) = \frac{1}{|E|} \sum_{(\boldsymbol{x},\boldsymbol{y})\in E} L\big(\boldsymbol{y}, f(\boldsymbol{x};\boldsymbol{\theta})\big)$$

gde je $\boldsymbol{\Theta}$ tensor 3 reda

$$\nabla C = \frac{\partial C}{\partial \boldsymbol{\Theta}} = \left(\frac{\partial C}{\partial \boldsymbol{\Theta}^1}, \dots, \frac{\partial C}{\partial \boldsymbol{\Theta}^{d+1}}\right)$$

Izračunajmo:

$$\frac{\partial L}{\partial \boldsymbol{\Theta}^{d+1}} = \frac{\partial L}{\partial \boldsymbol{y}'}\frac{\partial \boldsymbol{y}'}{\partial \boldsymbol{\Theta}^{d+1}} = \frac{\partial L}{\partial \boldsymbol{y}'}\frac{\partial \boldsymbol{y}'}{\partial \boldsymbol{h}_i^{d+1}}\frac{\partial \boldsymbol{h}_i^{d+1}}{\partial \boldsymbol{\Theta}^{d+1}} =$$
$$= \frac{\partial L}{\partial \boldsymbol{h}_i^{d+1}}\frac{\partial \boldsymbol{h}_i^{d+1}}{\partial \boldsymbol{\Theta}^{d+1}} = (\boldsymbol{y}' - \boldsymbol{y})\frac{\partial (\boldsymbol{h}_o^d \boldsymbol{\Theta}^{d+1})}{\partial \boldsymbol{\Theta}^{d+1}} = \boldsymbol{h}_o^d \otimes (\boldsymbol{y}' - \boldsymbol{y})$$

Ovaj izvod je identičan kao kod *softmax* klasifikatora kojeg smo izračunali ranije, pa

$$\frac{\partial L}{\partial \Theta_{i,j}^{d+1}} = \big(y_j' - y_j\big)(h_o^d)_i$$

Ako označimo sa $\boldsymbol{\delta}_{d+1} = \frac{\partial L}{\partial \boldsymbol{h}_i^{d+1}} = (\boldsymbol{y}' - \boldsymbol{y})$ tada je:

$$\frac{\partial L}{\partial \boldsymbol{\Theta}^{d+1}} = \boldsymbol{h}_o^d \otimes \boldsymbol{\delta}_{d+1}$$

Nastavimo sa računanjem $\frac{\partial L}{\partial \boldsymbol{\Theta}^d}$

$$\frac{\partial L}{\partial \boldsymbol{\Theta}^d} = \frac{\partial L}{\partial \boldsymbol{y}'}\frac{\partial \boldsymbol{y}'}{\partial \boldsymbol{\Theta}^d} = \frac{\partial L}{\partial \boldsymbol{y}'}\frac{\partial \boldsymbol{y}'}{\partial \boldsymbol{h}_i^{d+1}}\frac{\partial \boldsymbol{h}_i^{d+1}}{\partial \boldsymbol{\Theta}^d}$$

Izraz $\frac{\partial L}{\partial \boldsymbol{y}'}\frac{\partial \boldsymbol{y}'}{\partial \boldsymbol{h}_i^{d+1}}$ smo već izračunali ranije kao $\boldsymbol{\delta}_{d+1}$pa ćemo ga ponovo iskoristiti (tehnika memoizacije u dinamičkom programiranju)

$$\frac{\partial L}{\partial \boldsymbol{\Theta}^d} = \boldsymbol{\delta}_{d+1}\frac{\partial \boldsymbol{h}_i^{d+1}}{\partial \boldsymbol{\Theta}^d} = \boldsymbol{\delta}_{d+1}\frac{\partial}{\partial \boldsymbol{\Theta}^d}\boldsymbol{h}_o^d\boldsymbol{\Theta}^{d+1} = \boldsymbol{\delta}_{d+1}(\boldsymbol{\Theta}^{d+1})^{\mathbf{T}}\frac{\partial \boldsymbol{h}_o^d}{\partial \boldsymbol{\Theta}^d}$$
$$\frac{\partial \boldsymbol{h}_o^d}{\partial \boldsymbol{\Theta}^d} = \frac{\partial}{\partial \boldsymbol{\Theta}^d}f_{hd}(\boldsymbol{h}_i^d) = f'_{hd}(\boldsymbol{h}_i^d)\frac{\partial \boldsymbol{h}_i^d}{\partial \boldsymbol{\Theta}^d}$$

Kad uvrstimo u jednačinu iznad,

$$\frac{\partial L}{\partial \boldsymbol{\Theta}^d} = \boldsymbol{\delta}_{d+1}(\boldsymbol{\Theta}^{d+1})^{\mathbf{T}} \circ f'_{hd}(\boldsymbol{h}_i^d)\frac{\partial \boldsymbol{h}_i^d}{\partial \boldsymbol{\Theta}^d}$$

Ako pažljivije pogledamo,

$$\boldsymbol{\delta}_{d+1}(\boldsymbol{\Theta}^{d+1})^{\mathbf{T}} \circ f'_{hd}(\boldsymbol{h}_i^d) = \frac{\partial L}{\partial \boldsymbol{h}_i^d}$$

Označimo ovaj izraz sa

$$\boldsymbol{\delta}_d = \frac{\partial L}{\partial \boldsymbol{h}_i^d} = \boldsymbol{\delta}_{d+1}(\boldsymbol{\Theta}^{d+1})^{\mathbf{T}} \circ f'_{hd}(\boldsymbol{h}_i^d)$$

Jednačina $\frac{\partial L}{\partial \boldsymbol{\Theta}^d}$ postaje

$$\frac{\partial L}{\partial \boldsymbol{\Theta}^d} = \boldsymbol{\delta}_d\frac{\partial \boldsymbol{h}_i^d}{\partial \boldsymbol{\Theta}^d} = \frac{\partial}{\partial \boldsymbol{\Theta}^d}\boldsymbol{h}_o^{d-1}\boldsymbol{\Theta}^{\boldsymbol{d}} = \boldsymbol{h}_o^{d-1} \otimes \boldsymbol{\delta}_d$$

Matematičkom indukcijom možemo pokazati sledeće pravilo koje nazivamo skup jednačina za propagaciju greške unazad:

$$\boldsymbol{\delta}_{d+1} = (\boldsymbol{y}' - \boldsymbol{y})$$

$$\boldsymbol{\delta}_l = \boldsymbol{\delta}_{l+1}(\boldsymbol{\Theta}^{l+1})^{\mathbf{T}} \circ f'_{hl}(\boldsymbol{h}_i^l)$$

$$\frac{\partial L}{\partial \boldsymbol{\Theta}^l} = \boldsymbol{h}_o^{l-1} \otimes \boldsymbol{\delta}_l$$

Sada kada znamo da izračunamo $\frac{\partial L}{\partial \boldsymbol{\Theta}^l}$ možemo da se vratimo nazad na gradijent funkcije greške

$$\frac{\partial C}{\partial \boldsymbol{\Theta}^l} = \frac{1}{|E|} \sum_{(\mathbf{x},\mathbf{y})\in E} \frac{\partial L}{\partial \boldsymbol{\Theta}^l} = \frac{1}{|E|} \sum_{(\mathbf{x},\mathbf{y})\in E} \boldsymbol{h}_o^{l-1} \otimes \boldsymbol{\delta}_l$$

Iako ova jednačina može da se koristi za trening neuralne mreže, u praksi je jako sporo da se vrtimo petljom kroz čitav set sumirajući gradijente. Zato ćemo primeniti isti trik kao i ranije da vektorizujemo ulaz i izlaz neuralne mreže. Ako sve elemente trening skupa, spakujemo u $|E|$x$(k+1)$ matricu $\boldsymbol{X}$, a odgovarajuće koncept vektore u $|E|$xm matricu $\boldsymbol{Y}$, kao što smo ranije naveli, algoritam evaluacije postaje

$$\boldsymbol{H}_o^l = f_{hl}\left(\boldsymbol{H}_i^l\right) \text{ za } l = 1, 2, \dots, d$$
$$\boldsymbol{H}_i^l = \boldsymbol{H}_o^{l-1}\boldsymbol{\Theta}^l$$

$$\boldsymbol{H}^0 = \boldsymbol{X}$$

$$\boldsymbol{Y}' = S\left(\boldsymbol{H}_i^2\right)$$

Ovo znači da se i vektori greške $\boldsymbol{\delta}_l$ vektorizuju u matrice i postaju $\boldsymbol{\Delta}_l$. Trivijalnim algebarskim manipulacijama možemo jednačine propagacije unazad da vektorizujemo u sledeći oblik:

$$\boldsymbol{\Delta}^{d+1} = (\boldsymbol{Y}' - \boldsymbol{Y})$$

$$\boldsymbol{\Delta}^l = \boldsymbol{\Delta}^{l+1}(\boldsymbol{\Theta}^{l+1})^{\mathbf{T}} \circ f'_{hl}\left(\boldsymbol{H}_i^l\right)$$

$$\frac{\partial L}{\partial \boldsymbol{\Theta}^l} = (\boldsymbol{H}_o^{l-1})^{\boldsymbol{T}}\boldsymbol{\Delta}^l$$

$$\text{za } l = 1, 2, \dots, d$$

U primeru ispod je Pajton kod koji implementira potpuno povezanu neuralnu mrežu. Ovaj kod služi da bi poboljšao razumevanje jednačina iznad, i u nastavku rada se neće koristiti.

```
#Module activations.py

def softmax(Z):
    return np.exp(Z) / np.sum(np.exp(Z), axis=1, keepdims=True)

def sigmoid(Z, derivative=False):
    S = 1/(1+np.exp(-Z))
    return np.multiply(S, (1-S)) if derivative else S

def crossentropy(Y_prim, Y):
    return -np.sum(np.sum(Y * np.log(Y_prim), axis=1))/Y.shape[0]
```

```
#Module neural_network.py

class NeuralNetwork:

    def __init__(self, c):
        self.__c = c
        self.__d = len(c)

    @staticmethod
    def load(path):
        with open(path, 'rb') as handle:
            return pk.load(handle)

    def run_epoch(self, X, Y, learning_rate=0.001):
        Theta_Grad = self.__initializeGradTensor(X.shape[1], Y.shape[1])
        Y_prim = self.forward(X)

        print("Epoch ", self.__epoch," completed.",
            " Loss: ", activations.crossentropy(Y_prim, Y))
        self.__epoch = self.__epoch + 1

        Delta = (Y_prim - Y)
        Theta_Grad[self.__d+1]=np.dot(
            activations.sigmoid(self.__H[self.__d]).T,
            Delta)
        self.__ThetaTensor[self.__d+1] -= learning_rate*Theta_Grad[self.__d+1]

        for l in range(self.__d, 0, -1):
            Delta = np.multiply(
                np.dot(Delta, self.__ThetaTensor[l+1].T),
                activations.sigmoid(self.__H[l], True))
            Theta_Grad[l] = np.dot(activations.sigmoid(self.__H[l-1]).T, Delta)
            self.__ThetaTensor[l] -= learning_rate*Theta_Grad[l]

    def train(self, X, Y, learning_rate=0.001, epoch=100):
        self.initialize(X.shape[1], Y.shape[1])
```

```
        for e in range(epoch):
            self.run_epoch(X, Y, learning_rate)

    def __initializeGradTensor(self, input_dim, output_dim):
        GradTensor = [np.zeros((1, 1))] # Dummy matrix for easier interpretation
        GradTensor.append(np.zeros((input_dim, self.__c[0])))
        for l in range(self.__d-1):
            GradTensor.append(
                np.zeros((self.__c[l], self.__c[l+1])))
        GradTensor.append(
            np.zeros((self.__c[self.__d-1], output_dim)))
        return GradTensor

    def initialize(self, input_dim, output_dim):
        self.__ThetaTensor = [np.zeros((1, 1))] # Dummy matrix
        self.__ThetaTensor.append(
            np.random.normal(scale=0.1, size=(input_dim, self.__c[0])))
        for l in range(self.__d-1):
            self.__ThetaTensor.append(
                np.random.normal(scale=0.1, size=(self.__c[l], self.__c[l+1])))
        self.__ThetaTensor.append(
            np.random.normal(scale=0.1, size=(self.__c[self.__d-1], output_dim)))

        self.__epoch = 1

    def save(self, path):
        with open(path, 'wb+') as handle:
            pk.dump(self, handle, protocol=pk.HIGHEST_PROTOCOL)

    def forward(self, X):
        self.__H = [X]

        H_o = X
        # Compute hidden layers
        for l in range(1, self.__d+1):
            H_i = np.dot(H_o, self.__ThetaTensor[l])
            self.__H.append(H_i)
            H_o = activations.sigmoid(H_i)

        # Compute output Softmax layer
        H_i = np.dot(H_o, self.__ThetaTensor[self.__d + 1])
        self.__H.append(H_i)
        return activations.softmax(H_i)
```

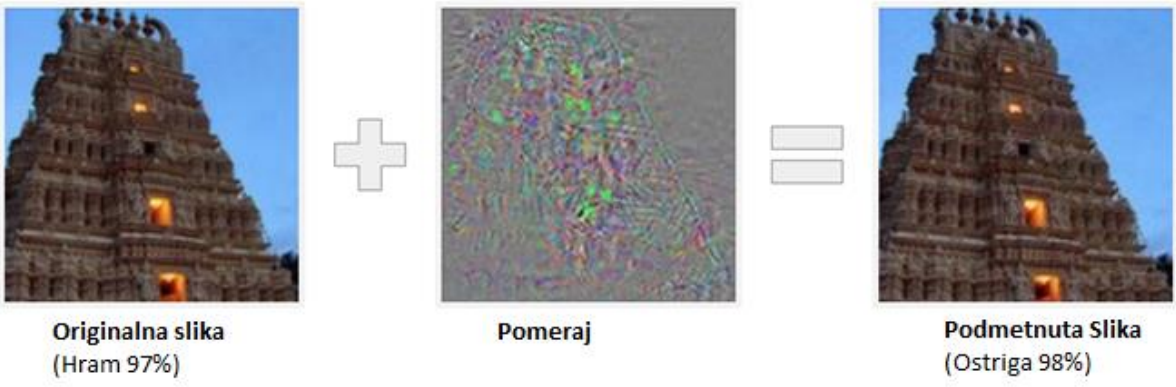
3.3.1. Konvolucione neuralne mreže

Neuralna mreža opisana iznad je sposobna da aproksimira bilo koju Borel-merljivu funkciju sa dovoljnim brojem skrivenih čvorova čak i sa jednim skrivenim slojem [15], međutim u praksi ove neuralne mreže sporo konvergiraju, ili se ne generalizuju dobro. Pošto ćemo se u daljem nastavku rada fokusirati na klasifikaciju slike, napomenućemo novu arhitekturu neuralne mreže koja se naziva Konvoluciona Neuralna Mreža [16].

U prvobitno opisanom modelu smo koristili isključivo potpuno povezane slojeve. To znači da je svaki neuron u sloju l povezan unapred sa svakim neuronom u sloju $l+1$, itd. Konvolucione mreže uvode 2 nova sloja, konvolucioni sloj, i agregacioni sloj. Konvolucioni sloj ima zadatak da nauči filtere koji kao rezultat konvolucije pronađu korisna svojstva slike, a agregacioni sloj kombinuje pronađena svojstva sa različitih delova slike kako bi klasifikacija bila invarijantna usled translacije objekta. Uvođenje ovih slojeva ne menja opisani algoritam propagacije unazad, samo se menjaju konkretni izvodi. Pošto ovi slojevi nisu bitni za ideju, i stoga prevazilaze opseg ovog rada, izostavićemo ih. Čitalac koji želi više da sazna o konvolucionim mrežama, može da pročita na linku http://cs231n.github.io/convolutional-networks/ , kao i u sledećim radovima [9], [12, Ch. 9], [16].

4. Napadi na modele dubokog učenja

Nakon što smo izložili teoretsku osnovu dubokih modela, možemo preći na glavnu ideju ovog rada, koja objašnjava kako zbuniti duboke modele da nam daju ulaz koji mi želimo. Glavna ideja kod napada koje ćemo izložiti jeste da modifikujemo ulaznu sliku u model na pametan način, tako da je model pogrešno klasifikuje, i da nam da željeni izlaz.



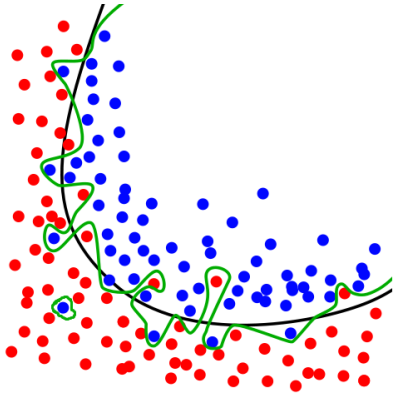
Ilustracija 11: Primer podmetnute slike koja kao rezultat klasifikacije daje ostrigu sa verovatnoćom 0.98. Originalna slika, i podmetnuta slika izgledaju identično za ljudsko oko.

Većina modela mašinskog učenja je podložna ovim napadima, ali modeli zasnovani na gradijentnom spustu su posebno podobni za ovakve napade zbog toga što ih je veoma lako izvesti kao što ćemo kasnije videti. Također, opisani napadi se mogu izvesti na razne probleme kao što su regresija, klasifikacija, ne-nadgledano učenje (eng. *Unsupervised Learning*, učenje sa pojačavanjem (eng. *Reinforcement Learning*). Mi ćemo se u nastavku rada fokusirati na klasifikaciju slika.

Slike su po svojoj prirodi (za razliku od tekstualnog ulaza) elementi topološkog skupa (tačnije manifolda) $I = [0,1]^{d*r}$ gde je r rezolucija slike, a d je dubina i uglavnom je 3 (RGB kanali). Ovo znači da ako bilo koji piksel pomerimo za neko malo $\boldsymbol{\epsilon} \in I$, također ćemo imati validnu sliku ukoliko slika $\boldsymbol{x} + \boldsymbol{\epsilon} \in I$. Ukoliko je rezolucija slike r velika, norma vektora $\|\boldsymbol{\epsilon}\|_2$ može da bude ogromna, iako je svaki element vektora ϵ mali.

Neka je $\|\boldsymbol{\epsilon}\|_2 = \sqrt{\sum_i \epsilon_i^2}$ L2 norma vektora ϵ. Ako fiksiramo ovu normu na $\|\boldsymbol{\epsilon}\|_2 = p \in \mathbb{R}$, i povećamo dimenzionalnost (rezoluciju), dopuštamo elementima vektora ϵ da budu manji, i tako razlike između $\boldsymbol{x}$, i $\boldsymbol{x} + \boldsymbol{\epsilon}$ budu manje primetne.
Uzmimo na primer da je $p = 1000$. Ako je rezolucija slike 1000 piksela (ukupan broj), tada da bi prešli L2 normu od 1000 jedinica, potrebno je da se pomaknemo u proseku $\frac{p}{1000}\sqrt{1000} \approx 31.62$ jedinica po svakoj osi. Ako je rezolucija slike 1 000 000 piksela, tada se u proseku moramo pomeriti $\frac{p}{1000000}\sqrt{1000000} = 1$ jedinicu po svakoj osi. Ovo svojstvo može da unese probleme u model kao što ćemo ubrzo videti, i istraživači u oblasti veštačke inteligencije ga nazivaju „prokletstvo dimenzionalnosti“.

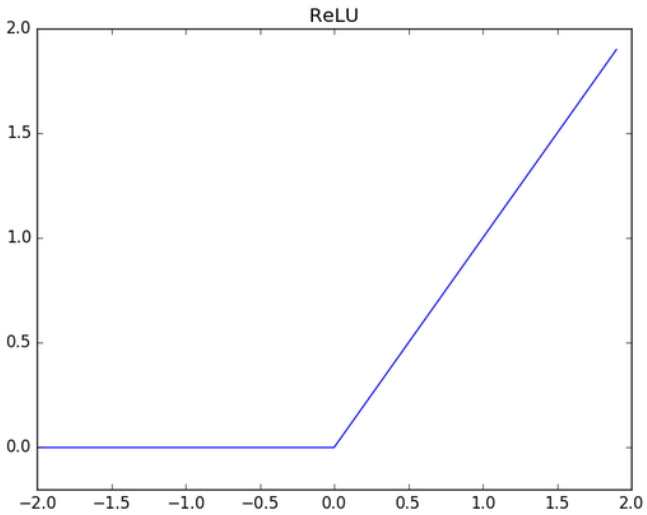
Još ranije [17], istraživači su došli na ideju da je moguće konstruisati ovakve primere koji će naterati model da proizvede željeni izlaz, ali su se uvek pitali zašto je to tako lako da se izvede kao što ćemo videti kasnije. Njihova prva hipoteza bila je da duboki modeli overfituju trening podatke (imaju veliku varijansu), i da se u prostoru slike I nalaze klasteri masa verovatnoće unutar kojih se možemo pomeriti i na taj način zbuniti model. Međutim, da je ova hipoteza tačna, tada bi svaka napadačka slika bila jedinstvena za model, ali i za svaku trening sesiju. Razlog tome je što ako model overfituje (ako mu je varijansa velika), pri svakom treniranju granica podele se znatno menja. Kao što se ispostavilo, to nije bio slučaj, i napadačke slike generisane jednom, mogu da se prenesu i da rade čak i kada se model ponovo istrenira, ali i kad se ta slika pusti kroz drugi sličan model (sa sličnom arhitekturom).



Ilustracija 12: Model je overfitovao podatke. U donjem desnog uglu vidimo jednu masu verovatnoće koja okružuje izolovani primerak plave boje.

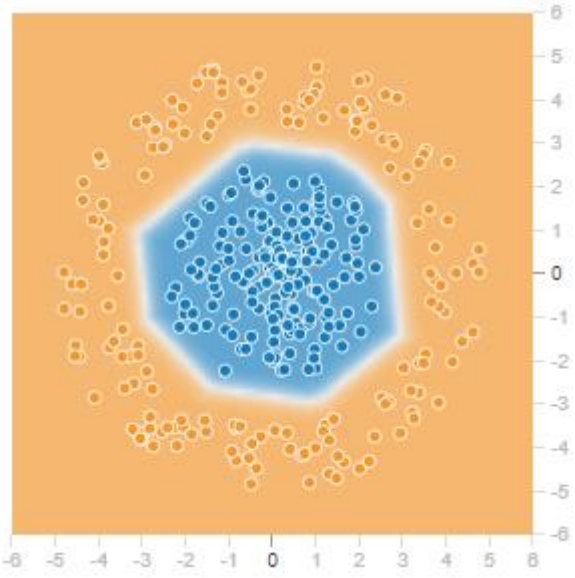
Međutim, problem je bio što su moderni modeli linearni u delovima. Razlog ove linearnosti jeste aktivaciona funkcija koja se koristi u modernim modelima, a to je ReLU (eng. *Rectifier Linear Unit*)

$$ReLU(x) = x^+ = \max(0, x)$$



ReLU funkcije su uvedene kako bi rešile problem gubitka gradijenta kod dubokih modela [13]. Za čitaoce koji žele više o ovome, mogu da posete link: https://sefiks.com/2018/05/31/an-overview-to-gradient-vanishing-problem/

Granica podele modela istreniranih sa ReLU aktivacionom funkcijom je linearna u delovima, kao što možemo videti na slici ispod.



Ilustracija 13: Na slici vidimo da iako je granica podele nelinearna, ona je linearna u delovima

Neuralna mreža aproksimira neprekidnu granicu podele (neprekidna funkcija, ili manifold) između klasa sa konačno mnogo trening podataka. Ovo znači da neuralna mreža ekstrapolira granicu podele skoro svuda (svuda osim u konačno mnogo tačaka), i to linearno. Ova ekstrapolacija se obično dešava oko teoretske granice (funkcija koja stvarno pravi podelu između klasa, i koju ne možemo da izmerimo). Kod visoko dimenzionalnih prostora kao što je prostor slike, teoretske granice su veoma kompleksne i oblasti klasa nisu koneksne (povezane). Ako se one linearno ekstrapoliraju u tačkama gde nemamo dovoljno podataka, imamo podlogu za napadačke tehnike koje ćemo opisati kasnije.

4.1. Tehnike izvođenja napada

Da bi izveli napad, iskoristićemo činjenicu da je skup slika topološki prostor, i da je preslikavanje iz slike u distribuciju klasa diferencijabilno, tj. $\boldsymbol{y}' = f(\boldsymbol{x}; \boldsymbol{\Theta})$ je diferencijabilna funkcija. Ovo znači da možemo optimizovati ulaz modela $\boldsymbol{x}$ tako da nam model vrati željenu vrednost vektora $\boldsymbol{y}'$.

Razlikujemo 2 vrste napada

- *ciljane napade* gde je ideja da nateramo model da nam izbaci određenu vrednost vektora $\boldsymbol{y}'$
- *obične napade* gde je ideja da nateramo model da vrati pogrešnu vrednost vektora $\boldsymbol{y}'$ koja god ona bila.

Napade dalje možemo grupisati u 3 grupe:

- *Analitičke napade:* Imamo potpuni pristup modelu. Tj. znamo arhitekturu modela, i imamo vrednosti svih parametara $\boldsymbol{\Theta}$**.** Ova dva podatka se nazivaju i modelujući graf mreže (eng. *Computational Graph*) **.** Ovakvi napadi su teoretski, ali služe kao osnova za ostale napade.
- *Polu analitički metod*: Nemamo potpuni pristup modelu, ali imamo pristup izlazu modela, tj. vektoru $\boldsymbol{y}'$**.**
- *Blackbox metod*: Nemamo pristup modelu, na izlazu dobijemo labelu kojoj klasi slika pripada.

Analitičke i polu analitičke metode izvodimo pomoću gradijenta modela zajedno sa *loss* funkcijom (slično kao pri treningu), s tim što sada ne uzimamo u obzir funkciju greške, nego direktno funkciju L.

Prilikom treninga smo definisali funkciju greške kao

$$C(\boldsymbol{\Theta}) = \frac{1}{|E|} \sum_{(\boldsymbol{x},\boldsymbol{y})\in \mathbf{E}} L(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\Theta})$$
$$L(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\Theta}) = -\sum_{i=1}^{m} y_i \log f(\boldsymbol{x}; \boldsymbol{\Theta})$$

Gde su x, y bili fiksirani, a optimizacija se radila nad funkcijom C i optimizovali su se parametri $\boldsymbol{\Theta}$**.** Da bi izveli napad, koristićemo sličnu ideju samo što ćemo funkciju greške definisati kao funkciju ulaza $\boldsymbol{x}$, a parametri $\boldsymbol{\Theta}$ će biti fiksirani.

$$C(\boldsymbol{x}, \boldsymbol{y}) = L(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\Theta}) = -\sum_{i=1}^{m} y_i \log f(\boldsymbol{x}; \boldsymbol{\Theta})$$

Da bi izveli običan napad koji nije ciljan, potrebno je da optimizujemo funkciju $-C(\boldsymbol{x}, \boldsymbol{y})$ spustom gradijenta. Dakle, za običan napad koristimo funkciju

$$J(\boldsymbol{x}) = -C(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{m} y_i \log f(\boldsymbol{x}; \boldsymbol{\Theta})$$

dok za ciljani napad koristimo funkciju:

$$J_t(\boldsymbol{x}) = C(\boldsymbol{x}, \hat{\boldsymbol{y}}) = -\sum_{i=1}^{m} \hat{y}_i \log f(\boldsymbol{x}; \boldsymbol{\Theta})$$

gde je $\hat{\boldsymbol{y}}$ željena vrednost koju želimo da nam model izbaci. U nastavku ćemo se fokusirati na ciljane napade.

Optimizaciju ćemo vršiti na isti način kao i kod treniranja, tj. spustom gradijenta.

$$\boldsymbol{x}_i' = \boldsymbol{x}_i' - \boldsymbol{\alpha} \boldsymbol{\nabla}_{\boldsymbol{x}_i} J_t$$
$$\boldsymbol{x}_0' = \boldsymbol{x}$$
$$\boldsymbol{\nabla}_{\boldsymbol{x}_i} J_t = \left(\frac{\partial J_t}{\partial (\boldsymbol{x}_i')_1}, \frac{\partial J_t}{\partial (\boldsymbol{x}_i')_2}, \dots, \frac{\partial J_t}{\partial (\boldsymbol{x}_i')_d} \right)$$

Razlika između analitičkog i polu analitičkog metoda jeste kako računamo ove izvode. Kod analitičkog metoda računamo ih propagacijom unazad kao pri treningu, dok kod polu analitičkog metoda radimo numeričku estimaciju gradijenta. Mi ćemo se fokusirati na analitički metod, a za polu analitički metod čitalac može sam da istraži tehnike za numeričku estimaciju gradijenta.

Izračunajmo $(\nabla_x C)_i = \frac{\partial C}{\partial (x)_i}$. Koristeći ranije izvedene jednačine, na sličan način možemo izvesti propagaciju unazad za napad:

$$\nabla C = \frac{\partial C}{\partial \boldsymbol{x}} = \boldsymbol{\delta}^1 (\boldsymbol{\Theta}^1)^{\mathbf{T}}$$

$$\boldsymbol{\delta}_{d+1} = (\boldsymbol{y}' - \hat{\boldsymbol{y}})$$
$$\boldsymbol{\delta}_l = \boldsymbol{\delta}_{l+1} (\boldsymbol{\Theta}^{l+1})^{\mathbf{T}} \circ f'_{hl}(\boldsymbol{h}_i^l)$$

$$\text{za } l = 1, 2, \dots, d$$

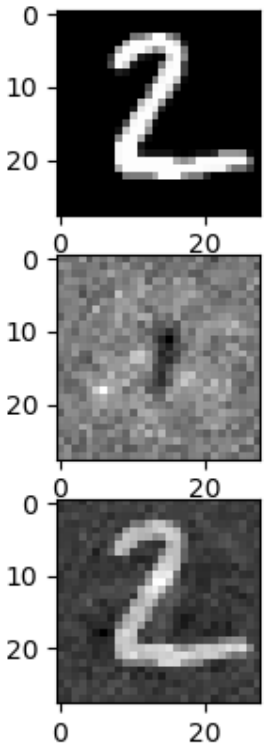
Za ciljane napade, $\nabla J_t = \nabla C$, dok je za obične napade $\nabla J = -\nabla C$.

Ispod je Pajton implementacija metode *run_adversarial* za našu klasu *NeuralNetwork* koja implementira jednačine iznad:

```
    def run_adversarial(self, x, y_target, num_iter=1, step_size=0.01):
        adversarial_x = x

        adv_perturbation = np.zeros(adversarial_x.shape)
        for _ in range(num_iter):
            y_prim = self.forward(adversarial_x)

            delta = (y_prim - y_target)
            for l in range(self.__d, 0, -1):
                delta = np.multiply(
                    np.dot(delta, self.__ThetaTensor[l+1].T),
                    activations.sigmoid(self.__H[l], True))
            adv_perturbation = np.dot(delta, self.__ThetaTensor[1].T)

            adversarial_x = adversarial_x - step_size*adv_perturbation
        return adversarial_x, adv_perturbation
```

Rezultat pokretanja ovog algoritma sa sledećim parametrima je ilustrovana ispod:

```
target = [0, 1, 0, 0, 0, 0, 0, 0, 0, 0] # Digit 1

model.run_adversarial(x, target, num_iter=7, step_size=0.1)
```



Ilustracija 14:(Gore) originalni ulaz u neuralnu mrežu klasifikovan kao cifra 2 sa verovatnoćom 0.97. (Sredina) šum koji dodajemo na ulaznu sliku. (Dole) zbir ulaza i šuma koga mreža klasifikuje kao cifru 1 sa verovatnoćom 0.77

U primeru iznad ipak vidimo razliku između ulaza i zbira ulaza i šuma. Postoje 2 razloga zašto je to tako. Prvi razlog jeste što je u primeru iznad korišćen MNIST skup za treniranje koji sadrži slike 28*28 piksela. To nisu slike velike dimenzionalnosti pa ovaj skup i naš model sa kojim smo ga istrenirali nije podložan „prokletstvu dimenzionalnosti“ koga smo opisali ranije.

L2 norma vektora šuma $\|\epsilon\|_2 \approx 0.928336$, maksimalan element apsolutne vrednosti gradijenta je 0.16779047, dok je prosečan pomeraj sa ovim šumom (prosečna vrednost apsolutne vrednosti gradijenta) je 0.02517

Drugi problem jeste što naša funkcija optimizacije ne uzima u obzir koliko se slika promenila tokom optimizacije. Ovaj problem rešavamo tako što u funkciju optimizacije uvodimo regularizaciju, tj. kažemo algoritmu da ujedno pri optimizaciji kroz prostor slika I

kreće tako da se što više približava našem ciljnom vektoru $\hat{\boldsymbol{y}}$, dok ujedno minimizuje L_p normu $\|\boldsymbol{\epsilon}\|_p$ slike, i tako je drži što bliže originalnoj slici. Ukoliko ulaz ima dovoljno veliku rezoluciju, ovaj metod nas vodi do rešenja koje za čoveka izgleda isto kao i originalna slika.

Reformulisaćemo našu funkciju optimizacije tako da sadrži i regularizaciju norme šuma:

$$J_{tr}(\boldsymbol{x};\gamma,p) = C(\boldsymbol{x},\hat{\boldsymbol{y}}) + \gamma\|\boldsymbol{\nabla} J_t\|_p$$

$$\|\boldsymbol{x}\|_p = \left(\sum_{i=1}^{n}|x_i|^p\right)^{\frac{1}{p}}$$

$$\|\boldsymbol{x}\|_0 = |\{|x_i| \neq 0,\ i = 1,\ldots n\}|$$

$$\|\boldsymbol{x}\|_\infty = \max(|x_i|)$$

Prilikom optimizacije ove funkcije računali bi

$$\boldsymbol{\nabla} J_{tr} = \boldsymbol{\nabla} C + \gamma \frac{\sum_i (\boldsymbol{\nabla} J_{tr})_i}{\|\boldsymbol{\nabla} J_{tr}\|_p}\ \boldsymbol{\nabla}^2 J_{tr}$$

Ovaj problem minimizacije se može rešiti LBFGS algoritmom [17].

Norma p utiče na to koliko smo osetljivi na ekstreme. Što je parametar p veći to više tera algoritam da raširi pomeraje u šumu preko čitave slike, i više smo osetljivi na ekstreme. Sa normom 0 možemo da vidimo manji broj piksela koji se pomeraju, ali njihove vrednosti mogu biti velike. Sa 0 normom, možemo videti promene u slici samo u malom broju piksela ili nekoj regiji slike, dok sa ∞ normom možemo zapaziti promene preko čitave slike, ali će one biti veoma male. U praksi se koristi norma ∞.

Algoritam koji ćemo sada izložiti kombinuje normu ∞ i aproksimira $\boldsymbol{\nabla} J_t$. Brza metoda znaka gradijenta koristi znak gradijenta i intenzitet pomeraja ϵ:

$$\boldsymbol{x}' = \boldsymbol{x} - \epsilon * \text{sign}(\boldsymbol{\nabla} J_t)$$

Ovaj pomeraj će da ima $\|\boldsymbol{x}' - \boldsymbol{x}\|_\infty = \epsilon$, i predstavlja aproksimaciju pomeraja $\boldsymbol{\nabla} J_{tr}$.
Uvođenjem momentuma [18] možemo da poboljšamo ovu metodu i da ubrzamo konvergenciju:

$$\boldsymbol{g}_{t+1} = \mu \boldsymbol{g}_t + \frac{\boldsymbol{\nabla} J_t}{\|\boldsymbol{\nabla} J_t\|_1}$$

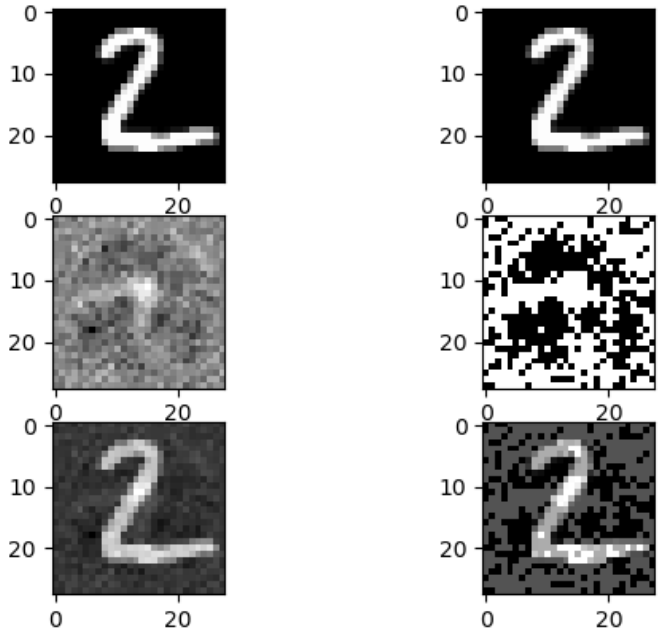
$$\boldsymbol{x}'_{t+1} = \boldsymbol{x}'_t + \epsilon * \text{sign}(\boldsymbol{g}_{t+1})$$

$$\boldsymbol{g}_0 = 0$$
$$\boldsymbol{x}'_0 = \boldsymbol{x}$$

Ispod je primer koda koji implementira iterativni metod sa momentumom

```
def run_ifgsm(self, x, y_target, eps=1, step_size=1, momentum=0, num_iter=1):
    r'''Iterative FGSM.
    x: Input image
    y_target: Optional target value
    eps: L_{\infty} norm of the perturbation
    num_iter: Number of iterations
    '''

    clip_min =  x - eps
    clip_max = x + eps

    adversarial_x = x
    g = 0
    for _ in range(num_iter):
        _, gradient = self.run_adversarial(x, y_target)
        grad_1norm = np.linalg.norm(gradient, 1)

        g = momentum*g + (gradient/grad_1norm)

        signed_grad = np.sign(g) * step_size
        adversarial_x = np.clip(adversarial_x + signed_grad, clip_min, clip_max)
    adv_perturbation = adversarial_x - x

    return adversarial_x, adv_perturbation
```

Iterativni metod znaka gradijenta ne radi dobro na slikama malih dimenzija jer ne može da iskoristi „prokletstvo dimenzionalnosti“ slika kao što vidimo na primeru ispod, ali se na realnim skupovima kao što su *ImageNet* i CIFAR-10 pokazuje kao bolje rešenje.

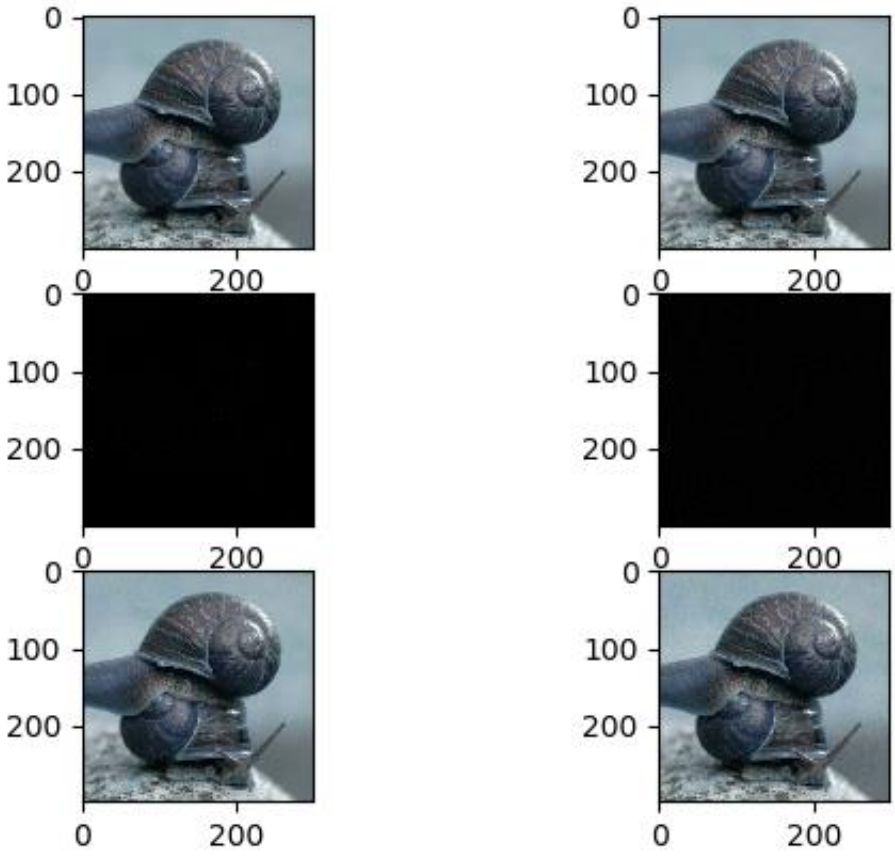


Ilustracija 15: (Leva polovina) originalna slika, šum, i na dnu rezultat spusta gradijentom. (Desna polovina) originalna slika, šum, i rezultat iterativnog metoda znaka gradijenta.

Tabela ispod prikazuje efikasnost napada na MNIST klasifikator. Originalna slika na vrhu je ulaz i model je klasifikuje kao cifru 2 sa verovatnoćom 0.97. Izlaz je modifikovana slika koja je ciljana od strane napadača da se klasifikuje kao cifra 1. Iz tabele vidimo da metod vrednosti gradijenta ima manju L_2 normu od metoda znaka gradijenta, ali i veću L_∞ normu.

	Napad spustom gradijenta. Parametri: step_size(0.1) num_iter(7)	Iterativni metod znaka gradijenta. Parametri: step_size(0.1) num_iter(7) eps(0.24) momentum(0.2)
Izlaz modela nakon napada	Klasifikovano kao 1 sa verovatnoćom 0.5	Klasifikovano kao 1 sa verovatnoćom 0.52
L_2 norma šuma	2.20714048677303	6.719999849397163
L_∞ norma šuma	0.331258678351833	0.24000000953674316
Srednja apsolutna vrednost piksela šuma	0.060804051069489815	0.23999999462132704

Na slikama veće dimenzionalnosti, model se ponaša potpuno drugačije. Ispod je rezultat primene tehnike vrednosti gradijenta kao i znaka gradijenta na *Inception v3* [19] modelu i slikama dimenzije (299, 299, 3).



Ilustracija 16: (Levo) originalna slika, šum, i rezultat na dnu posle primene metoda vrednosti gradijenta. (Desno) originalna slika, šum, i rezultat posle primene metode znaka gradijenta sa momentumom.

Tabela ispod prikazuje efikasnost napada na *ImageNet* klasifikator (*mobilenet* neuralna mreža). Originalna slika na vrhu je ulaz i model je klasifikuje kao „puž“ sa verovatnoćom 0.99. Izlaz je modifikovana slika koja je ciljana od strane napadača da se klasifikuje kao zmaj „kite“.

Možemo videti da je u proseku srednja vrednost šuma kod metoda vrednosti gradijenta manja, ali je maksimalna vrednost veća. Ovo je očekivano ponašanje pošto metoda znaka gradijenta optimizuje i vrši regularizaciju u L_∞ normi, pa samim tim model raspoređuje pomeraje šuma na sve piksele. Također možemo primetiti kako za sličnu vrednost L_2 norme, ostali parametri su znatno manji nego za primere iznad sa MNIST podacima. Razlog tome jeste dimenzionalnost slika.

Originalna slika klasifikovana kao puž sa verovatnoćom 0.99	Napad spustom gradijenta. Parametri: step_size(0.1) num_iter(5)	Iterativni metod znaka gradijenta. Parametri: step_size(0.05) num_iter(5) eps(0.016) momentum(0.8)
Izlaz modela nakon napada	Klasifikovano kao zmaj („kite“) sa verovatnoćom 0.99	Klasifikovano kao zmaj („kite“) sa verovatnoćom 0.99
L_2 norma šuma	4.2268567	8.286149
L_∞ norma šuma	0.085820615	0.016000003
Srednja apsolutna vrednost piksela šuma	0.0056709372	0.015999993

4.2. Tehnike odbrane

Odbrana od napada opisanih u ovom radu još uvek je otvoreno istraživačko pitanje, i sve poznate metode ne garantuju sigurnost. Ipak, metod koji je najkorisniji jeste suparničko treniranje. Ideja je da istreniramo model nad nekim skupom, zatim nad tim skupom generišemo protivničke slike kao što smo ranije opisali i te slike ubacimo u novi skup za trening. Na kraju ponovo istreniramo model sa novim skupom. Iako ova tehnika ne unosi sigurnost u model, veoma je dobra za generalizaciju modela (smanjenje njegove varijanse). Napadač koji ima više procesorske moći može jednostavno da ponovi sve navedene tehnike više puta i napadne model sa jačim napadima. Pored suparničkog treniranja postoje i razne druge tehnike koje su također neuspešne [20], [21].

Postavlja se pitanje zašto je odbrana od ovih modela toliko teška. Naime, od modela se zahteva da daju dobar izlaz na svaki mogući ulaz, ali se za njihovo treniranje koristi konačno mnogo podataka. Dalje, probabilistički modeli uče da daju izlaz tako što aproksimiraju distribuciju podataka koje su videli tokom treninga, dakle, distribuciju stvarnih slika koje se javljaju u prirodi. Kada modelu prikažemo sliku sa dodatnim šumom koji nije beli šum, menja se distribucija i narušavaju se pretpostavke koje je model uspostavio tokom treninga. Drugi problem je nedostatak teoretskih modela koji bi opisali šta se tačno dešava nakon optimizacije. Ovakvi modeli su neophodni da bi mogli da konstruišemo argument i

odgovorimo na pitanje „šta je suparnički primer?“. Bez ovakvih modela nismo u stanju da utvrdimo da li bilo koja tehnika odbrane daje rešenje ili ne.

Primer koji najbolje ilustruje zašto se ovakvi napadi mogu lako konstruisati jeste primer „Pametnog Hansa“. Pametni Hans je bio konj u doba renesanse koji je navodno znao osnovnu aritmetiku, tj. da sabira brojeve. Na glavnom trgu grada bi se okupili svi građani i zadavali konju aritmetičke zadatke. Jedan građanin bi rekao „Hans, koliko je 2+3?“, na šta bi Hans udario kopitom od tlo 5 puta i u tom trenutku bi publika počela da aplaudira.

Jedan psiholog nije mogao da veruje šta se dešava pa je uzeo Hansa da ga testira. Kada ga je doveo u svoje prostorije, i izolovao od svega, Hans nije više mogao da radi aritmetiku. Ispostavilo se da Hans nije znao da radi aritmetiku nego je posle svakog pitanja tapkao dok ne čuje aplauze publike. Ni Hans ni njegov pastor nisu bili prevaranti, nego je Hans naučio da rešava problem u okruženju u kome je treniran i u kome je stalno testiran. To okruženje je bilo ono gde je Hans bio okružen publikom koja bi počela da aplaudira kada bi Hans udario određen broj puta kopitom o tlo. Kada je Hans izolovan iz takvog okruženja, nije više mogao da obavlja zadatak.

5. Zaključak

Duboki modeli leže u srcu moderne veštačke inteligencije, i možemo ih sresti skoro u bilo kom modernom proizvodu. Nalaze se u pametnim telefonima kako bi poboljšali kameru, i prepoznali vlasnika, u računarima kako bi prepoznali govor. Također, razni servisi na internetu koriste duboke modele kako bi pružili što prijatnije iskustvo za korisnika. Odgovarajuće mere sigurnosti su potrebne kako korisnik modela ne bi pretrpeo štetu nastalu usled malicioznog napada. Kao što smo ranije pokazali, ove napade je veoma lako izvesti, pa čak i onda kada nemamo direktan pristup modelu tako što numerički procenimo gradijente. Ipak prava opasnost leži od toga što je moguće konstruisati primer za jedan model (neki od popularnih modela koje možemo da treniramo kući), i preneti ga na sličan model koji radi u pozadini servisa [22].

Sve navedene ranjivosti dubokih modela treba da nam ukažu da je sigurnost veštačke inteligencije ozbiljan i nerešen problem, te je potrebno ozbiljno se pozabaviti istim pre nego što čovečanstvo napravi sledeći korak prema opštoj veštačkoj inteligenciji. Iako trenutno ne postoji efikasna odbrana od tehnika napada izloženih u ovom radu, ipak nije sve izgubljeno. Napadi se znatno mogu otežati ako se onemogući napadaču da vidi verovatnoće koje model izbacuje, kao i ako se ograniči broj upita ka servisu koji pruža usluge veštačke inteligencije. Ove metode u ovom momentu mogu da otežaju znatno napade, ali ni one nisu sasvim otporne. Također je bitno napomenuti da su probabilistički modeli za klasifikaciju slika najviše ranjivi jer su slike topološki prostori sa velikom dimenzionalnošću. Dakle, redukcija rezolucije može također da se koristi kao tehnika odbrane.

Od tehnika koje smo naveli, metoda znaka gradijenta daje bolje rezultate za fiksnu jedinicu vremena nego metoda vrednosti gradijenta jer brže konvergira i uzima u obzir i regularizaciju. Obe metode mogu da budu moćno oružje u rukama napadača, i mogu da naštete sistemima koji su kritični kao što su automobili, industrija, i slično. Trenutno su ovi modeli dobro zaštićeni i još uvek je teže napasti takve sisteme u praksi, ali sa vremenom će to postajati lakše. Iako ovaj problem nije prioritetan, da bi uveli veštačku inteligenciju u svakodnevan život, i prepustili joj da obavlja kritične zadatke, pitanje odbrane od ovakvih napada sigurno treba da se odgovori.

6. Reference

[1] S. Hoehl, K. Hellmer, M. Johansson, and G. Gredebäck, "Itsy Bitsy Spider…: Infants React with Increased Arousal to Spiders and Snakes," *Front. Psychol.*, vol. 8, Oct. 2017.

[2] F. T. Bruss, "250 years of 'An essay towards solving a problem in the doctrine of chances. By the late Rev. Mr. Bayes, F.R.S. communicated by Mr. Price, in a letter to John Canton, A.M.F.R.S.,'" 2014.

[3] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bull. Math. Biophys.*, vol. 5, no. 4, pp. 115–133, Dec. 1943.

[4] A. M. TURING, "I.—COMPUTING MACHINERY AND INTELLIGENCE," *Mind*, vol. LIX, no. 236, pp. 433–460, Oct. 1950.

[5] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, Oct. 1986.

[6] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities.," *Proc. Natl. Acad. Sci. U. S. A.*, vol. 79, no. 8, pp. 2554–8, Apr. 1982.

[7] Y. Yann, Y. Yann, L. Le, C. Cun, and S. Trainable, "Yann LeCunThe The Challenges of of Machine Learning."

[8] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks."

[10] R. DiPietro, "A Friendly Introduction to Cross-Entropy Loss." [Online]. Available: https://rdipietro.github.io/friendly-intro-to-cross-entropy-loss/. [Accessed: 23-Sep-2018].

[11] D. J. C. MacKay, *Information theory, inference, and learning algorithms*. Cambridge University Press, 2003.

[12] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. .

[13] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for Activation Functions," Oct. 2017.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," Feb. 2015.

[15] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, no. 2, pp. 251–257, Jan. 1991.

[16] Y. LeCunn and Y. Bengio, "Convolutional Networks for Images, Speech, and Time'Series," 1995.

[17] C. Szegedy *et al.*, "Intriguing properties of neural networks."

[18] Y. Dong *et al.*, "Boosting Adversarial Attacks with Momentum," Oct. 2017.

[19] Z. Alom, M. Hasan, C. Yakopcic, T. M. Taha, and V. K. Asari, "Improved Inception-Residual Convolutional Neural Network for Object Recognition."

[20] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, “Distillation as a Defense to Adversarial Perturbations against Deep Neural Networks,” Nov. 2015.

[21] N. Papernot, P. McDaniel, A. Sinha, and M. Wellman, “Towards the Science of Security and Privacy in Machine Learning,” Nov. 2016.

[22] F. Tramèr, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel, “The Space of Transferable Adversarial Examples,” Apr. 2017.