SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1966

Optimizirane izlazne funkcije klasifikatora temeljenog na umjetnim neuronskim mrežama u domeni implementacijskih napada na kriptografske uređaje

Juraj Fulir

Zagreb, lipanj 2019.

Umjesto ove stranice umetnite izvornik Vašeg rada.

Da bi ste uklonili ovu stranicu obrišite naredbu `\izvornik`.

ZAHVALA'n'STUFF

SADRŽAJ

1. Uvod

TODO: *Opis problema*

2. Implementacijski napadi na kriptografske uređaje

2.1. Side-channel napadi

TODO: *Postoji nekoliko vrsta.*

TODO: *Ovdje se obrađuje DPA.*

2.2. Izvedba napada

TODO: *Uštekaj uređaj, osciloskop na to i to mjesto i snimaj*

TODO: *Provjeri mogućnosti i zaključi najvjerojatniju*

TODO: *Problem netraktabilnosti postupka -> neuralke <3*

2.3. DPA skupovi podataka

TODO: *Tko i cilj**

TODO: *Ne zaboravi referencu na stranicu!*

2.3.1. DPAv2

TODO: *Kada je napravljen i ko ga je radil*

TODO: *Jel HW ili onaj pravi*

[IMAGE: *PCA redukcija iz jn*]

[IMAGE: *Statistike iz jn*]

TODO: *Mjere dobrote klasifikacije*

2.3.2. DPAv4

TODO: *Kada je napravljen i ko ga je radil*
TODO: *Jel HW ili onaj pravi*

[IMAGE: *PCA redukcija iz jn*]

[IMAGE: *Statistike iz jn*]

TODO: *Mjere dobrote klasifikacije*

3. Klasifikator temeljen na umjetnim neuronskim mrežama

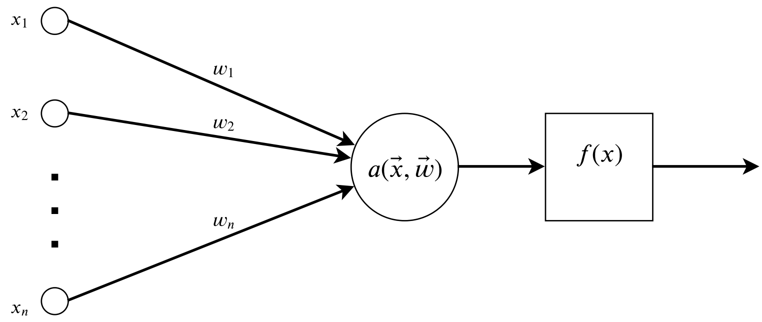
3.1. Umjetne neuronske mreže

Umjetne neuronske mreže (nadalje „neuronske mreže") koristimo za modeliranje više-dimenzijske funkcije ili distribucije kojom se aproksimira rješenje zadanog problema iz konačnog broja primjera. Vrlo su moćan alat za savladavanje teških zadataka u raznim područjima, često dostižući ljudske performanse na zadanom problemu. Danas su vrlo raširene u raznim područjima od kojih su samo neka: računalni vid (Krizhevsky et al., 2012; Redmon et al., 2016), prirodna obrada jezika (Mikolov et al., 2013; Kim, 2014) i podržano učenje (Mnih et al., 2013; Fang et al., 2017).

3.1.1. Građa

Neuronske mreže građene su od međusobno povezanih jedinica, tzv. neurona, modeliranih prema pojednostavljenom modelu biološkog neurona. Neuron očitava ulazne značajke sustava ili izlaze drugih neurona te ažurira svoje unutarnje stanje i stvara odziv. Utjecaj ulaza na neuron vrednuje se težinama (engl. *weights*) koje definiraju kako se neuron ponaša u ovisnosti o pojedinim ulazima. Aktivacijski prag neurona (engl. *bias*) određuje jedinstvenu osjetljivost neurona na jačinu podražaja. Težine i prag neurona nazivamo parametrima neurona.
TODO: *Što sve biolozi vele o neuronima? https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3812748/*

Način na koji iz ulaza gradimo unutarnje stanje neurona opisujemo ulaznom funkcijom. Pretvorbu unutarnjeg stanja neurona u izlazni signal opisujemo izlaznom funkcijom koja se detaljnije obrađuje u poglavlju 4. Ulazna i izlazna funkcija definiraju prijenosnu funkciju koja ujedno opisuje ponašanje cijelog neurona. Pojam prijenosne funkcije je potreban jer ih ponekad ne možemo podijeliti na ulazne i izlazne. Izlaznu



Slika 3.1: Prikazani osnovni dijelovi neurona su težine dendrita (w_i), ulazna funkcija ($a(\vec{x}, \vec{w})$) i izlazna funkcija ($f(x)$). Prag neurona nije prikazan zbog jednostavnosti dijagrama.

funkciju ne treba miješati s funkcijom izlaza mreže koja predstavlja nelinearnost posljednjeg sloja mreže. U literaturi se umjesto pojma izlazne funkcije vrlo često koristi pojam aktivacijske funkcije. No u preglednim znanstvenim radovima (Duch i Jankowski, 1999, 2000, 2001) pojam aktivacijske funkcije predstavlja ulaznu funkciju, što je neispravno sa stajališta teorijske neuroznanosti jer aktivacija neurona označava pojavu odaslanog signala iz some u akson (Dayan i Abbott, 2005, p. 234). U starijoj literaturi je pojam aktivacijske i prijenosne funkcije često ekvivalentno korišten. S obzirom na nekonzistencije u literaturi i iz potrebe razlikovanja funkcija koje prikupljaju i odašilju signale iz neurona u ovom se radu koristi prvo navedena nomenklatura (ulazna, izlazna i prijenosna funkcija).

$$t(x) = (f \circ a)(x) = f(a(x)) \tag{3.1}$$

Najpopularnije ulazne funkcije jesu afina funkcija i unakrsna korelacija. Afina funkcija je skalarni produkt vektora ulaza s vektorom težina neurona uz dodatak vrijednosti praga. Parametri neurona definiraju nagib i pomak ravnine u prostoru ulaza koja opisuje ulaz neurona. Primijenjuje se kada se ulazi u model mogu zapisati vektorom značajki čiji raspored nije bitan.

$$f(\vec{x}; \underline{W}, \vec{b}) = \underline{W}^T \cdot \vec{x} + \vec{b} \tag{3.2}$$

Unakrsna korelacija, za razliku od afine funkcije, koristi informaciju o susjednosti ulaznih značajki. Ulaz za takav model je definiran n-dimenzijskim tenzorom, a neuron uzima samo jednu regiju tenzora (vidljivu regiju) i nad njime računa skalarni produkt s n-dimenzijskim tenzorom parametara (jezgrom). Kada su ulazi slike u boji ulazni tenzor ima 3 dimenzije (visina, širina i RGB kanali) pa stoga i svaka jezgra ima 3 dimenzije, no znantno manje visine i širine. Unakrsna korelacija prozvana je konvolucijom jer radi na istom principu, a jedina razlika je da se elementi jezgre indeksiraju zrcaljeno po obje osi. S obzirom da se parametri jezgre uče automatski, nije nam bitno definirati orijentaciju jezgre. Unakrsna korelacija koristi iste parametre za svaku vidljivu regiju što ju čini štedljivijom od afine funkcije te ostvaruje neosjetljivost na translaciju, što je vrlo korisno u računalnom vidu.

$$f(\underline{X}; \underline{W}) = \underline{X} \circledast \underline{W} \tag{3.3}$$

TODO: *Je li uopće potrebno spominjat konvoluciju?*

Postoje i aktivacijske funkcije temeljene na udaljenosti vektora ...

TODO: *Spomeni distance based aktivacije (ANFIS?)*

TODO: *Spomeni i složenije metode: (Lin et al., 2014)*

Povezivanjem neurona gradi se arhitektura mreže koja određuje kako podatci i gradijenti teku kroz mrežu, a time utječu na brzinu učenja i inferencije neuronske mreže. Najčešće se koriste slojevite unaprijedne arhitekture zbog jednostavnosti izvedbe. Unaprijedne arhitekture propuštaju podatke samo u jednom smjeru odnosno već izračunati neuroni se ne izračunavaju ponovno, što je posebno pogodno za optimizaciju širenjem unatrag, detaljnije opisanu u poglavlju 3.1.2. Slojevite arhitekture omogućuju paralelizaciju izvođenja operacija na grafičkim karticama što značajno ubrzava postupke učenja i inferencije. Pri definiciji slojevite arhitekture najčešće je dovoljno navesti samo redoslijed slojeva, no ponekad je potrebno definirati i način povezivanja slojeva npr. pri uporabi preskočnih veza (Srivastava et al., 2015; He et al., 2016; Huang et al., 2017). Prvi sloj služi za postavljanje ulaza mreže i nazivamo ga ulaznim slojem mreže. Posljednji sloj mreže služi nam za ekstrakciju izlaza te mjerenje kakvoće mreže i nazivamo ga izlaznim slojem mreže. Svi slojevi između ulaznog i izlaznog sloja nazivaju se skrivenim slojevima.

Potpuno povezana arhitektura je najjednostavnija arhitektura za zadatak klasifikacije. Svaki neuron u potpuno povezanom sloju aktivira se pomoću svih izlaza iz prethodnog sloja. Za naučeni potpuno povezani sloj kažemo da vrši ekstrakciju značajki iz svojih ulaza. Geometrijski gledano, svaki neuron vrši mapiranje značajki iz dimenzije prethodnog sloja u novu dimenziju s ciljem modeliranja boljih značajki.

TODO: *Daj neki dokaz za ovo gore.*

3.1.2. Optimizacija umjetne neuronske mreže

Optimizacijom parametara neuronska mreža prilagođava se danom zadatku, odnosno kažemo da mreža 'uči'. Optimizaciju parametara najčešće izvodimo gradijentnim spustom, uz pretpostavku derivabilnosti svih komponenata neuronske mreže. Kada ta pretpostavka ne vrijedi koriste se algoritmi kombinatorne optimizacije poput algoritma roja čestica koji se spominje u Čupić et al. (2013). U ovom radu neuronske mreže optimiraju se gradijentnim spustom.

Gradijentni spust

[IMAGE: *gradijentni spust unimodalna vs višemodalna (gdje preskoći brdo i uleti u bolji optimum)*]

Gradijentni spust je algoritam pronalaska minimuma funkcije vođen gradijentom te funkcije. Za zadanu početnu točku iterativno se pomiče u smjeru suprotnom od gradijenta funkcije u toj točki dok ne zadovolji neki od uvijeta zaustavljanja. Na strmim funkcijama gradijent je često prevelik i može izazvati oscilaciju ili divergenciju (slika 3.1.2). Stoga se gradijent pri pomaku skalira koeficijentom pomaka μ. Dobro odabran koeficijent pomaka može osigurati bržu konvergenciju, a kod višemodalnih funkcija i pronalazak boljeg optimuma (slika 3.1.2).

Početna točka utječe na ishod algoritma. Kod višemodalnih funkcija s optimumima različitih kvaliteta, početna točka može definirati u koji će lokalni optimum algoritam konvergirati (slika 3.1.2).

Input: funkcija $f(\vec{x})$
Input: početna točka $\vec{x_0}$
Input: koeficijent pomaka η
Input: broj iteracija n
for *n iteracija* **do**
    $\vec{g_i} \leftarrow \vec{\nabla}_{\vec{x}} f(\vec{x_i})$;
    $\vec{x_{i+1}} \leftarrow \vec{x_i} - \eta \cdot \vec{g_i}$
end
Result: konačna točka $\vec{x_i}$ je pronađeni optimum

Algorithm 1: Gradijentni spust

[IMAGE: *graidjentni spust stope spuštanja (velka, mala, taman)*]

Broj iteracija definira koliko se puta pomičemo iz početne točke, što definira i trajanje algoritma. Generalno želimo skratiti vrijeme pretraživanja te povećati koeficijent spusta kako bismo koristili manje pomaka. No u praksi najčešće nailazimo na višemodalne funkcije sa strmim regijama koje izazivaju oscilacije i mogu izazvati divergenciju. Stoga se češće koriste manji pomaci kroz više iteracija. Dodatno se mogu dodati modifikacije gradijenta koje nude ograničavaju veličinu gradijenta (odsijecanje gradijenta i sl.).

[IMAGE: *gradijentni spust sa početnim točkama (jedna ode u lok, jedna ode u glob, jedna zapne desno na platou)*]

Problem se javlja ako algoritam odluta u visoravan na kojoj su gradijenti vrlo mali, a sama regija je s obzirom na pomake ogromna (slika 3.1.2). Kad gradijent postane neupotrebljivo malen kažemo da je *iščeznuo*. U takvim slučajevima pomaže dodavanje momenta koji se akumulira kroz više iteracija i dodaje vektoru gradijenta. Kad algoritam naiđe na regiju s vrlo malim gradijentima, moment pokušava izvući algoritam iz visoravni pomičući ga u smjeru koji je akumuliran. Kako moment ne bi izvukao algoritam iz optimuma, dodaje mu se koeficijent *zaboravljanja* kojim se stari vektor momenta djelomično zaboravlja u korist novog vektora pomaka (jednadžba 3.4). Moment može pomoći i pri zaobilaženju lokalnih optimuma (slika 3.1.2).

$$\begin{aligned} \vec{v} &\leftarrow \alpha \cdot \vec{v} - \eta \cdot \vec{g} \\ \vec{x} &\leftarrow \vec{x} + \vec{v} \end{aligned} \tag{3.4}$$

[IMAGE: *moment prije i na visoravni, moment za savladavanje brda*]

Algoritam je primijenjiv na funkcije proizvoljne dimenzionalnosti uz pretpostavku derivabilnosti u svakoj točki. Za proizvoljnu realnu funkciju, uz dobro odabrane hiperparametre, algoritam će konvergirati u jedan od lokalnih optimuma, no algoritam generalno nema garanciju konvergencije u globalni optimum. Garanciju pronalaska globalnog optimuma nudi jedino za trivijalne unimodalne funkcije uz odgovarajuće hiperparametre algoritma (slika 3.1.2).

Problem odabira hiperparametara proizlazi iz činjenice da u generalno praksi nemamo definiranu funkciju koju minimiziramo (već samo skup primjera te funkcije) i/ili ju ne možemo jasno vizualizirati (kada radimo s funkcijama visoke dimenzionalnosti). Čak i da imamo definiranu funkciju najčešće ne znamo koju vrijednost poprima globalni optimum, a pohlepna pretraga je netraktabilna. Unatoč tome, gradijentni spust efikasno i učinkovito pronalazi optimume koji su dovoljno dobri za većinu praktičnih primjena (Redmon et al., 2016).

Funkcija gubitka

Pri učenju umjetnih neuronskih mreža potrebno je definirati funkciju gubitka. Funkcija gubitka, za dani ulaz, uspoređuje predikciju mreže sa željenim vrijednostima te dodjeljuje iznos pogreške (realan broj). Potrebno je pažljivo odabrati funkciju gubitka jer utječe na učenje svakog parametra (kao što je opisano u poglavlju 3.1.2) te definira što je ishod učenja.

Najčešće ne znamo definiciju funkcije gubitka na čitavoj promatranoj domeni već posjedujemo samo prijere te funkcije u podatcima koje smo izmjerili i koje smatramo reprezentativnim. Ovdje pretpostavljamo da će neuronska mreža ostvariti svojstvo generalizacije, koje je detaljnije objašnjeno u poglavlju 3.1.5. Stoga se u narednim formulama koristi notacija sumiranja.

Funkcija gubitka često je usko vezana uz vrstu problema koji se rješava (klasifikacija, regresija i ostali), način učenja (nadzirano, polu-nadzirano, nenadzirano, podržano) i izlaznu funkciju posljednjeg sloja neuronske mreže. U ovom radu vrši se klasifikacija nadziranim učenjem, no u nastavku se navodi i primjer funkcije gubitka za regresiju. Funkciju gubitka definiramo kao inverz funkcije izglednosti da neuronska mreža modelira funkciju predstavljenu primjerima podatkovnog skupa. Funkciju gubitka minimiziramo pa je definirana **negativnim logaritmom izglednosti** (3.5). Logaritam je uveden zbog pojednostavljivanja distribucija koje sadrže eksponente.

$$L(\theta \mid x, y) = -\log\, p(f(X;\theta) = y \mid X = x) \tag{3.5}$$

Zbog jednostavnosti zapisa, funkciju modela zapisujemo skraćeno kao aproksimaciju odnosno predikciju funkcije izlaza podatkovnog skupa te se ovisnost o ulazima podrazumijeva.

$$\hat{y} = f(x;\theta) \tag{3.6}$$

S obzirom da želimo procijeniti kvalitetu modela na čitavom podatkovnom skupu zanima nas očekivanje funkcije gubitka (3.23).

$$\begin{aligned} L(\theta \mid \mathcal{D}) &= \mathbb{E}_{(x,y)\sim p_{\mathcal{D}}}[L(\theta \mid x, y)] \\ &= -\frac{1}{|\mathcal{D}|} \sum_{(x,y)\in\mathcal{D}} \log\, p(\hat{y} = y \mid x) \end{aligned} \tag{3.7}$$

U problemima regresije, pretpostavlja se da izlazi mreže prate Gaussovu distribuciju s jediničnom kovarijacijskom matricom. Njenim uvrštavanjem u negativnu log. izglednost (3.5) dobivamo funkciju kvadratnog gubitka koja računa odstupanje izlaza neuronske mreže od željenih vrijednosti. Uz ovaj gubitak najčešće se koristi funkcija identiteta u izlaznom sloju.

$$\begin{aligned} p(\hat{y} = y \mid x) &= \mathcal{N}(y \mid \mu = \hat{y}, \underline{\Sigma} = \underline{I}) \\ &= \frac{1}{\sqrt{2\pi}\Sigma^{1/2}} \exp(-(y - \hat{y})^T \Sigma^{1/2} (y - \hat{y})) \end{aligned} \tag{3.8}$$

Uvrštavanjem u (3.5) dobivamo kvadratni gubitak:

$$
\begin{aligned}
L(\theta \mid x, y) &= -\log\left[\frac{1}{\sqrt{2\pi}\Sigma^{1/2}}\right] - \log\left[\exp(-(y-\hat{y})^T\Sigma^{1/2}(y-\hat{y}))\right] \\
&= -\log c - (y-\hat{y})^T\Sigma^{1/2}(y-\hat{y}) \\
&= \left|\begin{array}{l} c = konst. \\ \Sigma = I \end{array}\right| \\
&= (y-\hat{y})^T(y-\hat{y}) = \sum_i (y_i - \hat{y}_i)^2 \qquad (3.9)
\end{aligned}
$$

U problemima klasifikacije, pretpostavlja se da su izlazi mreže međusobno isključive slučajne varijable te stoga prate generaliziranu Bernoullijevu distribuciju (kategoričku distribuciju).

$$
p(\hat{y} = y \mid x) = \hat{y}_1^{y_1} \cdot \hat{y}_2^{y_2} \cdots \hat{y}_c^{y_c} = \prod_{i=1}^{C} \hat{y}_i^{y_i}, \quad \sum_i \hat{y}_i = 1 \qquad (3.10)
$$

Uvrštavanjem u 3.5 dobivamo kategorički gubitak:

$$
L(\theta \mid x, y) = -\log\left[\prod_{i=1}^{C} \hat{y}_i^{y_i}\right] = -\sum_{i=1}^{C} y_i \log(\hat{y}_i) \qquad (3.11)
$$

Optimizacija širenjem unatrag

Funkcija gubitka opisuje pogrešku čitave mreže te ovisi o svakom ugodljivom parametru mreže. Takva formulacija problema omogućuje nam da svaki parametrar mreže ugađamo gradijentnim spustom. Dakle, za parametriziranu funkciju $f(x;\theta)$ tražimo one parametre θ^* za koje je gubitak najmanji na podatkovnom skupu.

$$
\theta^* = argmin_\theta\ L(\theta \mid \mathcal{D}), \quad \forall(x, y) \in \mathcal{D} \qquad (3.12)
$$

Optimiranje gradijentnim spustom zahtjeva derivabilnost funkcije koju optimiziramo po ulazima, što izrazi (3.9) i (3.11) zadovoljavaju. Pri tome koristi se pravilo ulančavanja parcijalne derivacije kompozicije funkcija.

$$
\frac{d}{dx} f(g(x)) = \frac{\partial f(g(x))}{\partial g(x)} \cdot \frac{\partial g(x)}{\partial x} \qquad (3.13)
$$

Derivacijom funkcije gubitka za regresiju (3.9) po ulazima, uz pretpostavku derivabilnosti čitave neuronske mreže po ulazima, dobivamo sljedeći izraz:

$$
\begin{aligned}
\frac{dL(\theta \mid x, y)}{dx} &= \frac{d}{dx} \sum_i (y_i - \hat{y}_i(x))^2 \\
&= 2 \cdot \sum_i (y_i - \hat{y}_i(x)) \cdot \frac{d\hat{y}_i(x)}{dx} \qquad (3.14)
\end{aligned}
$$

Derivaciju funkcije gubitka za klasifikaciju možemo drastično pojednostaviti ako za izlaznu funkciju izlaznog sloja mreže odaberemo funkciju *softmax* (3.15), a izlaze kodiramo *one-hot* oznakama (3.16). Ulaz u funkciju softmax su aktivacije neurona izlaznog sloja koje su ovdje označene vektorom $\vec{s}(x)$.

$$\hat{y}(\vec{x}) = \mathrm{softmax}(\vec{s}(x)) = \frac{\exp(\vec{s}(x))}{\sum_{i=1}^{C} exp(s_i(x))} \tag{3.15}$$

$$\vec{y} = [y_1, y_2, \cdots, y_C], \quad y_i \in \{0, 1\}, \quad \sum_{i=1}^{C} y_i = 1 \tag{3.16}$$

Uvrštavanjem u (3.11) dobivamo:

$$\begin{aligned} L(\theta \mid x, y) &= -\sum_{i=1}^{C} y_i \log \frac{\exp(s_i(x))}{\sum_{j=1}^{C} exp(s_j(x))} \\ &= -\sum_{i=1}^{C} \left[y_i \cdot s_i(x) - y_i \log \left(\sum_{j=1}^{C} exp(s_j(x)) \right) \right] \\ &= -\sum_{i=1}^{C} [y_i \cdot s_i(x)] + \log \left[\sum_{j=1}^{C} exp(s_j(x)) \right] \cdot \sum_{i=1}^{C} y_i \\ &= \log \left[\sum_{j=1}^{C} exp(s_j(x)) \right] - \sum_{i=1}^{C} [y_i \cdot s_i(x)] \end{aligned} \tag{3.17}$$

Derivacijom (3.17) po ulazima, koristeći pravilo (3.13) dobivamo:

$$\begin{aligned} \frac{dL(\theta \mid x, y)}{dx} &= \frac{dL(\theta \mid x, y)}{d\vec{s}(x)} \cdot \frac{d\vec{s}(x)}{dx} \\ &= \left[\vec{s}(x) - y \right] \cdot \frac{ds(x)}{dx} \end{aligned} \tag{3.18}$$

S obzirom da se mreža sastoji od ulančanih nelinearnih neurona s parametrima, gradijent gubitka moramo proslijediti sekvencijalno širenjem unazad (prema ulazima u mrežu). Pojedini neuron možemo smatrati parametriziranom funkcijom koju je moguće prikazati grafom 3.1.2. Vidimo da se ulazni gradijent prolaskom kroz neuron širi na ostale elemente i na ulaze neurona koji vode do prethodnih neurona. Primijetimo i da se širi u suprotnom smjeru od toka podataka. Iz toga proizlazi naziv "*širenjem unazad*" (engl. *backpropagation*).

[IMAGE: *neuron kao graf + tok gradijenata*]

Želimo li učiti mrežu gradijentnim spustom, svaki parametar mreže treba imati pristup gradijentu funkcije gubitka. S obzirom da je neuron parametrizirana funkcija, pomoću koje ulančavanjem gradimo mrežu, dovoljno je pokazati da pojedini neuron osigurava svojim parametrima pristup gradijentu te da gradijent šalje svojim prethodnicima s kojima je povezan. Na slici 3.1.2 vidimo da je to ostvarivo, što dokazuju i izrazi:

$$\begin{aligned} \frac{\partial L(x,y;\theta)}{\partial s(x;w)} &= \frac{\partial L(x,y;\theta)}{\partial o(x;w)} \cdot \frac{\partial o(x;w)}{\partial s(x;w)} \\ &= \frac{\partial L(x,y;\theta)}{\partial o(x;w)} \cdot f'(x) \end{aligned} \tag{3.19}$$

$$\begin{aligned} \frac{\partial L(x,y;\theta)}{\partial x_i} &= \frac{\partial L(x,y;\theta)}{\partial s(x;w)} \cdot \frac{\partial s(x;w)}{\partial x_i} \\ &= \frac{\partial L(x,y;\theta)}{\partial o(x;w)} \cdot f'(x) \cdot w_i \end{aligned} \tag{3.20}$$

$$\begin{aligned} \frac{\partial L(x,y;\theta)}{\partial w_i} &= \frac{\partial L(x,y;\theta)}{\partial s(x;w)} \cdot \frac{\partial s(x;w)}{\partial w_i} \\ &= \frac{\partial L(x,y;\theta)}{\partial o(x;w)} \cdot f'(x) \cdot x_i \end{aligned} \tag{3.21}$$

$$\begin{aligned} \frac{\partial L(x,y;\theta)}{\partial w_0} &= \frac{\partial L(x,y;\theta)}{\partial s(x;w)} \cdot \frac{\partial s(x;w)}{\partial w_0} \\ &= \frac{\partial L(x,y;\theta)}{\partial o(x;w)} \cdot f'(x) \cdot 1 \end{aligned} \tag{3.22}$$

Stohastički gradijentni spust

U prošlom poglavlju koristili smo funkciju gubitka na jednom podatkovnom paru, no želimo minimizirati očekivanje funkcije gubitka (3.23) na cijelom podatkovnom skupu jer želimo što preciznije procijeniti naš model i učit ga da aproksimira čitavu uzorkovanu funkciju što bolje. Nažalost, podatkovni skupovi poput spomenutog u poglavlju 2.3 vrlo su veliki i nije ih praktično koristiti pri optimizaciji gradijentom jer bismo ih čitave morali držati u brzoj memoriji. Ovo predstavlja velik problem pri računanju s grafičkim karticama. Očekivanje gubitka služi nam za procjenu smjera i veličine gradijenta. Ako gradijent procjenjujemo na čitavom podatkovnom skupu on će nas dovesti do prvog optimuma koji najčešće nije dobar, odnosno snažno ovisi o odabiru početne točke.

S druge strane, ako gradijent računamo na pojedinom podatkovnom paru kao u (3.11) ostvarili smo stohastičko kretanje jer će svaki primjer usjeriti gradijent u drugom smjeru. Posljedica toga je dulje vrijeme pretrage, no veća otpornost na loše lokalne optimume. Iako će algoritam generalno konvergirati, postupak je zahtjevan jer za svaki primjer podatkovnog skupa trebamo provesti korak algoritma. Ovaj problem također ograničava svojstvo ubrzanja grafičkim karticama koje nude masivnu paralelizaciju operacija nad podatcima.

Iz navedenih razloga želimo vršiti procjenu gradijenta na "nekoliko" primjera. Podskup takvih primjera nazivamo **mini-grupom** (engl. *mini-batch*). Najčešće se uzima najveći broj primjera koji grafička kartica može efikasno obraditi i koji je potencija broja 2. Mini-grupe i dalje zadržavaju stohastičnost kretanja gradijenta (posebice ako je veličina manja od broja klasa), ali procjenjuju bolje od potpuno stohastičnog te brže konvergiraju. Sada formula (3.23) poprima oblik koji odgovara očekivanju pojedine mini-grupe $\mathcal{M}_i$.

$$\begin{aligned} L(\theta \mid \mathcal{D}) &= \mathbb{E}_{(x,y)\sim p_{\mathcal{M}_i}}[L(\theta \mid x, y)] \\ &= -\frac{1}{|\mathcal{M}_i|} \sum_{(x,y)\in\mathcal{M}_i} \log\, p(\hat{y} = y \mid x) \end{aligned} \tag{3.23}$$

Mini-grupe se najčešće definiraju prije početka učenja nasumičnim rasporedom. To je najjednostavniji i najbrži pristup, no definiranjem fiksnih mini-grupa unosi induktivnu pristranost jer ne možemo znati je li odabran raspored idealan za model koji evaluiramo. Pristranost možemo ublažiti miješanjem podatkovnog skupa između epoha, no skupocijenost te operacije ograničava njeno korištenje u praksi. Također, problem mogu stvoriti mini-grupe pristrane jednoj ili više dominantnih (većinskih) klasa koje će snažno usmjeravati gradijent prema tim klasama i zanemarivati ostale. Taj problem je posebno izražen u nebalansiranim podatkovnim skupovima, a može se zaobići tehnikom težinskog uzorkovanja s ponavljanjem gdje je vjerojatnost odabira primjera obrnuto proporcionalna dominantnosti (brojnosti) njegove klase. Treba paziti da se provede dovoljan broj uzorkovanja tako da je model učen na svakom primjeru podatkovnog skupa.

Input: funkcija $f(\vec{x})$

Input: inicijalizirani parametar θ_0

Input: stopa učenja η

Input: kriterij zaustavljanja k

while *kriterij k nije zadovoljen* **do**

  for $\mathcal{M}_i \in \mathcal{D}$ **do**

    $\vec{g_i} \leftarrow \frac{1}{|\mathcal{M}|} \sum_{(\vec{x},\vec{y}) \in \mathcal{M}} \vec{\nabla}_\theta L(\theta \mid \vec{x}, \vec{y})$;

    $\vec{\theta_{i+1}} \leftarrow \vec{\theta_i} - \eta \cdot \vec{g_i}$

  end

end

Algorithm 2: Stohastički gradijentni spust

Optimizator

Optimizator brine o pomaku odnosno ažuriranju parametara modela pri gradijentnom spustu. Dosad smo vidjeli dva načina: klasični s koeficijentom pomaka (algoritam 1) i s dodatkom momenta (3.4). Moment

TODO: *uporaba momenta i momenta na kvadrat (interpretacija)*

TODO: *Adam*

Promijenjiva stopa učenja

TODO: *zbog nekonveksnih izbočina tam dolje, stopa može biti prejaka*

[IMAGE: *konveksasta fja u kojoj skok u višljim regijama stvara manje problema od skokova u nižim*]

TODO: *koristim step reduction*

TODO: *postoji množenje faktorom pri detekciji konvergencije, ali ne želim ga jer ne znam kada konvergira (stepeničasto učenje)*

Inicijalizacija parametara

Inicijalizacija parametara mreže je sinonim za odabir početne točke pri gradijentnom spustu.

TODO: *važnost dobre inicijalizacije*

TODO: *Xavier*

TODO: *postoji i nenadzirana inicijalizacija sloj-po-sloj RBM (Krizhevsky, 2010) ili autoenkoderom, ali u kontekstu evolucije je preskupa*

Pretprocesiranje podataka

TODO: *normalizacija po značajkama*
Koeficijenti se računaju pomoću značajki iz podatkovnog skupa za učenje. Koeficijent varijance ovisi o koeficijentu sredine pa se njihovo računanje ne može paralelizirati.

$$\vec{\mu} = \frac{1}{N} \sum_{\vec{x} \in \mathcal{D}} \vec{x} \tag{3.24}$$

$$\vec{\sigma^2} = \frac{1}{N-1} \sum_{\vec{x} \in \mathcal{D}} (\vec{x} - \vec{\mu})^2 \tag{3.25}$$

Prije no što model dobije ulazne značajke, normaliziramo ih formulom:

$$\vec{z} = \frac{\vec{x} - \vec{\mu}}{\vec{\sigma^2}} \tag{3.26}$$

Koeficijenti normalizacije se računaju samo na skupu za učenje, a normalizacija se mora primijeniti prilikom inferencije odnosno na skupu za testiranje.

[IMAGE: *slika značajki prije i poslije normalizacije*]

TODO: *sada su vrijednosti centriranje sa std.dev.*

3.1.3. Regularizacija

Decizijska granica je sjecište
TODO: *geometrijski opis deciziske granice*

Podatkovni skupovi najčešće sadrže šum zbog nesavršenog uzorkovanja stvarne funkcije, pogrešnog označavanja ili višeznačnosti primjera. Bez regularizacije, model će s ciljem minimiziranja gubitka svoju decizijsku granicu saviti kako bi što ispravnije obuhvatio sve primjere pa čak i šum. Tada kažemo da je model počeo učiti šum odnosno da je **prenaučen**.
TODO: *podnaučena, generalizira, prenaučena*

[IMAGE: *podnaučena, generalizira, prenaučena*]

Regularizacija parametara

Prenaučenost je često posljedica velikih normi vektora parametara. Stoga se u funkciju gubitka dodaje regularizacijski član po težinama $\Omega(\theta)$. Utjecaj regularizacije moguće je mijenjati hiperparametrom α.

$$\tilde{L}(\theta \mid x, y) = L(\theta \mid x, y) + \alpha\Omega(\theta) \tag{3.27}$$

Najčešće se koristi regularizacija **L2** normom koja akumulira kvadrate svih parametara modela. Pri ažuriranju parametra formula dobiva skaliranu vrijednost tog parametra što smanjuje vrijednost negativnog gradijenta.

$$\Omega(\vec{\theta}) = \frac{1}{2}\|\vec{w}\|_2^2 = \frac{1}{2}\sum_i w_i^2 \tag{3.28}$$

$$\vec{\theta}_{i+1} \leftarrow \vec{\theta}_i - \eta \cdot \vec{g}_i + \alpha \cdot \theta \tag{3.29}$$

TODO: *Spomeni pokoju još (L1, adversarial iz Hintona, ...)*

Regularizacija šumom

Unošenjem šuma u podatke netom prije nego ih dobije model moguće je graditi robusnost modela na šum, ali ujedno to sprječava model da se pretrenira na podatkovnom skupu za treniranje. Jedan primjer možemo nasumično zašumiti nekoliko puta te se time povećava efektivna veličina podatkovnog skupa.

U ovom se radu ne koristi regularizacija šumom jer bi mogla imati nepredvidivi utjecaj na učenje. Primjena na slike je vrlo jasna jer uz dovoljno malu amplitudu aditivnog šuma teksture na slici će ostati očuvane zahvaljujući susjedstvu piksela, odnosno izrezivanjem slike obrasci će ostati netaknuti. Značajke podatkovnih skupova navedenih u poglavlju 2.3 su PCA redukcije originalnih slijedova signala koji su ispunjeni šumom očitavanja.

TODO: *ajd svejedno isprobaj, grafi za ljubav*

Rano zaustavljanje

Pri učenju modela funkcija gubitka na skupu za učenje i testiranje generalno opada, no u jednom trenu gubitak generalizacije počinje rasti. Zaustavimo li učenje u trenutku kada je gubitak generalizacije najmanji dobili smo optimalan model za zadane hiperparametre. U praksi te funkcije neće biti naročito glatke zbog stohastičkog gradijentnog spusta te greška će generalizacije povremeno i rasti kako savladava lokalne optimume.

Postupak ranog zaustavljanja tretira broj epoha kao hiperparametar koji se pretražuje po liniji. No umjesto da za svaku vrijednost broja epoha nanovo treniramo model, ovdje ga treniramo jednom i jednostavno odaberemo vrijednost za koju model generalizira najbolje. To upućuje na potrebu krosvalidacije koja je opisana u poglavlju 3.1.4.

[EQUATION: *Algoritam ranog zaustavljanja*] (3.30)

3.1.4. Odabir hiperparametara

Do ovdje su navedeni hiperparametri koji se koriste pri spomenutim tehnikama optimizacije neuronske mreže (poglavlja 3.1.2 - 3.1.3). No neuronska mreža ima i strukturalne hiperparametre.

Arhitektura mreže je vrlo bitan hiperparametar koji određuje složenost modela te utječe na brzinu inferencije i učenja modela. Razvijene su razne arhitekture koje koriste preskočne veze za postizanje vrlo dubokih arhitektura (He et al., 2016; Huang et al., 2017). Preskočne veze omogućavaju direktniji prijenos gradijenta što pomaže kod problema iščezavajućeg gradijenta u dubokim mrežama (poglavlje 3.1.6). Arhitektura može omogućiti dodatnu paralelizaciju inferencije i učenja tako da se teške operacije raspodijele na više uređaja, a rezultati spoje samo kada je to nužno (Krizhevsky et al., 2012).

Izlazne funkcije su također važan hiperparametar, no u praksi se većinom ignoriraju zbog manjka intuicije o njihovom utjecaju na učenje pojedinog modela na pojedinom podatkovnom skupu. U praksi se najčešće odabiru funkcije koje su brze i koje se pokazuju korisnima u raznim radovima. Najčešće se koristi ReLU opisan u poglavlju 4.2.2. U povratnim neuronskim mrežama za rekurzivne slojeve popularan je tangens hiperbolni opisan u poglavlju 4.2.20. U poglavlju 4 navedene su i opisane brojne funkcije te je njihov utjecaj ispitan u poglavlju 7.

Procjena generalizacije i odabir modela

Skup podataka kojim učimo model najčešće ne opisuje stvarnu funkciju potpuno, već sadrži primjere koje smatramo reprezentativnim i koji su dovoljni za njeno modeliranje. Kako bismo procjenili koliko dobro naš model procjenjuje stvarnu funkciju ispitujemo model na podatcima koji nisu korišteni prilikom učenja, odnosno na neviđenim podatcima. Ta se metoda zove **unakrsna validacija**, a skupove nazivamo **skupom za učenje** i **skupom za testiranje**. Dakako, važno je pobrinuti se da su oba skupa reprezentativna stvarnoj funkciji, ali da ne sadrže iste primjere. U suprotnom mreža će naučiti pogrešnu funkciju što može dati lažno pesimistične rezultate ili ćemo nepotpuno ili pristrano testirati što može dovesti do lažno optimističnih rezultata.

Svojstvo modela da dobro modelira na neviđenim primjerima naziva se generalizacija i detaljnije je opisano u poglavlju 3.1.5. Pri odabiru hiperparametara ili pri odabiru modela potrebno je usporediti generalizaciju svakog pomoću zajedničke mjere. U problemima regresije najčešće se koristi ukupna vrijednost funkcije gubitka na čitavom skupu za testiranje.

$$\begin{array}{|c|c|c|} \hline \hat{y} \backslash y & \top & \bot \\ \hline \top & \text{TP} & \text{FP} \\ \hline \bot & \text{FN} & \text{TN} \\ \hline \end{array} \tag{3.31}$$

Pri **binarnoj klasifikaciji** definiramo **matricu zabune** (3.31) koja sadrži četiri elementa (3.38) koji definiraju vrstu pogodka i pogreške.

$$\begin{aligned} \text{Stvarno pozitivni:} \quad TP &= \sum_{(x,y)\in\mathbb{D}} \mathbb{1}\{h(x) = \top \land y = \top\} \\ \text{Stvarno negativni:} \quad TN &= \sum_{(x,y)\in\mathbb{D}} \mathbb{1}\{h(x) = \bot \land y = \bot\} \\ \text{Lažno pozitivni:} \quad FP &= \sum_{(x,y)\in\mathbb{D}} \mathbb{1}\{h(x) = \top \land y = \bot\} \\ \text{Lažno negativni:} \quad FN &= \sum_{(x,y)\in\mathbb{D}} \mathbb{1}\{h(x) = \bot \land y = \top\} \end{aligned} \tag{3.32}$$

Iz tih skupova tada gradimo složenije mjere. **Točnost** je mjera kojom iskazujemo postotak točno pogođenih primjera:

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.33}$$

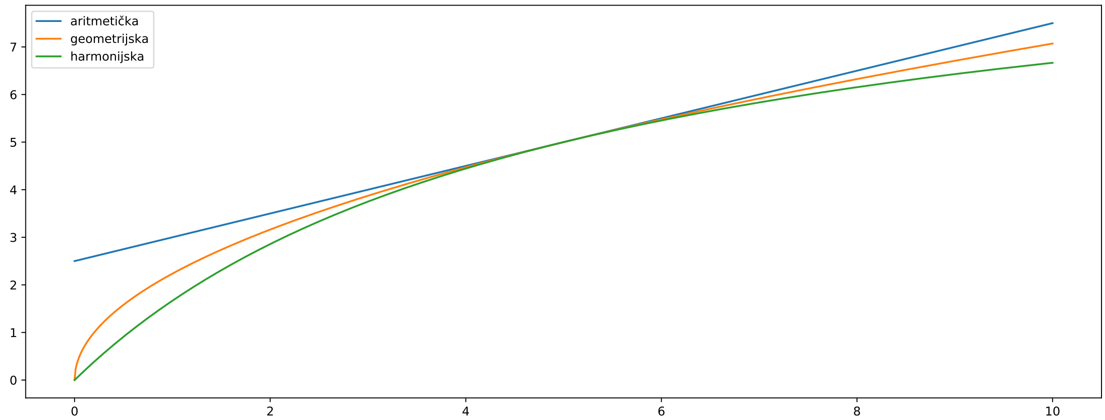
Točnost je dobra mjera, no samo ako je brojnost klasa u podatkovnom skupu balansiran. Ako je brojnost jedne klase puno veća od druge tada će trivijalan klasifikator, koji sve primjere klasificira u tu klasu, davati veliku točnost i razlika naspram ispravnijeg klasifikatora biti će nezamjetna. Stoga se kod nebalansiranih setova češće koristi $\mathbf{F}_1$ **mjera**, koja uzima u obzir **preciznost** klasifikatora u razlikovanju pozitivnih primjera od negativnih (3.34) i njegov **odziv** odnosno obuhvat svih pozitivnih primjera testnog skupa (3.35). F_1 mjera je definirana kao hamonijska sredina između preciznosti i odziva (3.36). Postoji i generalizirana mjera F_β koja dodjeljuje veću težinu preciznosti ili odzivu (3.37), no u ovom radu koristi se samo F_1 koja pridijeljuje jednaku težinu. Harmonijska sredina se koristi jer je najstroža između Pitagorinih mjera za sredinu kao što prokazuje slika 3.2.

$$\text{Preciznost:} \quad P = \frac{TP}{TP + FP} \tag{3.34}$$

$$\text{Odziv:} \quad R = \frac{TP}{TP + FN} \tag{3.35}$$

$$\mathrm{F}_1: \quad F_1 = 2 \cdot \frac{P \cdot R}{P + R} \tag{3.36}$$

$$\mathrm{F}_\beta: \quad F_\beta = (1 + \beta^2) \cdot \frac{P \cdot R}{\beta^2 \cdot P + R} \tag{3.37}$$



Slika 3.2: Pitagorine mjere za sredinu između dviju vrijednosti

Mjere binarne klasifikacije možemo primijeniti pri **višeklasnoj klasifikaciji**, no matrica zabune je dimenzija $C \times C$ gdje je C broj klasa. Elementi matrice računaju se slično kao i kod binarne klasifikacije, ali za svaku klasu posebno.

$$\begin{aligned}
\text{Stvarno pozitivni:} \quad TP_i &= \sum_{(x,y)\in\mathbb{D}} \mathbb{1}\{h(x) = C_i \land y = C_i\} \\
\text{Stvarno negativni:} \quad TN_i &= \sum_{(x,y)\in\mathbb{D}} \mathbb{1}\{h(x) \neq C_i \land y \neq C_i\} \\
\text{Lažno pozitivni:} \quad FP_i &= \sum_{(x,y)\in\mathbb{D}} \mathbb{1}\{h(x) = C_i \land y \neq C_i\} \\
\text{Lažno negativni:} \quad FN_i &= \sum_{(x,y)\in\mathbb{D}} \mathbb{1}\{h(x) \neq C_i \land y = C_i\}
\end{aligned} \tag{3.38}$$

Za izračun složenijih mjera poput točnosti i F_1 mjere moramo računati prosjek po klasama. Razlikujemo dva pristupa računanju prosjeka: makro i mikro. **Makro prosjekom** se prvo izračunaju mjere svake klase naspram svih ostalih te se uzme njihov prosjek. Ova mjera pretpostavlja jednak utjecaj svih klasa nez obzira na njihovu veličinu (Murphy, 2012).

$$Acc^M = \sum_{i=1}^{K} \frac{Acc_i}{K} \quad P^M = \sum_{i=1}^{K} \frac{P_i}{K} \quad R^M = \sum_{i=1}^{K} \frac{R_i}{K} \quad F_1^M = \sum_{i=1}^{K} \frac{F_{1;i}}{K} \tag{3.39}$$

Mikro prosjekom se prvo zbroje matrice zabune po pojedinim klasama, a zatim se nad zbrojenom matricom računaju mjere. Na mikro prosjek više utječe veličina klasa i koristi se u nebalansiranim skupovima.

$$\begin{aligned} Acc^\mu &= \frac{\sum_i (TP_i + TN_i)}{\sum_i (TP_i + TN_i + FP_i + FN_i)} = Acc^M \\ FP = FN \implies P^\mu = R^\mu = F_1^\mu &= \frac{\sum_i TP_i}{\sum_i (TP_i + FP_i)} \end{aligned} \tag{3.40}$$

Krosvalidacija

TODO: *ovo je bitno*

... pri čemu se skup za učenje podijeli na manji skup za učenje i skup za validaciju. Tek nakon što otkrijemo optimalan broj epoha model trenira na čitavom skupu za učenje i ispituje na skupu za testiranje koji mjeri stvarnu generalizaciju.

Pretraživanje po rešetci

Najjednostavniji način za pretraživanje hiperparametara je pretraživanje po rešetci. Za svaki hiperparametar koji optimiziramo definiramo vrijednosti koje želimo ispitati. Algoritam tada evaluira dani model za svaku kombinaciju hiperparametara i vraća kombinaciju ili model koji ostvaruje najbolje rezultate.

Iako se optimalni hipermarametri mogu nalaziti izvan zadanih skupova i neće biti pronađeni, postupak je brz i daje dovoljno dobre rezultate za praktičnu primjenu. Često je dovoljno da pronađe kombinaciju hiperparametara za koju model ne divergira niti prestaje učiti određen broj iteracija.

3.1.5. Svojstva

Univerzalna aproksimacija

Teorem univerzalne aproksimacije tvrdi da unaprijedna neuronska mreža može modelirati proizvoljnu Borel mjerljivu funkcju proizvoljno dobro uz nekoliko uvijeta: mora imati linearni izlaz, barem jedan nelinearni skriveni sloj koji koristi sažimajuću funkciju i "dovoljan" broj skrivenih neurona. Dakle, postoji arhitektura i postoje parametri kojima neuronska mreža može modelirati zadani podatkovni skup. No, teorem ne iskazuje kako doći do tih parametara što optimizaciju neuronske mreže čini vrlo teškom (Goodfellow et al., 2016).

Dubina

Iako je prema teoremu univerzalne aproksimacije dovoljan jedan nelinearni skriveni sloj za predstavljanje proizvoljne Borel mjerljive funkcije, gornja granica veličine tog sloja je eksponencijalno velika naspram broja ulaza što je netraktabilno. Dodavanjem dubine moguće je iskoristiti pravilnosti u funkciji koju aproksimiramo kako bismo smanjili potreban broj neurona. Primjer su funkcije simetrične oko neke osi. Ako skriveni slojevi mreže vrše preklapanje te funkcije preko same sebe, uzastopnim preklapanjem dobiva se sve jednostavnija funkcija. Preklapanje mogu vršiti po-dijelovima-linearne izlazne funkcije poput ReLU i Maxout. Dakako, ne postoji garancija da stvarna funkcija zadovoljava svojstvo simetrije, no u praksi dublje mreže generaliziraju bolje (Krizhevsky et al., 2012; Srivastava et al., 2015; He et al., 2016; Huang et al., 2017). Postoje i druge interpretacije utjecaja dubine, poput svojstva dekompozicije zadatka na manje cjeline ili interpretacija neuronske mreže ako računalnog programa koje nadilaze temu ovog rada (Goodfellow et al., 2016).
TODO: *VC dimenzija*

Kompresija

TODO: *kompresija*
TODO: *https://www.quantamagazine.org/new-theory-cracks-open-the-black-box-of-deep-learning-20170921/*

Generalizacija

[IMAGE: *underfit-fit-overfit*]

TODO: *generalizacija*

[IMAGE: *train-test U curve*]

Ovo je posebno izraženo kod klasifikacije slika gdje za jednu generičku klasu (npr. automobil) postoji neprebrojivo mogućih slika s različitim modelom, bojom ili kutom gledanja automobila. No mi raspolažemo sa svega nekoliko stotina tisuća primjera koje smatramo reprezentativnim za tu klasu i želimo izgraditi klasifikator koji je robustan na većinu perturbacija slike.
TODO: *negdje spomeni da su neuralke zapravo vrlo ograničene jer ograničavamo prostor mogućnosti zadavanjem fiksnih izlaznih fja, arhitekture i načina optimizacije (induktivna pristranost ograničavanjem skupa hipoteza). leži negdje između GP i ručno izgrađenih modela (jer je neuralka samo stablo funkcija kao u TF). CGP unosi ograničenje strukture što je bliže neuralki i daje zanimljive rezultate (atari cgp). Čini se da im godi balans između strukture i nasumičnosti.*

3.1.6. Problemi

Odabir hiperparametara

Arhitektura, prijenosne funkcije i parametri definiraju neuronsku mrežu te njihov pravilan odabir značajno utječe na performanse neuronske mreže. Nažalost nije ih moguće optimalno odabrati u zatvorenom obliku, već se to svodi na problem pretraživanja kao što je pretraživanje po rešetci 3.1.4. Arhitekture mogu poprimiti vrlo složene oblike kao što je predstavljeno radovima **?**, He et al. (2016), Huang et al. (2017), Szegedy et al. (2015) i Redmon et al. (2016) te se najčešće grade ručno s maksimalnom traktabilnom složenošću mreže. Detaljnije o odabiru izlaznih funkcija napisano je u poglavlju 4.

Iščezavajući gradijent

TODO: *Problem dubokih arhitektura*

Pretreniranost i neprijateljski primjeri

TODO: *Problem pretreniranosti + neprijateljski primjeri*

Derivabilne neuronske mreže optimiziraju se optimizatorom koji određuje kako se mijenjaju parametri. Za ugađanje parametara najčešće se koristi gradijentni spust, uz pretpostavku derivabilnosti čitave neuronske mreže. Kada pretpostavka ne vrijedi najčešće se koriste evolucijski algoritmi.

4. Izlazne funkcije

Izlazna funkcija služi unošenju nelinearnosti u neuronsku mrežu i ima utjecaj na njeno učenje. Prilikom inferencije izlazna funkcija stvara nelinearnosti u decizijskoj granici koje su parametrizirane parametrima neurona na koji se primijenjuje. U slojevitim arhitekturama izlazne funkcije se primijenjuju uzastopno s različitim parametrima i vrše nelinearne projekcije iz jedne dimenzije u drugu.

Prilikom učenja neuronske mreže važan nam je utjecaj derivacije izlazne funkcije na gradijent pri širenju unatrag. Prolaskom unazad gradijent se množi s parametrima slojeva koji se mogu zapisati matricom težina $\underline{W}$. Uzastopnom primijenom matrica težina, ako su loše kondicionirane, može doći do "*isćezavanja*" ili "*eksplozije*" gradijenta.

4.1. Odabir izlazne funkcije

Za odabir izlazne funkcije ne postoji jasno pravilo. U praksi se odabiru empirijski po pokazanim rezultatima i brzinom (te implementacijskom podrškom), s preferencijom onih koje imaju neka teorijski potkrijepljena svojstva.

TODO: *Bitka za odabir izlazne fje (nađi onaj rad di pljuje po sigmoidi i relu (elu rad?))*

U radu Basirat i Roth (2018) autori koriste simboličku regresiju kako bi izgradili dvodjelnu izlaznu funkciju, sa zasebnom funkcijom u negativnoj i pozitivnoj domeni te definiraju dvije ručno izgrađene funkcije. Pretražuju se funkcije koje su jednostavne kompozicije popularnih izlaznih funkcija s ciljem iskorištavanja njihovih dobrih svojstava. Funkcije se primijenjuju na duboke arhitekture ResNet56 i VGG16 i na nekoliko podatkovnih skupova: CIFAR-10, CIFAR-100 i Tiny-ImageNet. Rezultati pokazuju poboljšanje kompozicije u odnosu na popularne funkcije gdje pozitivan dio često sadrži linearnu komponentu, a oblikom funkcije podsjećaju na zglobnicu (slika 4.2).

4.1.1. Izlazne funkcije s učećim parametima

TODO: *članak: učeći parametri u fjama (onaj stari)*

U radu Ertugrul (2018) autori istražuju učeće izlazne funkcije ...

U radu TAF...

4.1.2. Periodičke izlazne funkcije

TODO: *članak: fourier mreža s periodičkim fjama*

TODO: *Članak: taming the waves*

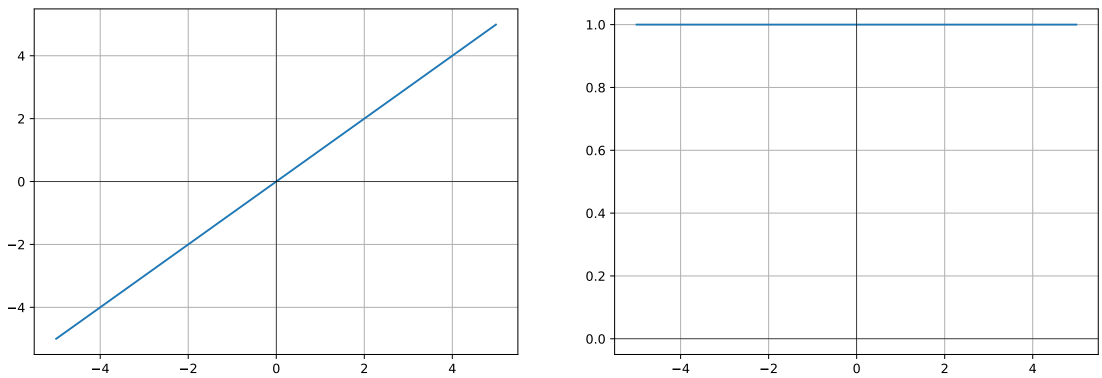
TODO: *ostale reference iz taming the waves*

4.2. Popularne izlazne funkcije

U nastavku su navedene izlazne funkcije koje je autor pronašao u literaturi i koje su isprobane u ovom radu. Za svaku funkciju napisana je formula i iscrtan izgled funkcije i njene derivacije te navedena neka poznata svojstva.

4.2.1. Funkcija identiteta

(engl. *Identity function*)



Slika 4.1: Funkcija identiteta i njena derivacija

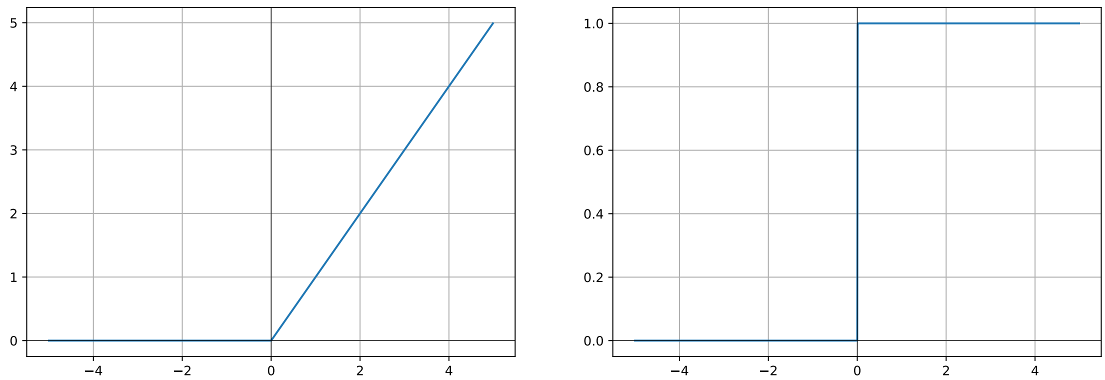
$$f(x) = x \qquad f'(x) = 1 \tag{4.1}$$

Funkcija identiteta je jednostavna i brza, no njome neuronska mreža može naučiti samo linearne funkcije. Ako primijenimo funkciju na dva uzastopna sloja vidimo da je konačna funkcija ponovno linearna što znači da ne možemo naučiti mrežu na nelinearnim podatcima:

$$\begin{aligned} f_l(x) &= w_l \cdot x + b_l \\ f_1(f_2(x)) &= w_1 \cdot (w_2 \cdot x + b_2) + b_1 \\ &= \underline{w_1 \cdot w_2} \cdot x + \underline{w_1 \cdot b_2 + b_1} \\ &= w_{1,2} \cdot x + b_{1,2} \end{aligned} \tag{4.2}$$

4.2.2. Zglobnica ili ispravljena linearna jedinica (ReLU)

(engl. *Rectified linear unit*)



Slika 4.2: Funkcija ReLU i njena derivacija

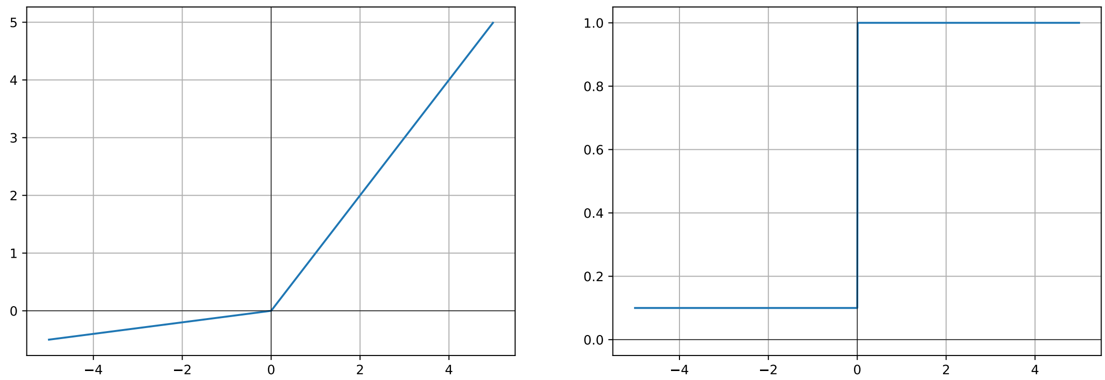
$$\begin{aligned} f(x) &= \begin{cases} x, & \text{ako } x > 0 \\ 0, & \textit{inače} \end{cases} \\ &= max(0, x) \end{aligned} \qquad f'(x) = \begin{cases} 1, & \text{ako } x > 0 \\ 0, & \textit{inače} \end{cases} \tag{4.3}$$

Funkcija ReLU svoje korijene povlači iz rada bioloških neurona, a u dubokom učenju je zamijenila dotadašnju sigmoidu i tangens hiperbolni. Zahvaljujući prizemljenosti na negativnoj domeni ReLU omogućava neuronskoj mreži učenje rijetke reprezentacije. Njihovim kombiniranjem u dubokoj mreži dobivamo model s eksponencijalno puno linearnih regija koje dijele zajedničke parametre (Nair i Hinton, 2010). Zahvaljujući linearnosti na pozitivnoj domeni funkcija ne doprinosi iščezavanju ni eksploziji gradijenta, a sam izračun funkcije i derivacije je vrlo brz i efikasan. (Glorot et al., 2011)

Problem u negativnoj domeni je mogućnost blokiranja gradijenta i neaktivnih neurona (mrtvi neuroni), no dok god postoji put kroz mrežu gdje su neuroni aktivni (u pozitivnoj domeni) učenje će raditi, što pokazuju i rezultati (Glorot et al., 2011). Drugi problem je u neograničenosti funkcije u pozitivnoj domeni, no nju rješavamo regularizacijom koja ograničava veličinu ulaza u neuron (Glorot et al., 2011).

4.2.3. Propusna ispravljena linearna jedinica (LReLU)

(engl. *Leaky ReLU*)



Slika 4.3: Funkcija LReLU i njena derivacija za $\alpha = 0.1$

$$f(x) = \begin{cases} x, & \text{ako } x > 0 \\ \alpha x, & \textit{inače} \end{cases} \qquad f'(x) = \begin{cases} 1, & \text{ako } x > 0 \\ \alpha, & \textit{inače} \end{cases} \tag{4.4}$$
$$= max(\alpha x, x)$$

Problem funkcije ReLU (poglavlje 4.2.2) je što postoji mogućnost pojave mrtvih neurona, koji su uvijek u neaktivnom području i kroz njih ne teče gradijent. Taj problem efektivno smanjuje kapacitet modela pod cijenu nepotrebnog memorijskog opterećenja i smanjene brzine izvođenja. Iz navedenih razloga uvedena je propusna ReLU funkcija koja u "neaktivnom" području propušta mali gradijent. Rezultati na zadatku prepoznavanja govora pokazuju da je propusna ReLU funkcija ekvivalentna standardnom ReLU, no rezultati su prikazani na relativno plitkim arhitekturama (do 4 skrivena sloja). (Maas et al.)

4.2.4. Parametrizirana ispravljena linearna jedinica (PReLU)

(engl. *Parametric ReLU*)

[IMAGE:]

$$f(x) = \begin{cases} x, & \text{ako } x > 0 \\ \alpha x, & \textit{inače} \end{cases} \qquad f'(x) = \begin{cases} 1, & \text{ako } x > 0 \\ \alpha, & \textit{inače} \end{cases} \tag{4.5}$$
$$= max(\alpha x, x)$$

α je učeći parametar

TODO: *koji problem rješava*

TODO: *svojstva*

TODO: *problemi*

4.2.5. Nasumična ispravljena linearna jedinica (RReLU)

(engl. *Randomized leaky ReLU*)

[IMAGE:]

$$f(x) = \begin{cases} x, & \text{ako } x > 0 \\ \alpha x, & \textit{inače} \end{cases} \qquad f'(x) = \begin{cases} 1, & \text{ako } x > 0 \\ \alpha, & \textit{inače} \end{cases} \tag{4.6}$$

$$\alpha \sim U(l, u), \quad l, u \in [0, 1] \tag{4.7}$$

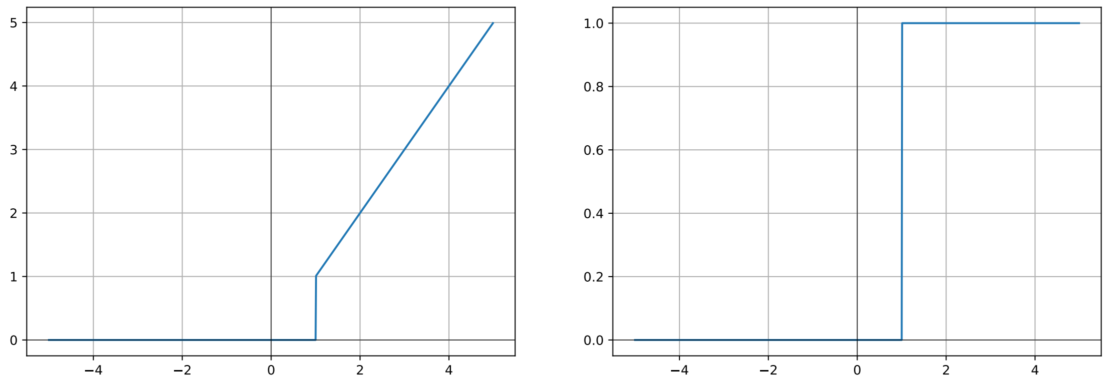
TODO: *koji problem rješava*

TODO: *svojstva*

TODO: *problemi*

4.2.6. Ispravljena linearna jedinica s pragom (ThReLU)

(engl. *Thresholded ReLU*)



Slika 4.4: Funkcija ThReLU i njena derivacija

$$f(x) = \begin{cases} x, & \text{ako } x > \theta \\ 0, & \textit{inače} \end{cases} \qquad f'(x) = \begin{cases} 1, & \text{ako } x > \theta \\ 0, & \textit{inače} \end{cases} \tag{4.8}$$

TODO: *koji problem rješava*

TODO: *svojstva*

TODO: *problemi*

4.2.7. (CReLU)

(engl. *Concatenated ReLU*)

[IMAGE:]

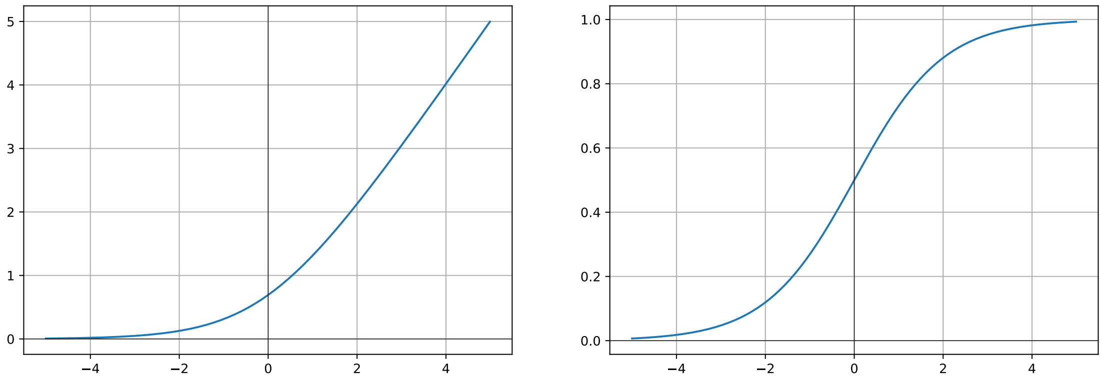
$$??? \tag{4.9}$$

TODO: *koji problem rješava*

TODO: *svojstva*

TODO: *problemi*

4.2.8. Softplus



Slika 4.5: Softplus i njegova derivacija

$$f(x) = log(1 + e^x) \qquad f'(x) = \frac{e^x}{1 + e^x} \tag{4.10}$$

TODO: *koji problem rješava*

TODO: *svojstva*

TODO: *problemi*

Funkcija Softplus nastaje ...

4.2.9. Noisy softplus

[IMAGE:]

$$??? \tag{4.11}$$

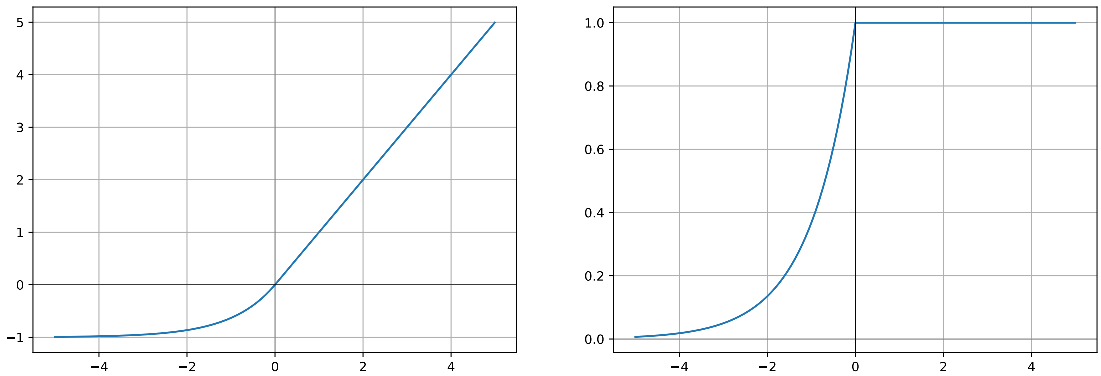
TODO: *koji problem rješava*

TODO: *svojstva*

TODO: *problemi*

4.2.10. Eksponencijalno-linearna jedinica (ELU)

(engl. *Exponential linear unit*)



Slika 4.6: Funkcija ELU i njena derivacija

$$f(x) = \begin{cases} x, & \text{ako } x > 0 \\ \alpha(e^x - 1), & \textit{inače} \end{cases} \qquad f'(x) = \begin{cases} 1, & \text{ako } x > 0 \\ \alpha e^x, & \textit{inače} \end{cases} \tag{4.12}$$

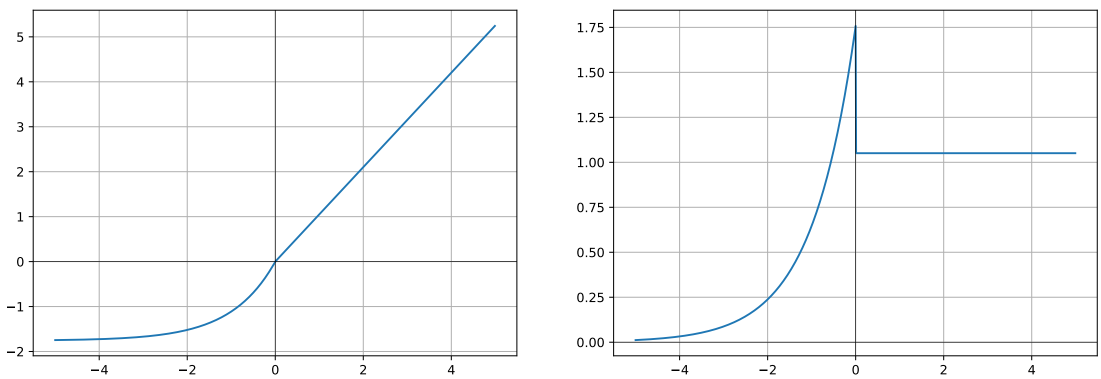
TODO: *koji problem rješava*

TODO: *svojstva*

TODO: *problemi*

4.2.11. Skalirana eksponencijalno-linearna jedinica (SELU)

(engl. *Scaled exponential linear unit*)



Slika 4.7: Funkcija SELU i njena derivacija

$$f(x) = \lambda \begin{cases} x, & \text{ako } x > 0 \\ \alpha(e^x - 1), & \textit{inače} \end{cases} \qquad f'(x) = \lambda \begin{cases} 1, & \text{ako } x > 0 \\ \alpha e^x, & \textit{inače} \end{cases} \tag{4.13}$$

TODO: *koji problem rješava*

TODO: *svojstva*

TODO: *problemi*

4.2.12. (GELU)

(engl. *Gaussian error linear unit*)

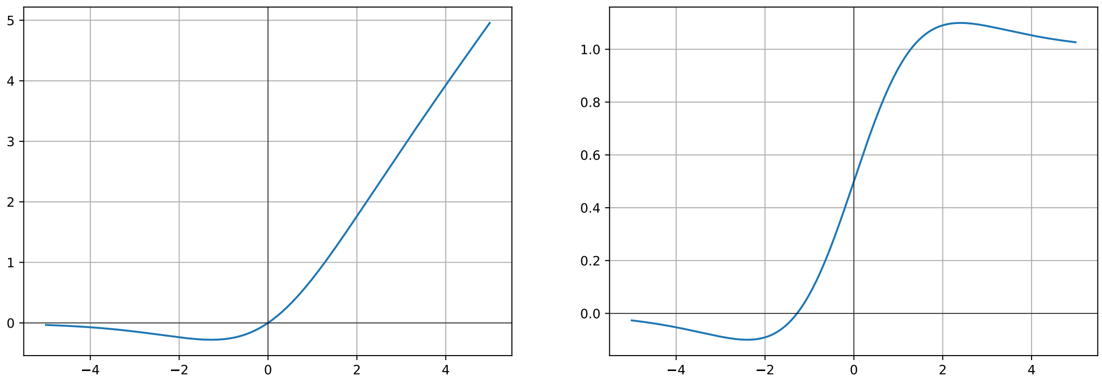
[IMAGE:]

TODO: *koji problem rješava*

TODO: *svojstva*

TODO: *problemi*

4.2.13. Swish



Slika 4.8: Funkcija Swish i njena derivacija

$$
\begin{aligned}
f(x) &= x \cdot \sigma(\beta x) \\
f'(x) &= \sigma(\beta x) \cdot (1 + \beta x \cdot (1 - \sigma(\beta x)) = \frac{e^x \cdot (e^x + x + 1)}{(e^x + 1)^2}
\end{aligned} \tag{4.14}
$$

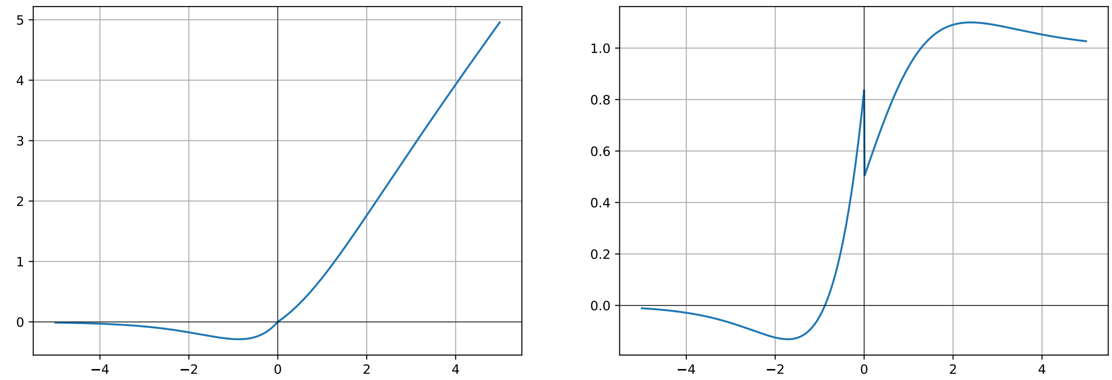
TODO: *koji problem rješava*

TODO: *svojstva*

TODO: *problemi*

4.2.14. ELiSH

(engl. *Exponential Linear Sigmoid Squashing*)



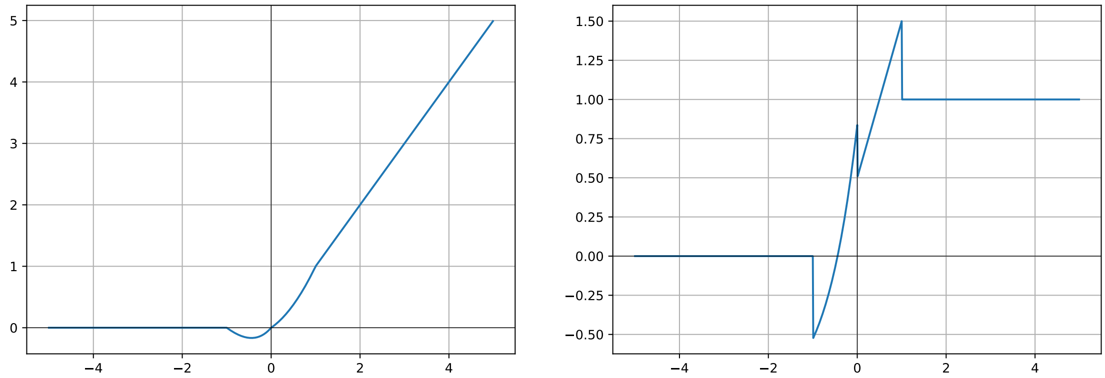
Slika 4.9: Funkcija ELiSH i njena derivacija

$$
\begin{aligned}
f(x) &= \begin{cases} swish(x), & x \geq 0 \\ ELU(x) \cdot \sigma(x), & \textit{inače} \end{cases} \\
f'(x) &= \begin{cases} swish'(x), & x \geq 0 \\ \sigma(x) \cdot (ELU'(x) + ELU(x) \cdot (1 - \sigma(x)), & \textit{inače} \end{cases}
\end{aligned}
\tag{4.15}
$$

Ova funkcija i njena brža aproksimacija *Tvrdi ELiSH* su kompozicije funkcija, različito definirane na pozitivnoj i negativnoj domeni. ELiSH pozitivnu domenu nasljeđuje od funkcije Swish (4.14), a negativna je umnožak funkcije ELU (4.12) i sigmoide (4.20). Obje funkcije su ručno izgrađene s ciljem iskorištavanja dobrog prijenosa informacije koji ostvaruju funkcije Swish i sigmoida te sprečavanjem iščezavajućeg gradijenta pomoću linearne komponente u funkciji Swish. (Basirat i Roth, 2018)

4.2.15. Tvrdi ELiSH

(engl. *Hard ELiSH*)



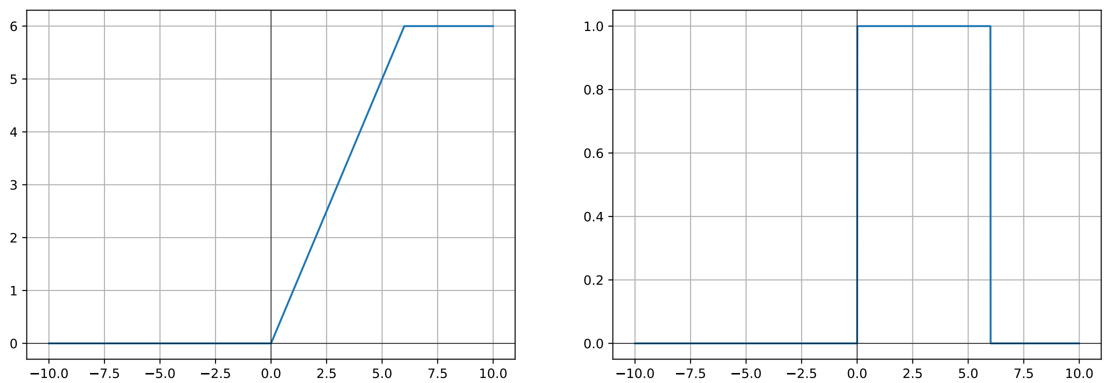
Slika 4.10: Funkcija Tvrdi ELiSH i njena derivacija

$$
\begin{aligned}
f(x) &= \begin{cases} x \cdot a(x), & x \geq 0 \\ ELU(x) \cdot a(x), & \textit{inače} \end{cases} \\
f'(x) &= \begin{cases} a(x) + x \cdot a'(x), & x \geq 0 \\ ELU'(x) \cdot a(x) + x \cdot ELU(x) \cdot a'(x), & \textit{inače} \end{cases}
\end{aligned} \tag{4.16}
$$

$$
\begin{aligned}
a(x) &= \max(0, \min(1, \frac{x+1}{2})) \approx HardSigmoid(x) \\
a'(x) &= \begin{cases} 0.5, & |x| \leq 1 \\ 0, & \textit{inače} \end{cases}
\end{aligned} \tag{4.17}
$$

Ova funkcija je aproksimacija funkcije ELiSH opisane u poglavlju 4.2.14. Uvedena je za potrebe bržeg izvođenja sigmoide i nasljeđuje svojstva ELiSH funkcije. (Basirat i Roth, 2018)

4.2.16. Ograničena ispravljena linearna jedinica (ReLUn)



Slika 4.11: Funkcija ReLU6 i njena derivacija

$$f(x) = \begin{cases} 0, & \text{ako } x < 0 \\ x, & \text{ako } x \in [0, n] \\ n, & \textit{inače} \end{cases} \qquad f'(x) = \begin{cases} 1, & \text{ako } x \in [0, n] \\ 0, & \textit{inače} \end{cases} \tag{4.18}$$

$$= min(n, max(0, x))$$

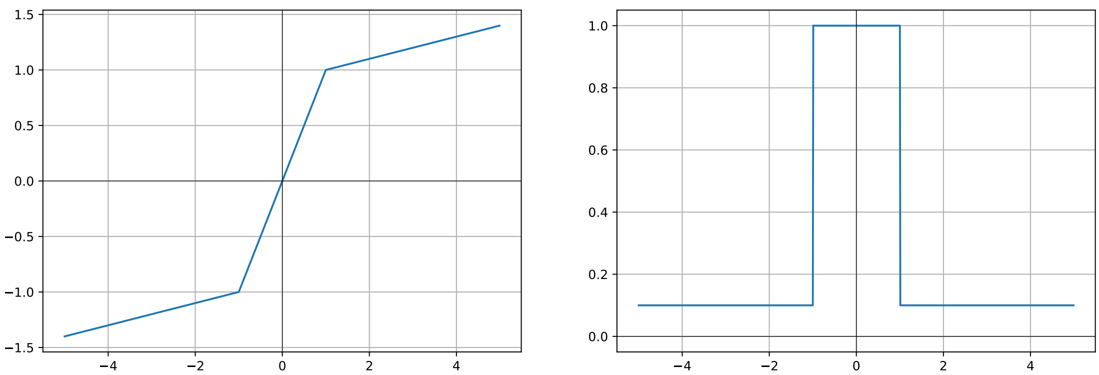
$$n \in \mathbb{R}$$

Ograničena ReLU funkcija uvedena je kako bi ograničeni Boltzmann-ov stroj (engl. *Restricted Boltzmann machine*) brže naučio rijetke značajke, koje kasnije ugađa konvolucijski klasifikator. Za razliku od ReLU koji se može interpretirati kao kompozicija beskonačno mnogo identičnih Bernoullijevih jedinica translatiranih po domeni, ograničeni ReLU interpretiramo kao kompoziciju n jedinica. Za vidljivi sloj korištena je ReLU1, a za skrivene slojeve ReLU6 funkcija. (Krizhevsky, 2010)

S obzirom da je ReLUn neuron aktivan samo na relativno uskoj regiji u pozitivnoj domeni, postoji opasnost od pojave mrtvih neurona (kroz koje ne prolazi gradijent). Idealna regija je $< 0, n >$ jer je u njoj neuron aktivan, a učenjem se može specijalizirati približavanjem zasićenju i stvoriti rijetke reprezentacije. Stoga je potrebno koristiti regularizaciju te pažljivu inicijalizaciju parametara.

4.2.17. Razlomljena linearna jedinica (PLU)

(engl. *Piecewise linear unit*)



Slika 4.12: Funkcija PLU i njena derivacija

$$f(x) = \begin{cases} x, & |x| \leq c \\ \alpha \cdot x, & \textit{inače} \end{cases} \qquad f'(x) = \begin{cases} 1, & |x| \leq c \\ \alpha, & \textit{inače} \end{cases}$$
$$= \max(\alpha x - c(1-\alpha), \min(x, \alpha x + c(1-\alpha))) \tag{4.19}$$

$$\alpha, c \in \mathbb{R}$$

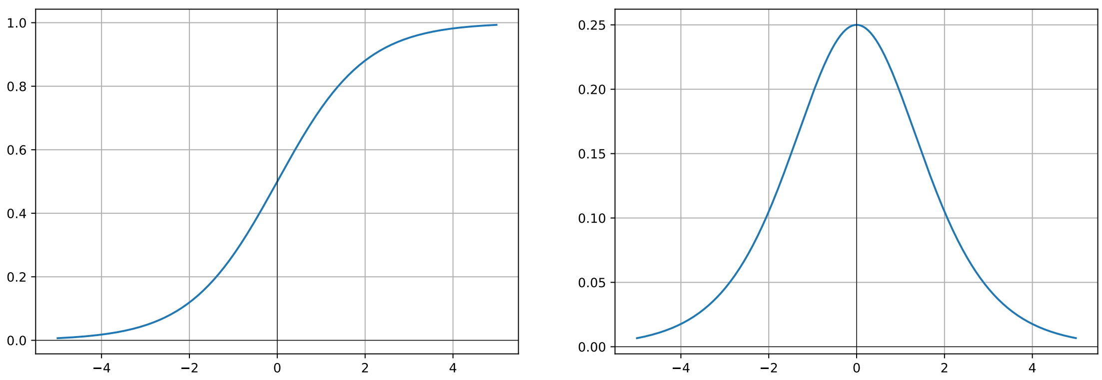
TODO: *koji problem rješava*

TODO: *svojstva*

TODO: *problemi*

4.2.18. Sigmoida (σ)

(engl. *Sigmoid*)



Slika 4.13: Sigmoida i njena derivacija

$$f(x) = \frac{1}{1+e^{-x}} \qquad f'(x) = \sigma(x)(1-\sigma(x)) \tag{4.20}$$

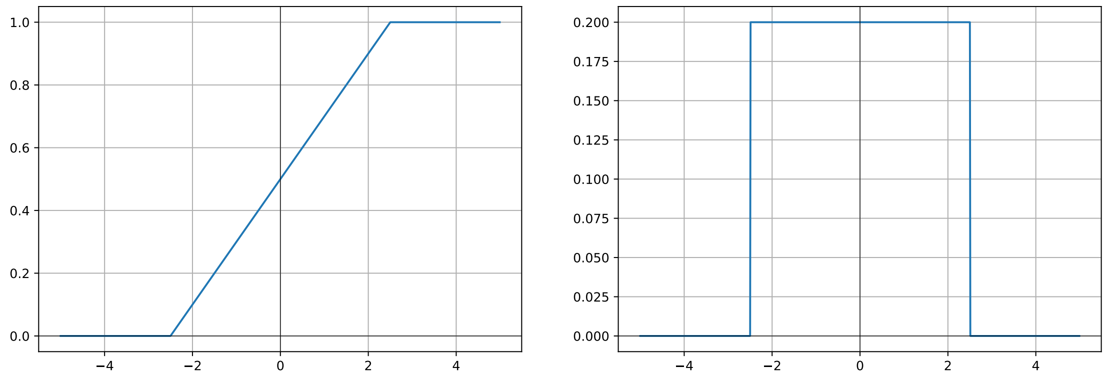
TODO: *koji problem rješava*

TODO: *svojstva*

TODO: *problemi*

4.2.19. Tvrda sigmoida

(engl. *Hard sigmoid*)



Slika 4.14: Tvrda sigmoida i njena derivacija

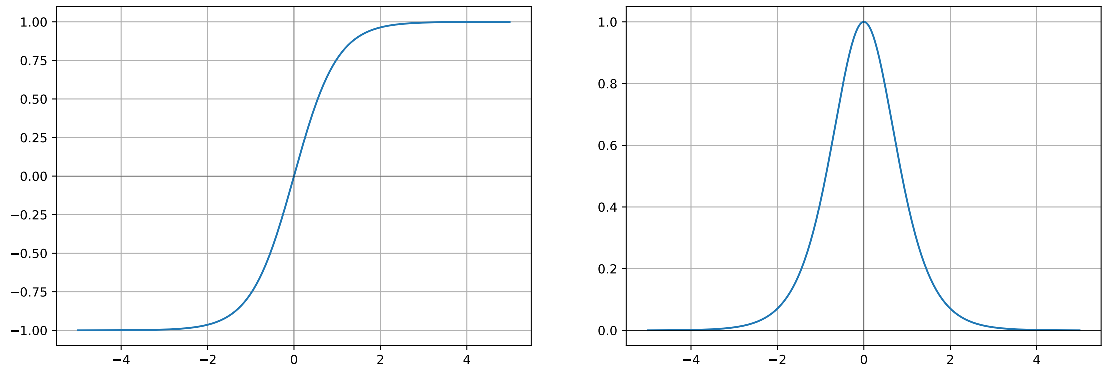
$$f(x) = min(1,\ max(0,\ 0.2x + 0.5)) \qquad f'(x) = \begin{cases} 0.2, & \text{ako } |x| \leq 2.5 \\ 0, & \text{inače} \end{cases} \tag{4.21}$$

TODO: *koji problem rješava*

TODO: *svojstva*

TODO: *problemi*

4.2.20. Tangens hiperbolni (tanh)



Slika 4.15: Funkcija tanh i njena derivacija

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \qquad f'(x) = 1 - tanh^2(x) \tag{4.22}$$

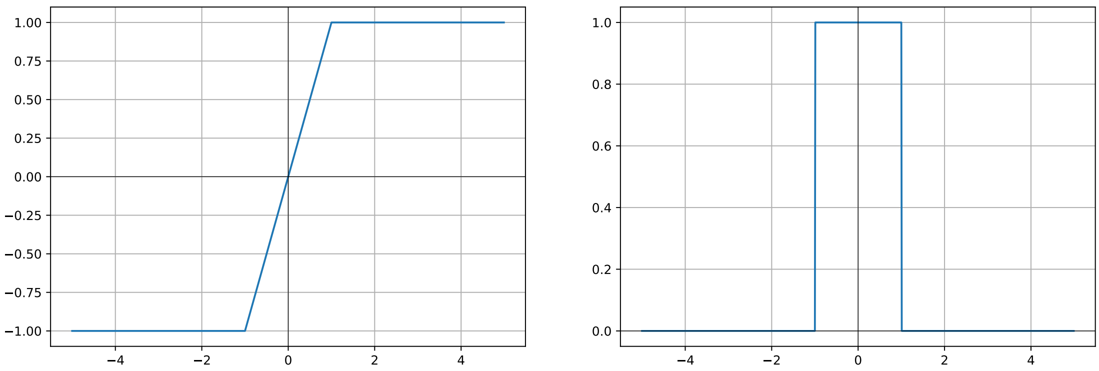
TODO: *koji problem rješava*

TODO: *svojstva*

TODO: *problemi*

4.2.21. Tvrdi tangens hiperbolni

(engl. *Hard tanh*)



Slika 4.16: Tvrdi tanh i njegova derivacija

$$f(x) = \begin{cases} -1, & \text{ako } x < 1 \\ x, & \text{ako } |x| \leq 1 \\ 1, & \text{inače} \end{cases} \qquad f'(x) = \begin{cases} 0, & \text{ako } x < 1 \\ 1, & \text{ako } |x| \leq 1 \\ 0, & \text{inače} \end{cases} \tag{4.23}$$

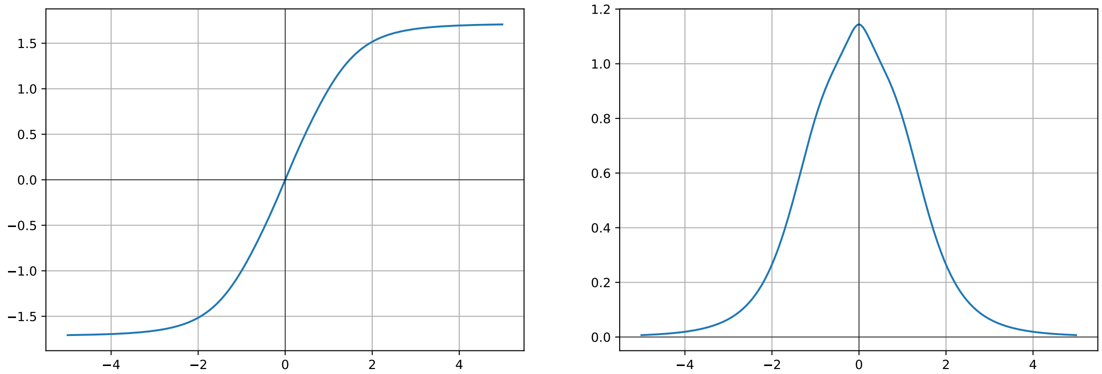
TODO: *koji problem rješava*

TODO: *svojstva*

TODO: *problemi*

4.2.22. Racionalna aproksimacija tanh

(engl. *Rational tanh*)



Slika 4.17: Racionalni tanh i njegova derivacija

$$f(x) = 1.7159 \cdot tanh^*(\frac{2}{3}x), \quad tanh^*(x) = sgn(x)(1 - \frac{1}{1 + |x| + x^2 + 1.41645 \cdot x^4})$$
$$f'(x) = 1.7159 \cdot \frac{2}{3} \cdot tanh^{*'}(\frac{2}{3}x), \quad tanh^{*'}(x) = \frac{1 + sgn(x) \cdot (2x + 4 \cdot 1.41645 \cdot x^3)}{(1 + |x| + x^2 + 1.41645 \cdot x^4)^2} \tag{4.24}$$

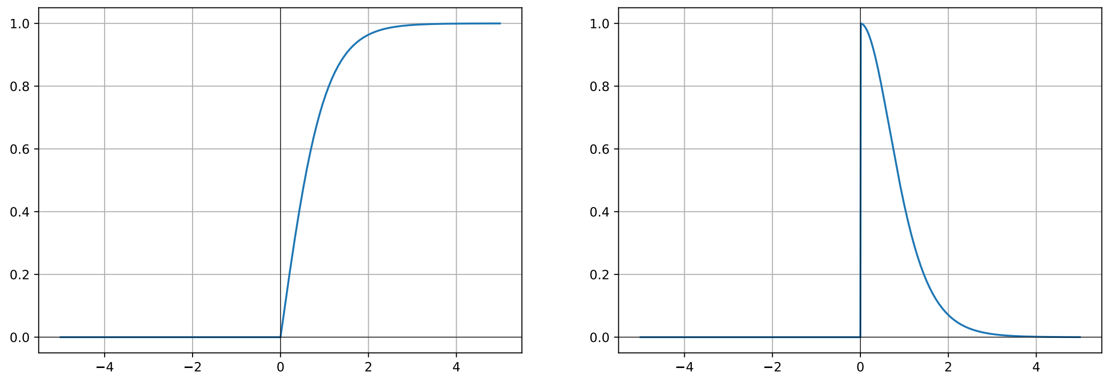
TODO: *koji problem rješava*

TODO: *svojstva*

TODO: *problemi*

4.2.23. Ispravljeni tanh

(engl. *Rectified tanh*)



Slika 4.18: Ispravljeni tanh i njegova derivacija

$$??? \tag{4.25}$$

TODO: *koji problem rješava*

TODO: *svojstva*

TODO: *problemi*

4.2.24. Softmax

[IMAGE:]

$$f(\vec{x}) = \frac{e^{\vec{x}}}{\sum_i e^{\vec{x}_i}} \qquad f'(x) = \frac{e^x}{1 + e^x} \tag{4.26}$$

TODO: *koji problem rješava*

TODO: *svojstva*

TODO: *problemi*

4.2.25. Hierarchical softmax

[IMAGE:]

$$??? \tag{4.27}$$

TODO: *koji problem rješava*

TODO: *svojstva*

TODO: *problemi*

4.2.26. Maxout

[IMAGE:]

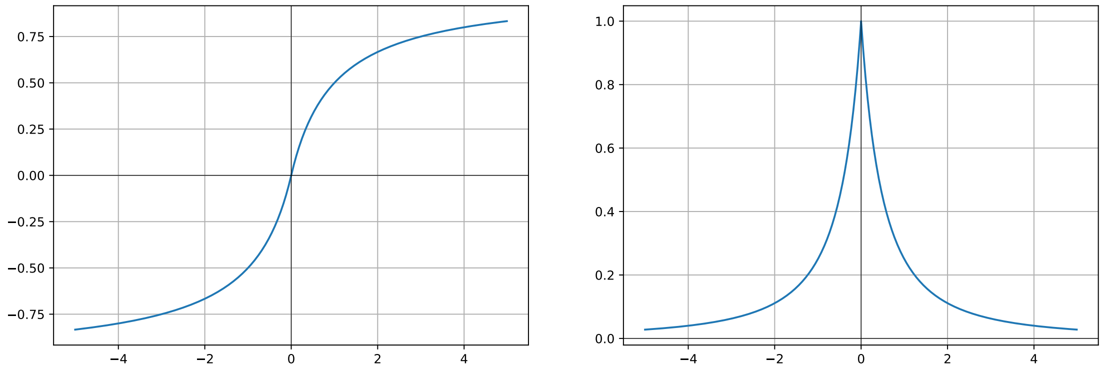
$$??? \tag{4.28}$$

TODO: *koji problem rješava*

TODO: *svojstva*

TODO: *problemi*

4.2.27. Softsign



Slika 4.19: Softsign i njegova derivacija

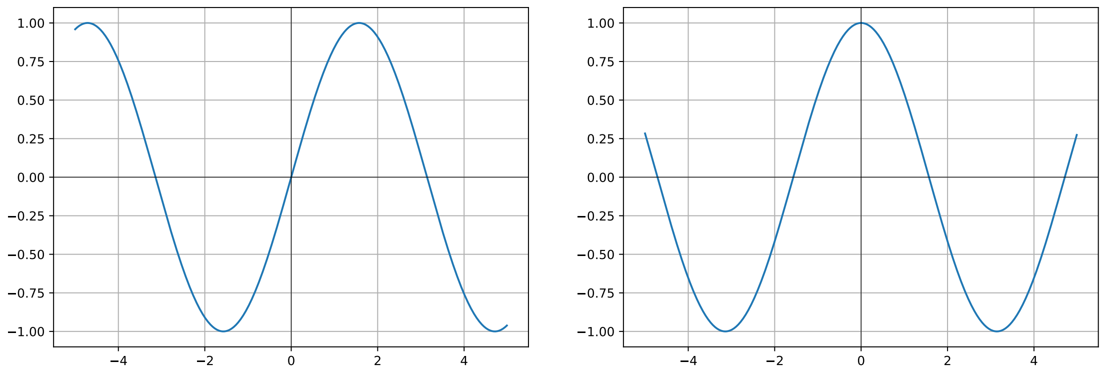
$$f(x) = \frac{x}{1+|x|} \qquad f'(x) = \frac{1+|x| - x \cdot sign(x)}{(1+|x|)^2} \tag{4.29}$$

TODO: *koji problem rješava*

TODO: *svojstva*

TODO: *problemi*

4.2.28. Sinus (sin)



Slika 4.20: Sinusoida i njegova derivacija

$$f(x) = sin(x) \qquad f'(x) = cos(x) \tag{4.30}$$

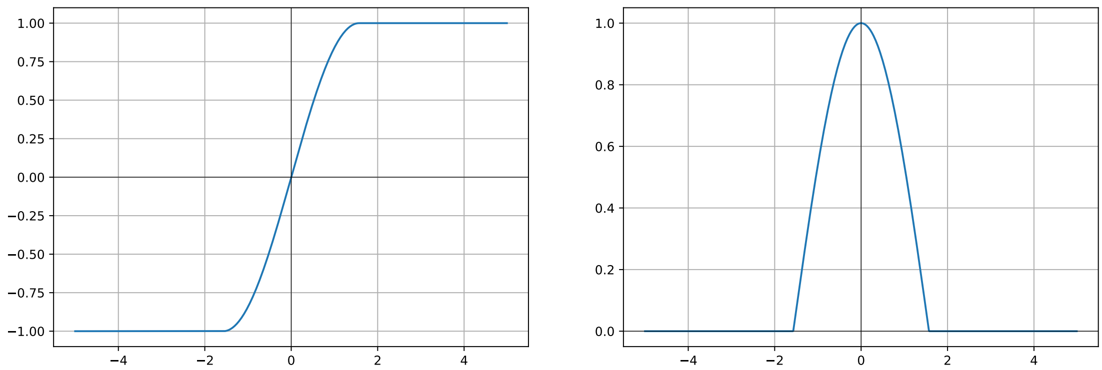
Sinus je periodička funkcija te nema teorijsku podlogu za uporabu u neuronskim mrežama, za razliku od monotonih funkcija. TODO: *članak dokaz univerzalne aproksimacije monotonih fja*

Unatoč tome, pojavljuje se u radovima pretrage izlaznih funkcija (Basirat i Roth, 2018) i u ovom radu te pokazuje obećavajuće rezultate. U radu Parascandolo et al. (2017) autori predstavljaju problematiku učenja sa sinusoidom na jednostavnom problemu. Problem stvaraju brojni lokalni optimumi na koje je izrazito osjetljiv gradijentni spust. Problem je ublažava učenje stohastičkim gradijentnim spustom koji zaglađuje valovitost funkcije gubitka. Dodatno, autori pokazuju da neuronska mreža zapravo ne ovisi snažno o periodičnosti funkcije tako da su rezultate usporedili sa skraćenom sinusoidom (**??**). Članak nije prošao postupak recenzije jer su dodatni eksperimenti urodili nekonzistentnim rezultatima u usporedbi s tangensom hiperbolnim.

Po uzoru na slične funkcije kao što je sigmoida (slika 4.13) i tangens hiperbolni (slika 4.15), sinusoida najveću vrijednost gradijenta daje upravo kada je aktivacija jednaka 0 (u ograničenom intervalu).

4.2.29. Ograničeni sinus (TrSin)

(engl. *Truncated sine*)

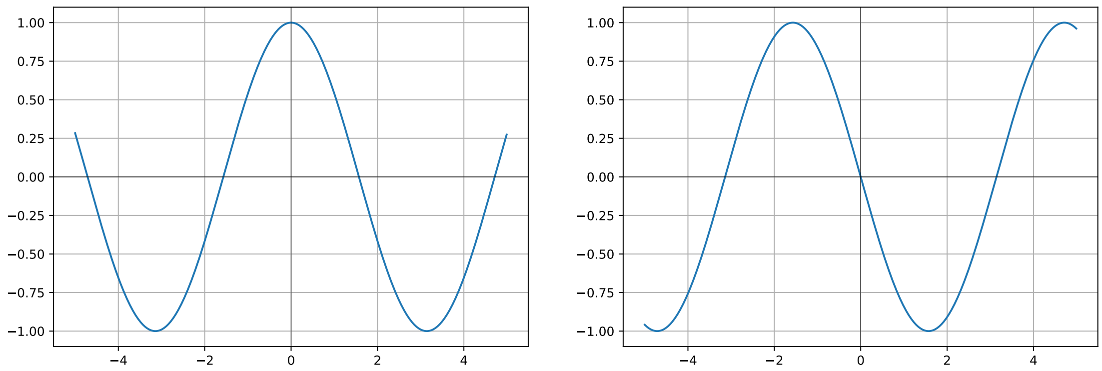


Slika 4.21: Ograničeni sinus i njegova derivacija

$$f(x) = \begin{cases} 0, & x < \frac{\pi}{2} \\ sin(x), & |x| \leq \frac{\pi}{2} \\ 1, & inače \end{cases} \qquad f'(x) = \begin{cases} cos(x), & |x| \leq \frac{\pi}{2} \\ 0, & inače \end{cases} \tag{4.31}$$

Ograničeni sinus korišten je u radu Parascandolo et al. (2017) za usporedbu s klasičnom sinusoidom i tangensom hiperbolnim. U usporedbi sa sinusom ispituje se utjecaj periodičnosti sinusa na performanse. S tangensom hiperbolnim uspoređene su performanse zbog sličnosti u obliku krivulja.

4.2.30. Kosinus (cos)



Slika 4.22: Kosinus i njegova derivacija

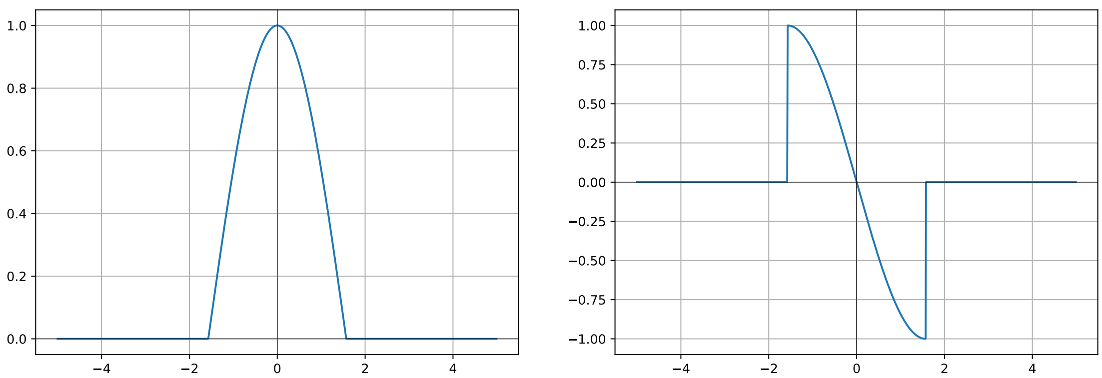
$$f(x) = cos(x) \qquad f'(x) = -sin(x) \tag{4.32}$$

Kosinus je funkcija sinusa pomaknuta za četvrtinu periode te dijeli ista svojstva i probleme. no, s obzirom da je pomak relativno velik u odnosu na očekivane veličine ulaza, možemo ga promatrati kao zasebnu izlaznu funkciju. Derivacija kosinusa se poprilično razlikuje od ostalih aktivacijskih funkcija. Unatoč tome, daje vrlo kompetitivne rezultate (poglavlje 7).

$$cos(x) = sin(x + \frac{\pi}{2}) \approx sin(x + 1.571) \tag{4.33}$$

4.2.31. Ograničeni kosinus (TrCos)

(engl. *Truncated cosine*)

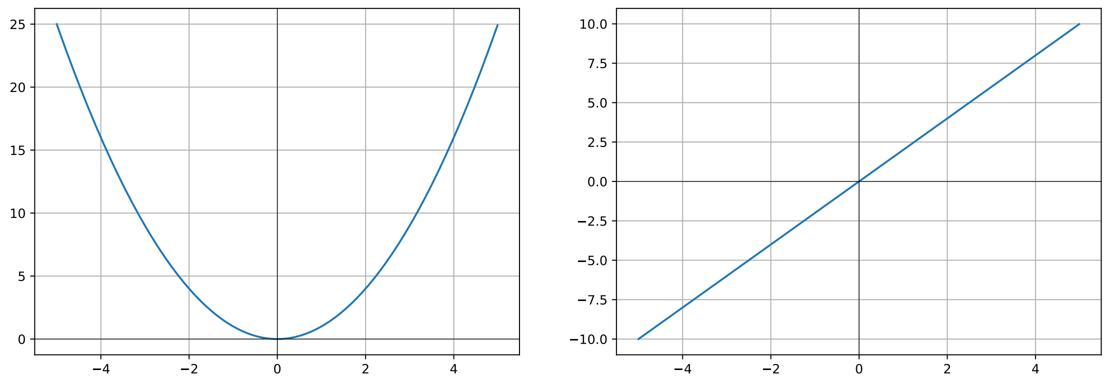


Slika 4.23: Ograničeni kosinus i njegova derivacija

$$f(x) = \begin{cases} cos(x), & |x| \leq \frac{\pi}{2} \\ 0, & inače \end{cases} \qquad f'(x) = \begin{cases} -sin(x), & |x| \leq \frac{\pi}{2} \\ 0, & inače \end{cases} \tag{4.34}$$

Ograničeni kosinus autori nisu pronašli u literaturi, no navodi se zbog potrebe ispitivanja ovisnosti o periodičnosti kosinusa (po uzoru na poglavlje 4.2.29).

4.2.32. Parabola x^2



Slika 4.24: Parabola i njena derivacija

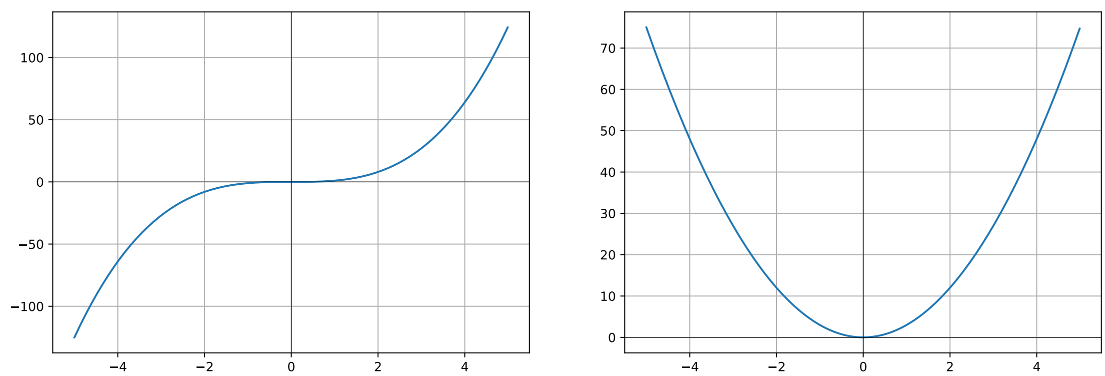
$$f(x) = x^2 \qquad f'(x) = 2x \tag{4.35}$$

TODO: *koji problem rješava*

TODO: *svojstva*

TODO: *problemi*

4.2.33. Kubna parabola x^3



Slika 4.25: Kubna parabola i njena derivacija

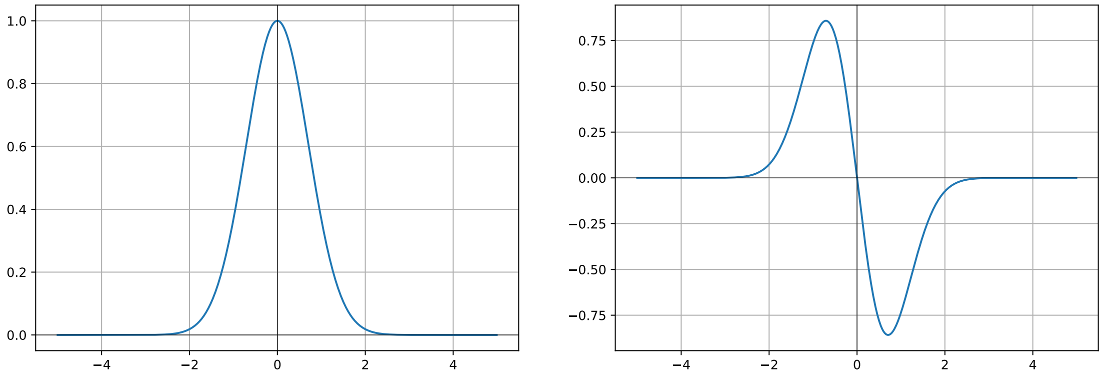
$$f(x) = x^3 \qquad f'(x) = 3x^2 \tag{4.36}$$

TODO: *koji problem rješava*

TODO: *svojstva*

TODO: *problemi*

4.2.34. Gauss



Slika 4.26: Gaussolika funkcija i njena derivacija

$$f(x) = e^{-x^2} \qquad f'(x) = -2x \cdot f(x) \tag{4.37}$$

TODO: *koji problem rješava*

TODO: *svojstva*

TODO: *problemi*

4.3. Izlazne funkcije posebne namjene

4.3.1. (DReLU)

(engl. *Dual ReLU*)

[IMAGE:]

$$??? \tag{4.38}$$

TODO: *koji problem rješava*

TODO: *svojstva*

TODO: *problemi*

Koristi se u rekurzivnim neuronskim mrežama kako bi se

5. Optimizacija simboličkom regresijom (tehnički genetskim programiranjem...)

5.1. Simbolička regresija

TODO: *Opis i svojstva SR*

TODO: *Utjecaj i brojnost parametara u GA (moš linkat i svoj završni rad :P)*

5.1.1. Pretraživanje izlaznih funkcija

TODO: *golden activation članak, kako radi detaljnije*

TODO: *swish članak detaljnije*

5.1.2. Neuronska mreža kao funkcija

TODO: *Kako predstaviti čitavu mrežu grafom*

TODO: *CGP Atari rad*

TODO: *Spomeni da je to kao evolucija prijenosne funkcije (i ulazna i izlazna se definiraju)*

5.2. Taboo evolucijski algoritam

TODO: *Problem konvergencije i stohastičnosti GP-a*

TODO: *EA oplemenjen taboo listom iz algoritma Taboo pretraživanja*

Naziv	Funkcija	Broj ulaza
x	x	0
const	$c \in \mathbb{R}$	0
+	$x + y$	2
-	$x - y$	2
*	$x \cdot y$	2
/	$\frac{x}{y+1e-12}$	2
min	$min(x, y)$	2
max	$max(x, y)$	2
abs	$\|x\|$	1
sin	$sin(x)$	1
cos	$cos(x)$	1
tan	$tan(x)$	1
exp	e^x	1
log	$log_e(x)$	1
pow2	x^2	1
pow3	x^3	1
pow	x^y	2

Naziv	Funkcija	Broj ulaza
elu	$ELU(x)$	1
gauss	e^{x^2}	1
lrelu	$LReLU(x)$	1
relu	$ReLU(x)$	1
selu	$SELU(x)$	1
sigmoid	$\frac{1}{1+e^{-x}}$	1
softmax	$Softmax(x)$	1
sotplus	$Softplus(x)$	1
softsign	$Softsign(x)$	1
swish	$Swish(x)$	1
tanh	$tanh(x)$	1

Tablica 5.1: Popis korištenih čvorova

5.3. Korišteni čvorovi (prostor pretraživanja)

U nastavku je dan popis čvorova koji ujedno definiraju prostor pretraživanja.

6. Implementacija

6.1. Razvojna okolina i alati

TODO: *tulavi DL4J*

TODO: *IntelliJ <3 <3 <3*

TODO: *funkcije iscrtavam u jn*

6.2. Parametri

TODO: *da, možeš definirat parametre u datoteki*

TODO: *dodavanje parametara*

TODO: *grid search mehanizam*

6.3. Evolucijski algoritmi

TODO: *in-house EA okolina*

TODO: *glavne komponente koda*

[IMAGE: *IntelliJ generirna UML paketa genetics*]

6.4. Neuronske mreže

TODO: *glavne komponente*

TODO: *automatizirana pohrana rezultata*

[IMAGE: *IntelliJ generirna UML paketa neurology*]

6.5. Paralelizacija

TODO: *workarbiter <3*

TODO: *sinkronizacija u GA i pomoćne klase/metode*

6.6. Loggovi

TODO: *kaj sve imaš i kak se koristi*

7. Rezultati

7.1. 9class

7.1.1. Uobičajene izlazne funkcije

TODO: *Opis postupka pretrage*

TODO: *Tablica*

TODO: *Komentar*

7.1.2. Utjecaj parametra veličine taboo liste

TODO: *Tablica*

TODO: *Komentar*

[IMAGE: *boxplot*]

7.2. 256class

7.2.1. Uobičajene izlazne funkcije

TODO: *Opis postupka pretrage*

TODO: *Tablica*

TODO: *Komentar*

7.2.2. Utjecaj parametra veličine taboo liste

TODO: *Tablica*

TODO: *Komentar*

[IMAGE: *boxplot*]

8. Stvari koje sam probao, ali nisu ispale korisne

Učeći parametri

TODO: *dokaz da na korištene funkcije nema utjecaja (stopi se s težinama ili biasom)*
TODO: *pokazati primjer fje gdje bi se mogao koristiti*

Dropout

TODO: *zahtjeva previše iteracija, što nije baš korisno u EA okruženju*
TODO: *probati maxout?*

Tensorflow Java API

TODO: *probo, ali je još u razvoju (puno toga je falilo)*

9. Buduća istraživanja

TODO: *Primjena CNN na vremenskim uzorcima po uzoru na onaj rad*
TODO: *Ispitivanje učinkovitosti korištene optimizacije na ostalim problemima*
TODO: *Paralelna evolucija arhitekture i aktivacijskih fja (Suganuma et al., 2017)*

10. Zaključak

TODO: *Radi/Ne radi*

TODO: *Pronađene zanimljivosti*

TODO: *Pouka za doma*

LITERATURA

Mina Basirat i Peter M. Roth. The quest for the golden activation function. *CoRR*, abs/1808.00783, 2018.

Peter Dayan i L. F. Abbott. *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. The MIT Press, 2005. ISBN 0262541858.

Włodzisław Duch i Norbert Jankowski. Survey of neural transfer functions. *Neural Computing Surveys*, 2(1):163–212, 1999. URL `ftp://ftp.icsi.berkeley.edu/pub/ai/jagota/vol2_6.pdf`.

Wlodzislaw Duch i Norbert Jankowski. Taxonomy of neural transfer functions. U *IJCNN*, 2000.

Wlodzislaw Duch i Norbert Jankowski. Transfer functions: hidden possibilities for better neural networks. U *ESANN*, 2001.

Ömer Faruk Ertugrul. A novel type of activation function in artificial neural networks: Trained activation function. *Neural networks : the official journal of the International Neural Network Society*, 99:148–157, 2018.

Meng Fang, Yuan Li, i Trevor Cohn. Learning how to active learn: A deep reinforcement learning approach. U *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, stranice 595–605, Copenhagen, Denmark, Rujan 2017. Association for Computational Linguistics. doi: 10.18653/v1/D17-1063. URL `https://www.aclweb.org/anthology/D17-1063`.

Xavier Glorot, Antoine Bordes, i Yoshua Bengio. Deep sparse rectifier neural networks. U Geoffrey Gordon, David Dunson, i Miroslav Dudík, urednici, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, svezak 15 od *Proceedings of Machine Learning Research*, stranice 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR. URL `http://proceedings.mlr.press/v15/glorot11a.html`.

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

K. He, X. Zhang, S. Ren, i J. Sun. Deep residual learning for image recognition. U *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, stranice 770–778, June 2016. doi: 10.1109/CVPR.2016.90.

G. Huang, Z. Liu, L. v. d. Maaten, i K. Q. Weinberger. Densely connected convolutional networks. U *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, stranice 2261–2269, July 2017. doi: 10.1109/CVPR.2017.243.

Yoon Kim. Convolutional neural networks for sentence classification. U *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, stranice 1746–1751, Doha, Qatar, Listopad 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1181. URL https://www.aclweb.org/anthology/D14-1181.

Alex Krizhevsky. Convolutional deep belief networks on cifar-10. 2010.

Alex Krizhevsky, Ilya Sutskever, i Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. U F. Pereira, C. J. C. Burges, L. Bottou, i K. Q. Weinberger, urednici, *Advances in Neural Information Processing Systems 25*, stranice 1097–1105. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-network pdf.

Min Lin, Qiang Chen, i Shuicheng Yan. Network in network. *CoRR*, abs/1312.4400, 2014.

Andrew L Maas, Awni Y Hannun, i Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models.

Tomas Mikolov, Kai Chen, Greg S. Corrado, i Jeffrey Dean. Efficient estimation of word representations in vector space, 2013. URL http://arxiv.org/abs/1301.3781.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, i Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL http://arxiv.org/abs/1312.5602.

Kevin P Murphy. *Machine learning: a probabilistic perspective*. Cambridge, MA, 2012.

Vinod Nair i Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. U *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, stranice 807–814, USA, 2010. Omnipress. ISBN 978-1-60558-907-7. URL `http://dl.acm.org/citation.cfm?id=3104322.3104425`.

Giambattista Parascandolo, Heikki Huttunen, i Tuomas Virtanen. Taming the waves: sine as activation function in deep neural networks. 2017. unpublished.

J. Redmon, S. Divvala, R. Girshick, i A. Farhadi. You only look once: Unified, real-time object detection. U *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, stranice 779–788, June 2016. doi: 10.1109/CVPR.2016.91.

Rupesh Kumar Srivastava, Klaus Greff, i Jürgen Schmidhuber. Highway networks. *CoRR*, abs/1505.00387, 2015.

Masanori Suganuma, Shinichi Shirakawa, i Tomoharu Nagao. A genetic programming approach to designing convolutional neural network architectures. U *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '17, stranice 497–504, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4920-8. doi: 10.1145/3071178.3071229. URL `http://doi.acm.org/10.1145/3071178.3071229`.

C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, i A. Rabinovich. Going deeper with convolutions. U *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, stranice 1–9, June 2015. doi: 10.1109/CVPR.2015.7298594.

Marko Čupić, Bojana Dalbelo Bašić, i Marin Golub. *Neizrazito, evolucijsko i neuroračunarstvo*. Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu, (2013-08-12) izdanju, 2013.

Optimizirane izlazne funkcije klasifikatora temeljenog na umjetnim neuronskim mrežama u domeni implementacijskih napada na kriptografske uređaje

Sažetak

Proučiti postojeće metode u izgradnji izlaznih funkcija u umjetnim neuronskim mrežama. Posebnu pažnju posvetiti evolucijskim algoritmima simboličke regresije za izgradnju ciljanih funkcija. Ustanoviti moguće nedostatke postojećih algoritama ili mogućnost poboljšanja. Primijeniti evoluirane izlazne funkcije u homogenoj ili heterogenoj umjetnoj neuronskoj mreži na skupovima DPAv2 i DPAv4 te odrediti mjere kvalitete izgrađenog klasifikatora: točnost, preciznost, odziv te F mjere. Usporediti učinkovitost ostvarenih postupaka s postojećim rješenjima iz literature. Radu priložiti izvorne tekstove programa, dobivene rezultate uz potrebna objašnjenja i korištenu literaturu.

Ključne riječi: Ključne riječi, odvojene zarezima.

Optimized output functions for classifiers based on artificial neural networks in the domain of implementation attacks on cryptographic devices

Abstract

Examine existing methods in building output functions in artificial neural networks. Give special attention to evolutionary algorithms of symbolic regression for constructing target functions. Apply evolved output functions in a homogeneous or heterogeneous artificial neural network on datasets DPAv2 and DPAv4 and examine quality measures of the built classifier: accuracy, precision, recall and F measures. Compare the efficiency of acquired methods with existing solutions from the literature. Alongside thesis attach source code of programs, acquired results with necessarry discussion and literature used.

Keywords: Keywords.