

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1966

**Optimizirane izlazne funkcije
klasifikatora temeljenog na
umjetnim nevronskim 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	1
2. Implementacijski napadi na kriptografske uređaje	2
2.1. Side-channel napadi	2
2.2. Izvedba napada	2
2.3. DPA skupovi podataka	2
2.3.1. DPAv2	2
2.3.2. DPAv4	3
3. Klasifikator temeljen na umjetnim neuronskim mrežama	5
3.1. Umjetne neuronske mreže	5
3.1.1. Građa	5
3.1.2. Optimizacija umjetne neuronske mreže	7
3.1.3. Regularizacija	17
3.1.4. Odabir hiperparametara	19
3.1.5. Svojstva	23
3.1.6. Problemi	24
4. Izlazne funkcije	25
4.1. Odabir izlazne funkcije	25
4.1.1. Izlazne funkcije s učećim parametima	26
4.1.2. Periodične izlazne funkcije	26
4.2. Popularne izlazne funkcije	26
4.2.1. Funkcija identiteta	26
4.2.2. Zglobnica ili ispravljena linearna jedinica (ReLU)	27
4.2.3. Propusna ispravljena linearna jedinica (LReLU)	28
4.2.4. Ispravljena linearna jedinica s pragom (ThReLU)	29
4.2.5. Softplus	30

4.2.6. Eksponencijalno-linearna jedinica (ELU)	30
4.2.7. Skalirana eksponencijalno-linearna jedinica (SELU)	31
4.2.8. Swish	32
4.2.9. ELiSH	32
4.2.10. Tvrdi ELiSH	33
4.2.11. Ograničena ispravljena linearna jedinica (ReLU-n)	34
4.2.12. Sigmoida (σ)	35
4.2.13. Tangens hiperbolni (tanh)	36
4.2.14. Racionalna aproksimacija tanh	36
4.2.15. Ispravljeni tanh	37
4.2.16. Softmax	37
4.2.17. Softsign	38
4.2.18. Sinus (sin)	38
4.2.19. Ograničeni sinus (TrSin)	39
4.2.20. Kosinus (cos)	40
4.2.21. Parabola x^2	40
4.2.22. Gaussova krivulja bez normalizacije	41
5. Optimizacija izlazne funkcije genetskim programiranjem	42
5.1. Genetsko programiranje	42
5.1.1. Simbolička regresija	42
5.1.2. Neuroevolucija genetskim programiranjem	42
5.2. Pretraživanje izlaznih funkcija	43
5.2.1. Algoritam	43
5.2.2. Skup čvorova (prostor pretraživanja)	43
5.2.3. Operatori križanja	44
5.2.4. Operatori mutacije	45
6. Implementacija	48
6.1. Razvojna okolina i alati	48
6.2. Organizacija koda	48
6.2.1. Evolucijski algoritmi	48
6.2.2. Neuronske mreže	50
6.2.3. Pomoćni mehanizmi za projekt	50
6.2.4. Izvršni programi projekta	51

7. Rezultati	52
7.1. DPAv4	52
7.1.1. Usporedba uobičajenih izlaznih funkcija	52
7.1.2. Izgradnja izlaznih funkcija simboličkom regresijom	52
7.1.3. Izgradnja heterogenog rasporeda izlaznih funkcija	52
7.2. DPAv2	52
7.2.1. Usporedba uobičajenih izlaznih funkcija	52
7.2.2. Izgradnja izlaznih funkcija simboličkom regresijom	59
7.2.3. Izgradnja heterogenog rasporeda izlaznih funkcija	59
8. Stvari koje sam probao, ali nisu ispale korisne	60
9. Buduća istraživanja	61
10. Zaključak	62
Literatura	63

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**

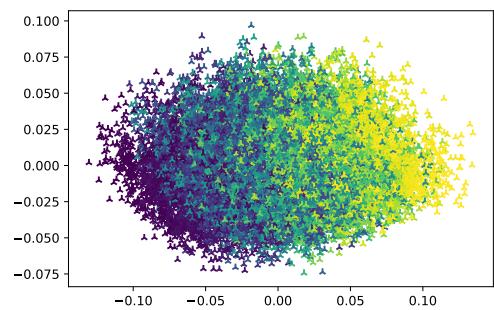
TODO: *Ne zaboravi referencu na stranicu!*

2.3.1. DPAv2

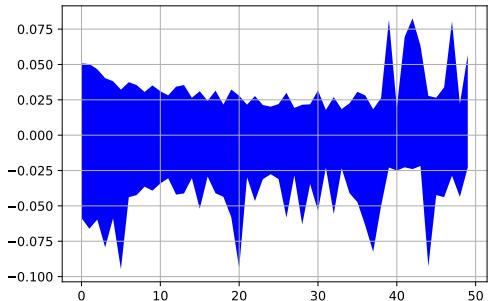
TODO: *Kad je napravljen i ko ga je radil*



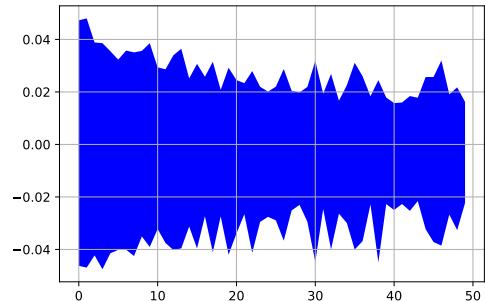
(a) PCA redukcija skupa za učenje



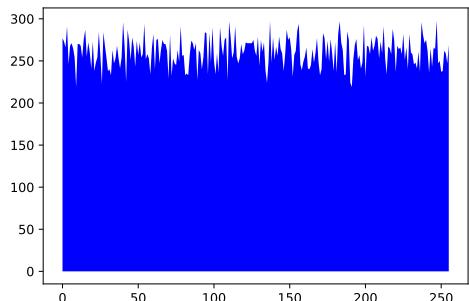
(b) PCA redukcija skupa za testiranje



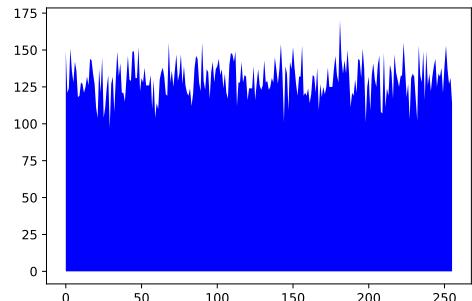
(c) Distribucija značajki skupa za učenje



(d) Distribucija značajki skupa za testiranje



(e) Distribucija oznaka skupa za učenje



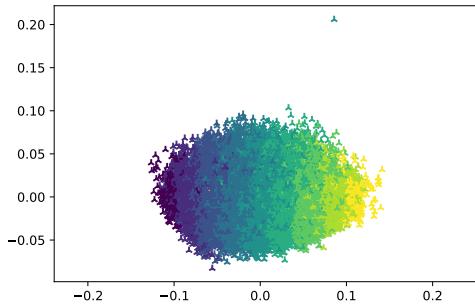
(f) Distribucija oznaka skupa za testiranje

Slika 2.1: Statistike podskupa za učenje i testiranje skupa DPAv2

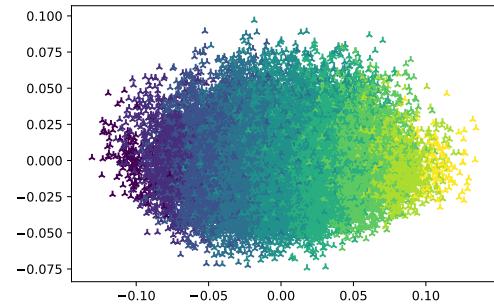
TODO: *Mjere dobrote klasifikacije*

2.3.2. DPAv4

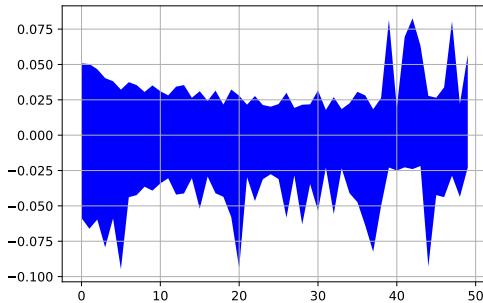
TODO: *Kada je napravljen i ko ga je radilo*



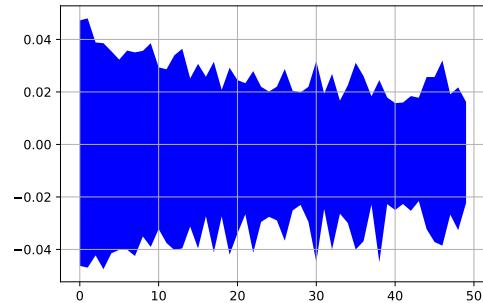
(a) PCA redukcija skupa za učenje



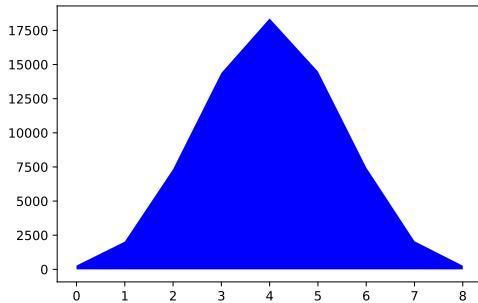
(b) PCA redukcija skupa za testiranje



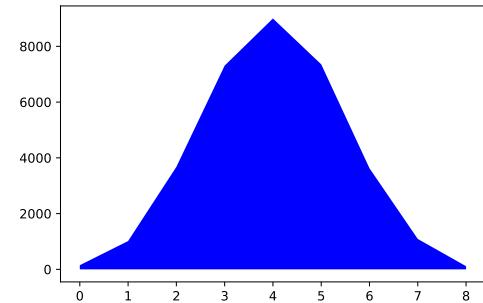
(c) Distribucija značajki skupa za učenje



(d) Distribucija značajki skupa za testiranje



(e) Distribucija oznaka skupa za učenje



(f) Distribucija oznaka skupa za testiranje

Slika 2.2: Statistike podskupa za učenje i testiranje skupa DPAv4

TODO: Mjere dobrote klasifikacije

3. Klasifikator temeljen na umjetnim neuronskim mrežama

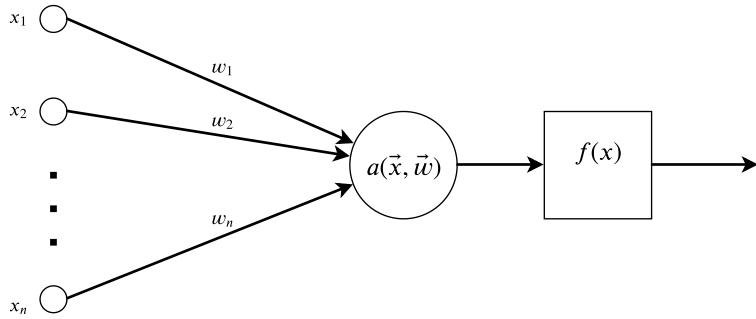
3.1. Umjetne neuronske mreže

Umjetne neuronske mreže (nadalje „neuronske mreže“) koristimo za modeliranje višedimenzijске 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.

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 poglavljju 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 funkciju ne treba miješati s funkcijom izlaznog sloja koja predstavlja nelinearnost posljednjeg sloja mreže.



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.

U literaturi postoje nekonzistencije oko pojma izlazne funkcije te se vrlo često i s pravom koristi pojam aktivacijske funkcije. U preglednim znanstvenim radovima (Duch i Jankowski, 1999, 2000, 2001) pojam aktivacijske funkcije predstavlja ulaznu funkciju, što je neispravno sa stajališta računalne neuroznanosti jer aktivacija neurona označava pojavu signala odasланог из some u akson (Dayan i Abbott, 2005, p. 234). U starijoj literaturi pojam aktivacijske i prijenosne funkcije često se koristi ekvivalentno (Xin Yao, 1999). 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).

$$t(x) = (f \circ a)(x) = f(a(x)) \quad (3.1)$$

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. Primjenjuje se kada se ulazi u model mogu zapisati vektorom značajki čiji raspored nije bitan.

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

Unakrsna korelacija, za razliku od afine funkcije, koristi informaciju o susjednosti ulaznih značajki. Unakrsna korelacija vrlo je slična operatoru konvolucije te se u literaturi gotovo uvijek naziva konvolucijom. Često se koristi za ekstrakciju obrazaca u slikama (Krizhevsky et al., 2012), no može se koristiti u problemima koji zahtjevaju obravu slijedova podataka. Jedan takav primjer je klasifikacija teksta u kome različit slijed riječi može prenositi različito značenje (Kim, 2014).

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 mijerenje 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. 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 nelinearno mapiranje značajki iz dimenzije prethodnog sloja u novu dimenziju s ciljem modeliranja boljih značajki.

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 prepostavku derivabilnosti svih komponenata neuronske mreže. Kada ta prepostavka 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 pronašlaska 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 uvjeta 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 pronašlak 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  $\eta$ 
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: pronađena optimalna točka  $\vec{x}_i$ 
```

Algorithm 1: Gradijentni spust

[IMAGE: *gradijentni spust stope spuštanja (velika, 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 pomicući 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 3.3). Moment može pomoći i pri zaobilaženju lokalnih optimuma (slika 3.1.2).

$$\begin{aligned}\vec{v} &\leftarrow \alpha \cdot \vec{v} - \eta \cdot \vec{g} \\ \vec{x} &\leftarrow \vec{x} + \vec{v}\end{aligned}\tag{3.3}$$

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

Algoritam je primjenjiv na funkcije proizvoljne dimenzionalnosti uz pretpostavku derivabilnosti u svakoj točki. Za proizvoljnu realnu funkciju, uz dobro odabране 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** (3.4). Logaritam je uveden zbog pojednostavljivanja distribucija koje sadrže eksponente.

$$L(\theta | x, y) = -\log p(f(X; \theta) = y | X = x) \quad (3.4)$$

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

$$\hat{y} = f(x; \theta) \quad (3.5)$$

S obzirom da želimo procijeniti kvalitetu modela na čitavom podatkovnom skupu zanima nas očekivanje funkcije gubitka (3.22).

$$\begin{aligned} L(\theta | \mathcal{D}) &= \mathbb{E}_{(x,y) \sim p_{\mathcal{D}}} [L(\theta | x, y)] \\ &= -\frac{1}{|\mathcal{D}|} \sum_{(x,y) \in \mathcal{D}} \log p(\hat{y} = y | x) \end{aligned} \quad (3.6)$$

U problemima regresije, prepostavlja se da izlazi mreže prate Gaussovu distribuciju s jediničnom kovarijacijskom matricom. Njenim uvrštavanjem u negativnu log. izglednost (3.4) dobivamo funkciju kvadratnog gubitka koja računa odstupanje izlaza neuronske mreže od željenih vrijednosti. Uz ovaj gubitak najčešće se koristi funkcija identiteta u izlaznom sloju.

$$\begin{aligned} p(\hat{y} = y | x) &= \mathcal{N}(y | \mu = \hat{y}, \Sigma = I) \\ &= \frac{1}{\sqrt{2\pi}\Sigma^{1/2}} \exp(-(y - \hat{y})^T \Sigma^{1/2} (y - \hat{y})) \end{aligned} \quad (3.7)$$

Uvrštavanjem u (3.4) dobivamo kvadratni gubitak:

$$\begin{aligned} L(\theta | x, y) &= -\log \left[\frac{1}{\sqrt{2\pi}\Sigma^{1/2}} \right] - \log \left[\exp(-(y - \hat{y})^T \Sigma^{1/2} (y - \hat{y})) \right] \\ &= -\log c - (y - \hat{y})^T \Sigma^{1/2} (y - \hat{y}) \\ &= \left| \begin{array}{l} c = \text{konst.} \\ \Sigma = I \end{array} \right| \\ &= (y - \hat{y})^T (y - \hat{y}) = \sum_i (y_i - \hat{y}_i)^2 \end{aligned} \quad (3.8)$$

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

$$p(\hat{y} = y \mid x) = \hat{y}_1^{y_1} \cdot \hat{y}_2^{y_2} \cdots \hat{y}_c^{y_c} = \prod_{i=1}^C \hat{y}_i^{y_i}, \quad \sum_i \hat{y}_i = 1 \quad (3.9)$$

Uvrštanjem u 3.4 dobivamo kategorički gubitak:

$$L(\theta \mid x, y) = -\log \left[\prod_{i=1}^C \hat{y}_i^{y_i} \right] = -\sum_{i=1}^C y_i \log(\hat{y}_i) \quad (3.10)$$

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.

$$\theta^* = \operatorname{argmin}_{\theta} L(\theta \mid \mathcal{D}), \quad \forall (x, y) \in \mathcal{D} \quad (3.11)$$

Optimiranje gradijentnim spustom zahtjeva derivabilnost funkcije koju optimiziramo po ulazima, što izrazi (3.8) i (3.10) zadovoljavaju. Pri tome koristi se pravilo ulančavanja parcijalne derivacije kompozicije funkcija.

$$\frac{d}{dx} f(g(x)) = \frac{\partial f(g(x))}{\partial g(x)} \cdot \frac{\partial g(x)}{\partial x} \quad (3.12)$$

Derivacijom funkcije gubitka za regresiju (3.8) po ulazima, uz prepostavku derivabilnosti čitave neuronske mreže po ulazima, dobivamo sljedeći izraz:

$$\begin{aligned} \frac{dL(\theta \mid x, y)}{dx} &= \frac{d}{dx} \sum_i (y_i - \hat{y}_i(x))^2 \\ &= 2 \cdot \sum_i (y_i - \hat{y}_i(x)) \cdot \frac{d\hat{y}_i(x)}{dx} \end{aligned} \quad (3.13)$$

Derivaciju funkcije gubitka za klasifikaciju možemo drastično pojednostaviti ako za funkciju izlaznog sloja mreže odaberemo funkciju *softmax* (3.14), a izlaze kodiramo *one-hot* oznakama (3.15). Ulaz u funkciju softmax su aktivacije neurona izlaznog sloja koje su ovdje označene vektorom $\vec{s}(x)$.

$$\hat{y}(\vec{x}) = \text{softmax}(\vec{s}(x)) = \frac{\exp(\vec{s}(x))}{\sum_{i=1}^C \exp(s_i(x))} \quad (3.14)$$

$$\vec{y} = [y_1, y_2, \dots, y_C], \quad y_i \in \{0, 1\}, \quad \sum_{i=1}^C y_i = 1 \quad (3.15)$$

Uvrštavanjem u (3.10) dobivamo:

$$\begin{aligned} L(\theta | x, y) &= - \sum_{i=1}^C y_i \log \frac{\exp(s_i(x))}{\sum_{j=1}^C \exp(s_j(x))} \\ &= - \sum_{i=1}^C \left[y_i \cdot s_i(x) - y_i \log \left(\sum_{j=1}^C \exp(s_j(x)) \right) \right] \\ &= - \sum_{i=1}^C [y_i \cdot s_i(x)] + \log \left[\sum_{j=1}^C \exp(s_j(x)) \right] \cdot \sum_{i=1}^C y_i \\ &= \log \left[\sum_{j=1}^C \exp(s_j(x)) \right] - \sum_{i=1}^C [y_i \cdot s_i(x)] \end{aligned} \quad (3.16)$$

Derivacijom (3.16) po ulazima, koristeći pravilo (3.12) dobivamo:

$$\begin{aligned} \frac{dL(\theta | x, y)}{dx} &= \frac{dL(\theta | x, y)}{d\vec{s}(x)} \cdot \frac{d\vec{s}(x)}{dx} \\ &= [\vec{s}'(x) - y] \cdot \frac{ds(x)}{dx} \end{aligned} \quad (3.17)$$

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:

$$\begin{aligned} \frac{\partial L(x, y; \theta)}{\partial s(x; w)} &= \frac{\partial L(x, y; \theta)}{\partial o(x; w)} \cdot \frac{\partial o(x; w)}{\partial s(x; w)} \\ &= \frac{\partial L(x, y; \theta)}{\partial o(x; w)} \cdot f'(x) \end{aligned} \quad (3.18)$$

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

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

$$\begin{aligned}\frac{\partial L(x, y; \theta)}{\partial w_0} &= \frac{\partial L(x, y; \theta)}{\partial s(x; w)} \cdot \frac{\partial s(x; w)}{\partial w_0} \\ &= \frac{\partial L(x, y; \theta)}{\partial o(x; w)} \cdot f'(x) \cdot 1\end{aligned}\tag{3.21}$$

Stohastički gradijentni spust

U prošlom poglavlju koristili smo funkciju gubitka na jednom podatkovnom paru, no želimo minimizirati očekivanje funkcije gubitka (3.22) 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 (3.10) 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 (3.22) poprima oblik koji odgovara očekivanju pojedine mini-grupe \mathcal{M}_i .

$$\begin{aligned} L(\theta \mid \mathcal{D}) &= \mathbb{E}_{(x,y) \sim p_{\mathcal{M}_i}} [L(\theta \mid x, y)] \\ &= -\frac{1}{|\mathcal{M}_i|} \sum_{(x,y) \in \mathcal{M}_i} \log p(\hat{y} = y \mid x) \end{aligned} \quad (3.22)$$

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: početni parametar  $\vec{\theta}_0$ 
Input: stopa učenja  $\eta$ 
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
Result: pronađeni optimalni parametar  $\vec{\theta}^*$ 
```

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 (3.3).

TODO: uporaba momenta i momenta na kvadrat (interpretacija)

TODO: Adam

Promjenjiva stopa učenja

Tijekom optimizacije gradijentnim spustom uz fiksnu stupu učenja optimizacija će početi oscilirati. Oscilacija se javlja kada je funkcija gubitka generalno konveksna po regijama, no sadrži

TODO: zašto je problem u stopi

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*]

Jedan način smanjivanja stope je zadavanje ključnih iteracija u kojima se prestaje koristiti stara stopa i koristi se nova. Moguće je zadati stope za svaku od zadanih ključnih iteracija *TODO: fali tekst (nađi onaj rad)*

Ključnim iteracijama možemo proglašiti one u kojima je detektirana konvergencija ili divergencija. Taj pristup omogućava nastavak pretrage, no problem je kako sa sigurnošću detektirati konvergenciju ili divergenciju. Tipično učenje mreže nije monotono, tj. gubitak mreže može na neko vrijeme rasti prije no što nastavi padati, što odgovara ponašanju u lokalnom optimumu. U tom slučaju smanjivanje stope učenja je nepoželjno.

TODO: Zašto zapravo smanjujemo stopu? Zato kaj je oština brda tamo dolje opasna za malu regiju u kojoj je optimum (vjerojatnost pogotka je malena)?

Pristup koji se koristi u ovom radu je zadanu početnu stopu u svakoj iteraciji pomožiti s konstantom manjom od 1. Pri tome se ne prepostavlja postojanje ključnih iteracija i stopa se konstantno smanjuje. Uz korištenje stohastičkog gradijentnog spusta

postupak navodi pretragu na sve finije regije funkcije gubitka. Postupak povlači sličnost s postupkom simuliranog kaljenja gdje se smanjivanjem temperature sustava smanjuje njegova stohastičnost i postupak konvergira u sve finije regije rješenja.

TODO: Koristim relativnu razliku za detekciju konvergencije. Samo abs nije dobar jer nemam pojma koje su vrijednosti lossa pri optimumu niti znam koji je najmanji loss za koji gradijent isčezne. Pri rel. se zada delta koja predstavlja postotak od trenutnog optimuma za koji se pomaknuo. Ako je pomak manji od 1% trenutnog optimuma, to je kvg.

Inicijalizacija parametara

Inicijalizacija parametara mreže odgovara odabiru početne točke pri gradijentnom spustu. Dobrom inicijalizacijom možemo osigurati pronalazak boljeg optimuma te u kritičnim slučajevima i osigurati konvergenciju mreže. S obzirom da neuronske mreže posjeduju jasnu strukturu (što je posebno izraženo u slojevitim arhitekturama) inicijalizacija ima dodatne utjecaje. U potpuno povezanim slojevima želimo da svaki neuron vrši ekstrakciju značajki. Pri tome, u efikasnom sloju svaki će neuron stvoriti svoju značajku. Ako je više neurona inicijalizirano dovoljno slično, ti neuroni će naučiti izvlačiti iste značajke. Kažemo da je došlo do **koadaptacije neurona**.

Inicijalizacija slojevitim nenadziranim učenjem ograničenog Boltzmann-ovog stroja (engl. *restricted Boltzmann machine*) pokazuje se dobrom (Krizhevsky, 2010). Postupak nenadzirano trenira stohastičku mrežu sloj po sloj na ulaznim značajkama kako bi slojevi ostvarili dobru ekstrakciju viših značajki. Nakon treniranja, mreži se dodaje izlazni klasifikacijski sloj i mreža se nadzirano ugađa na skupu za učenje. Iako je ostvarena inicijalizacija vrlo dobra, postupak je vremenski zahtjevan. Uvođenjem

U ovom radu koristi se Xavier inicijalizacija koja parametre uzorkuje iz centrirane distribucije s varijancom koja ovisi o broju ulaznih neurona ili značajki. *TODO: Ja koristim Xavier*

Preprocesiranje podataka

TODO: zašto normalizacija po značajkama

Koefficijenti se računaju pomoću značajki iz podatkovnog skupa za učenje. Koefficijent varijance ovisi o koefficijentu sredine pa se njihovo računanje ne može paraleli-

zirati, no to ne predstavlja velik problem s obzirom da ih računamo samo jedom pri učitavanju skupa.

$$\vec{\mu} = \frac{1}{N} \sum_{\vec{x} \in \mathcal{D}} \vec{x} \quad (3.23)$$

$$\vec{\sigma}^2 = \frac{1}{N-1} \sum_{\vec{x} \in \mathcal{D}} (\vec{x} - \vec{\mu})^2 \quad (3.24)$$

Prije no što ih predamo modelu, ulazne značajke normaliziramo formulom (??).

$$\vec{z} = \frac{\vec{x} - \vec{\mu}}{\vec{\sigma}^2} \quad (3.25)$$

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

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

TODO: komentar na sliku

3.1.3. Regularizacija

Decizijska granica opisuje točke u prostoru značajki za koje model dvoji između dviju ili više klasa. Geometrijski, decizijsku granicu možemo interpretirati kao presjek dviju ili više ploha u hiperprostoru. Pri optimizaciji model prilagođava decizijsku granicu kako bi što bolje razvrdorio primjere iz podatkovnog skupa koji su različitih klasa, a obuhvatio primjere iste klase. Podatkovni skupovi najčešće sadrže šum zbog nesavršenog uzorkovanja stvarne funkcije, pogrešnog označavanja ili više značnosti 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**. Na prenaučenost su posebno osjetljivi složeni modeli kao što su neuronske mreže.

[IMAGE: *podnaučena, generalizira, prenaučena*]

Kako bismo spriječili prenaučenost ili ju barem ublažili, koristimo tehnike regularizacije. Pri tome ne smijemo pretjerati jer suviše snažna regularizacija može ograničiti sposobnost učenja te model postaje **podnaučen**. Stoga je potrebno pronaći balans između složenosti modela i jačine regularizacije za koji model dobro **generalizira**. Pri uporabi neuronskih mreža obično se odabire složen model na koji se primjenjuju razne tehnike regularizacije.

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 .

$$\tilde{L}(\theta | x, y) = L(\theta | x, y) + \alpha\Omega(\theta) \quad (3.26)$$

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.

$$\Omega(\vec{\theta}) = \frac{1}{2}\|\vec{w}\|_2^2 = \frac{1}{2}\sum_i w_i^2 \quad (3.27)$$

$$\vec{\theta}_{i+1} \leftarrow \vec{\theta}_i - \eta \cdot \vec{g}_i + \alpha \cdot \theta \quad (3.28)$$

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

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 novo treniramo model, ovdje ga treniramo jednom i jednostavno odaberemo vrijednost za koju model generalizira najbolje. To upućuje na potrebu krosvalidacije koja je opisana u poglavljju ??.

[EQUATION: Algoritam ranog zaustavljanja] (3.29)

Normalizacija nad grupom

(engl. *batch normalization*)

TODO: što radi

TODO: kako

TODO: zašto umjesto dropousta

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 ??). 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 4.2.13. U poglavlju 4 navedene su i opisane brojne funkcije te je njihov utjecaj ispitana 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** (engl. *crossvalidation*), a skupove nazivamo **skupom za učenje** (engl. *train set*) i **skupom za testiranje** (engl. *test set*). 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. Postoji više postupaka mjerjenja generalizacije modela (k-preklopa, LOOCV, ...) koji osiguravaju da je generalizacija ispitana na svim primjerima, no zbog postojanja službenog skupa za testiranje i zahtjevnosti ostalih postupaka mjerjenja u ovom se radu koristi **metoda izdvajanja** (engl. *holdout*).

Pri odabiru hiperparametara ili pri odabiru modela potrebno je usporediti dobrote

njihovih rezultata. Pri tome se skup za učenje ponovno podijeli na skup za učenje i **skup za validaciju** (engl. *validation set*). Skupom za učenje optimiziramo model, a skupom za validaciju ispitujemo generalizaciju modela. Između kombinacija hiperparametara odabiremo onu za koju model najbolje generalizira na skupu za validaciju te njome nanovo optimiziramo model na originalnom skupu za učenje i dobivamo konačnu generalizaciju na skupu za testiranje. Navedeni postupak odgovara algoritmu **ugniježđene unakrsne provjere** (engl. *nested crossvalidation*). U ovom radu se umjesto unakrsne validacije k-preklopa koristi metoda izdvajanja u unutarnjoj i vanjskoj petlji, a postupak je opisan pseudokodom 3.

```

Input: model  $m$ 
Input: podatkovni skup  $\mathcal{D}$ 
Input: kombinacije vrijednosti hiperparametara  $\mathcal{K}$ 
Podijeli  $\mathcal{D}$  na  $\mathcal{D}_{train}$  i  $\mathcal{D}_{test}$ 
Podijeli  $\mathcal{D}_{train}$  na  $\mathcal{D}_{train'}$  i  $\mathcal{D}_{val}$ 
for svaku kombinaciju  $k_i \in \mathcal{K}$  do
    Inicijaliziraj  $m$  Optimiziraj  $m$  hiperparametrima  $k_i$  na  $\mathcal{D}_{train'}$ 
    Izmjeri generalizaciju  $g_i$  na  $\mathcal{D}_{val}$ 
end
 $k^* \leftarrow argmax_{g_i} k_i$ 
Inicijaliziraj  $m$  Optimiziraj  $m$  hiperparametrima  $k^*$  na  $\mathcal{D}_{train}$ 
Izmjeri konačnu generalizaciju  $g$  na  $\mathcal{D}_{test}$ 
Result: pronađeni optimalni model  $m^*$ 
```

Algorithm 3: Ugniježđena unakrsna provjera

Mjere generalizacije

Za različite vrste problema koristimo različite mjere. Iako je ukupna vrijednost funkcije gubitka na čitavom skupu za testiranje dobar pokazatelj, za probleme klasifikacije koristimo mjere koje detaljnije opisuju stvarne performanse modela.

$\hat{y} \setminus y$	\top	\perp	
\top	TP	FP	(3.30)
\perp	FN	TN	

Pri **binarnoj klasifikaciji** definiramo **matricu zabune** (3.30) koja sadrži četiri ele-

menta (3.37) koji definiraju vrstu pogodka i pogreške.

$$\begin{aligned}
 \text{Stvarno pozitivni: } TP &= \sum_{(x,y) \in \mathbb{D}} \mathbb{1}\{h(x) = \top \wedge y = \top\} \\
 \text{Stvarno negativni: } TN &= \sum_{(x,y) \in \mathbb{D}} \mathbb{1}\{h(x) = \perp \wedge y = \perp\} \\
 \text{Lažno pozitivni: } FP &= \sum_{(x,y) \in \mathbb{D}} \mathbb{1}\{h(x) = \top \wedge y = \perp\} \\
 \text{Lažno negativni: } FN &= \sum_{(x,y) \in \mathbb{D}} \mathbb{1}\{h(x) = \perp \wedge y = \top\}
 \end{aligned} \tag{3.31}$$

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

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.32}$$

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 ispravnog klasifikatora biti će nezamjetna. Stoga se kod nebalansiranih setova češće koristi **F₁ mjera**, koja uzima u obzir **preciznost** klasifikatora u razlikovanju pozitivnih primjera od negativnih (3.33) i njegov **odziv** odnosno obuhvat svih pozitivnih primjera testnog skupa (3.34). F₁ mjera je definirana kao harmonijska sredina između preciznosti i odziva (3.35). Postoji i generalizirana mjera F_β koja dodjeljuje veću težinu preciznosti ili odzivu (3.36), no u ovom radu koristi se samo F₁ koja pridijeljuje jednaku težinu. Harmonijska sredina se koristi jer je najstroža između Pitagorinih mjera za sredinu kao što prokazuje slika 3.2.

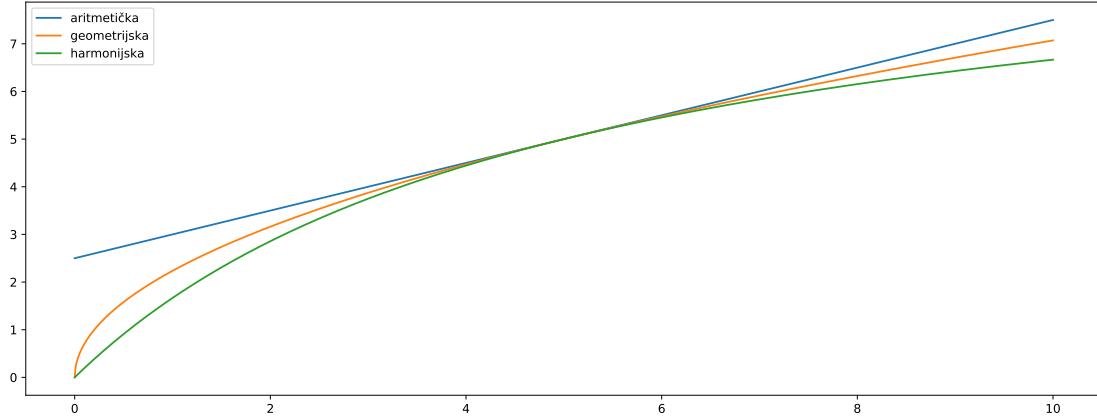
$$\text{Preciznost: } P = \frac{TP}{TP + FP} \tag{3.33}$$

$$\text{Odziv: } R = \frac{TP}{TP + FN} \tag{3.34}$$

$$F_1 : \quad F_1 = 2 \cdot \frac{P \cdot R}{P + R} \tag{3.35}$$

$$F_\beta : \quad F_\beta = (1 + \beta^2) \cdot \frac{P \cdot R}{\beta^2 \cdot P + R} \tag{3.36}$$

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



Slika 3.2: Pitagorine mjere za sredinu između dviju vrijednosti

slično kao i kod binarne klasifikacije, ali za svaku klasu posebno.

$$\begin{aligned}
 \text{Stvarno pozitivni: } & TP_i = \sum_{(x,y) \in \mathbb{D}} \mathbb{1}\{h(x) = C_i \wedge y = C_i\} \\
 \text{Stvarno negativni: } & TN_i = \sum_{(x,y) \in \mathbb{D}} \mathbb{1}\{h(x) \neq C_i \wedge y \neq C_i\} \\
 \text{Lažno pozitivni: } & FP_i = \sum_{(x,y) \in \mathbb{D}} \mathbb{1}\{h(x) = C_i \wedge y \neq C_i\} \\
 \text{Lažno negativni: } & FN_i = \sum_{(x,y) \in \mathbb{D}} \mathbb{1}\{h(x) \neq C_i \wedge y = C_i\}
 \end{aligned} \tag{3.37}$$

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** prvo se 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).

$$Acc^M = \sum_{i=1}^K \frac{Acc_i}{K} \quad P^M = \sum_{i=1}^K \frac{P_i}{K} \quad R^M = \sum_{i=1}^K \frac{R_i}{K} \quad F_1^M = \sum_{i=1}^K \frac{F_{1;i}}{K} \tag{3.38}$$

Mikro prosjekom prvo se 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.

$$\begin{aligned}
 Acc^\mu &= \frac{\sum_i (TP_i + TN_i)}{\sum_i (TP_i + TN_i + FP_i + FN_i)} = Acc^M \\
 FP = FN \implies P^\mu = R^\mu = F_1^\mu &= \frac{\sum_i TP_i}{\sum_i (TP_i + FP_i)}
 \end{aligned} \tag{3.39}$$

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 funkciju proizvoljno dobro uz nekoliko uvjeta: 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).

Reprezentacija dubinom

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 garantija 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 kao računalnog programa koje nadilaze temu ovog rada. (Goodfellow et al., 2016)

TODO: *VC dimenzija (Sontag, 1998)

Generalizacija

[IMAGE: underfit-fit-overfit]

Model sa svojstvom generalizacije dobro modelira stvarnu funkciju te ispravno radi na neviđenim primjerima.

[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.

3.1.6. Problemi

Odabir hiperparametara

Arhitektura, prijenosne funkcije i parametri definiraju neuronsku mrežu te njihov pravilan odabir značajno utjeće 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 Srivastava et al. (2015), 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 poglavljju 4.

Nestajući gradijent

TODO: Problem dubokih arhitektura

4. Izlazne funkcije

Izlazna funkcija služi unošenju nelinearnosti u neuronsku mrežu i ima utjecaj na njen učenje. Prilikom inferencije izlazna funkcija stvara nelinearnosti u decizijskoj granici koje su parametrizirane parametrima neurona na koji se primjenjuje. U slojevitim arhitekturama izlazne funkcije se primjenjuju 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 W . Uzastopnom primjenom 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 primjenjuju 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 linearu 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. Periodične izlazne funkcije

TODO: *članak: fourier mreža s periodičkim fjama*

Periodičke aktivacijske funkcije stvaraju brojne lokalne optimume u funkciji gubitka koji otežavaju učenje neuronskih mreža. S obzirom da mreže treniramo stohastičkim gradijentnim spustom većina lokalnih optimuma nije vidljiva i ublažava naglašenost lokalnih optimuma. Pri uporabi sinusoide autori pokazuju da će model konvergirati samo ako su težine inicijalizirane dovoljno malim vrijednostima da ulaz ne prelazi interval monotonosti sinusa $[-\frac{\pi}{2}, \frac{\pi}{2}]$. U usporedbi sinusoide s ograničenom sinusoidom na potpuno povezanom modelu pokazuje se nezamjetna razlika u performansama iako oko 40% ulaza neurona završava izvan intervala monotonosti, što upućuje da modeli ne ovise o periodičnosti funkcije. U primjeni na rekurzivnim modelima ne ostvaruje konzistentnu prednost nad monotonim tangensom hiperbolnim zbog čega članak nije prošao postupak recenzije. (Parascandolo et al., 2017)

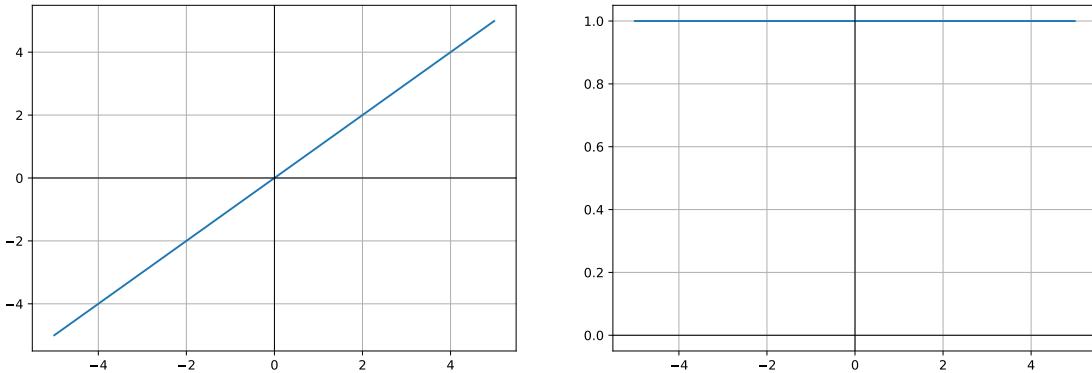
TODO: *ostale reference iz taming the waves*

4.2. Popularne izlazne funkcije

U nastavku su navedene pasivne izlazne funkcije koje je autor pronašao u literaturi i koje su isprobane u ovom radu. Funkcije su pasivne u smislu da ne sadrže učeće parametre te njihov izlaz ovisi isključivo o ulazu u funkciju. 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

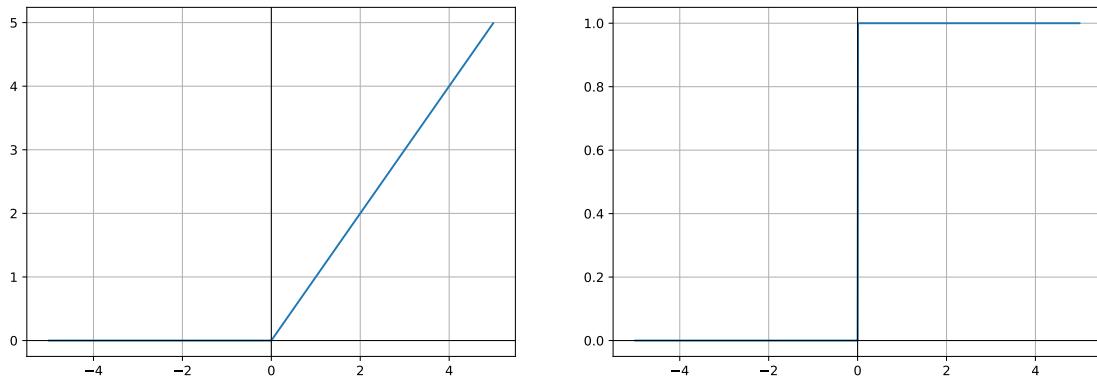
$$f(x) = x \quad f'(x) = 1 \quad (4.1)$$

Funkcija identiteta je jednostavna i brza, no njome neuronska mreža može naučiti samo linearne funkcije. Ako se primjeni na dva uzastopna sloja neuronske mreže, konačna će funkcija ponovno biti linearна što znači da se mreža ne može koristiti na nelinearnim podatcima. Derivacija funkcije je konstanta 1 što znači da je gradijent prolaskom kroz nju očuvan. Funkcija identiteta često se koristi kao funkcija izlaznog sloja kod regresije, no u dubokim arhitekturama (He et al., 2016; Huang et al., 2017) i novijim izlaznim funkcijama (?Basirat i Roth, 2018) koristi se paralelno s nelinearnim funkcijama.

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

4.2.2. Zglobnica ili ispravljenja linearna jedinica (ReLU)

(engl. *Rectified linear unit*)



Slika 4.2: Funkcija ReLU i njena derivacija

$$f(x) = \begin{cases} x, & \text{ako } x > 0 \\ 0, & \text{inače} \end{cases} \quad f'(x) = \begin{cases} 1, & \text{ako } x > 0 \\ 0, & \text{inače} \end{cases} \quad (4.3)$$

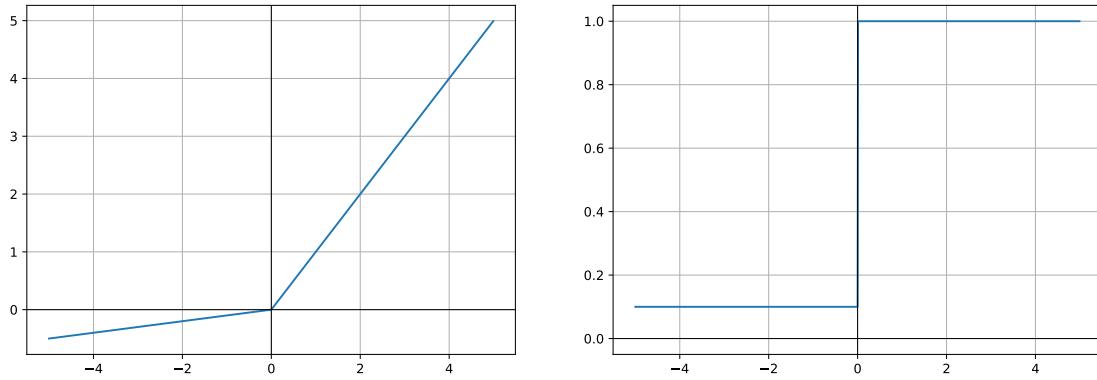
$$= \max(0, x)$$

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 eksponentijalno puno linearnih regija koje dijele zajedničke parametre (Nair i Hinton, 2010). Zahvaljujući linearnosti na pozitivnoj domeni funkcija ne doprinosi nestajanju 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 $\alpha = 0.1$

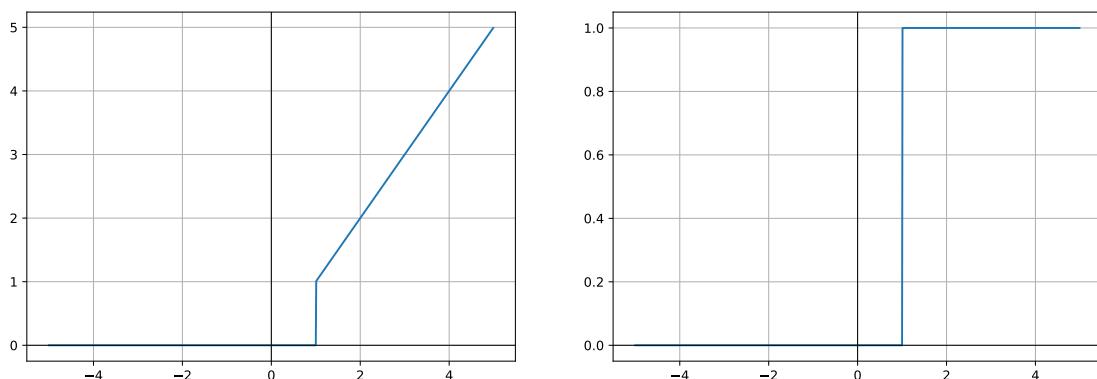
$$f(x) = \begin{cases} x, & \text{ako } x > 0 \\ \alpha x, & \text{inače} \end{cases} \quad f'(x) = \begin{cases} 1, & \text{ako } x > 0 \\ \alpha, & \text{inače} \end{cases} \quad (4.4)$$

$$= \max(\alpha x, x)$$

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 memoriskog 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. Ispravljena linearna jedinica s pragom (ThReLU)

(engl. *Thresholded ReLU*)



Slika 4.4: Funkcija ThReLU i njena derivacija

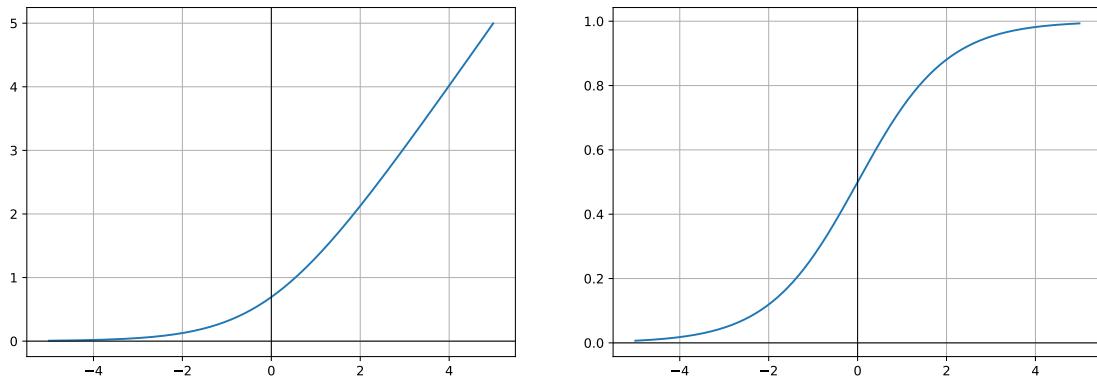
$$f(x) = \begin{cases} x, & \text{ako } x > \theta \\ 0, & \text{inače} \end{cases} \quad f'(x) = \begin{cases} 1, & \text{ako } x > \theta \\ 0, & \text{inače} \end{cases} \quad (4.5)$$

TODO: koji problem rješava

TODO: svojstva

TODO: problemi

4.2.5. Softplus



Slika 4.5: Softplus i njegova derivacija

$$f(x) = \log(1 + e^x) \quad f'(x) = \frac{e^x}{1 + e^x} \quad (4.6)$$

TODO: koji problem rješava

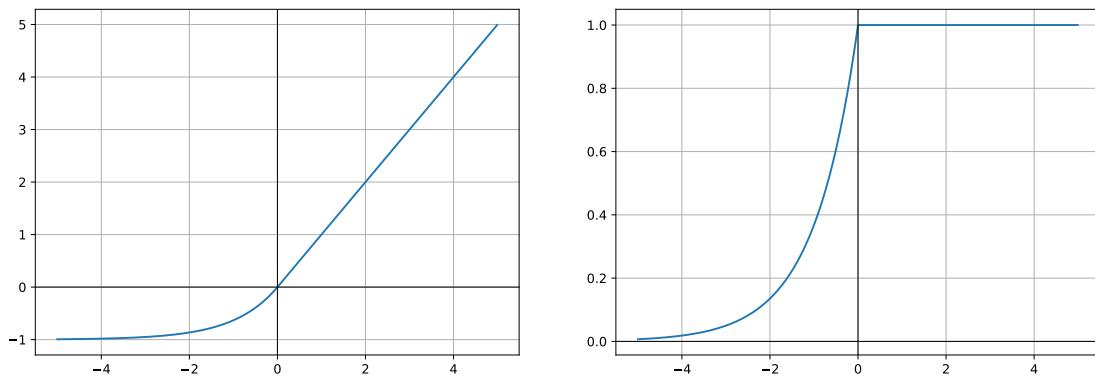
TODO: svojstva

TODO: problemi

Funkcija Softplus nastaje ...

4.2.6. Eksponencijalno-linearna jedinica (ELU)

(engl. Exponential linear unit)



Slika 4.6: Funkcija ELU i njena derivacija

$$f(x) = \begin{cases} x, & \text{ako } x > 0 \\ \alpha(e^x - 1), & \text{inače} \end{cases} \quad f'(x) = \begin{cases} 1, & \text{ako } x > 0 \\ \alpha e^x, & \text{inače} \end{cases} \quad (4.7)$$

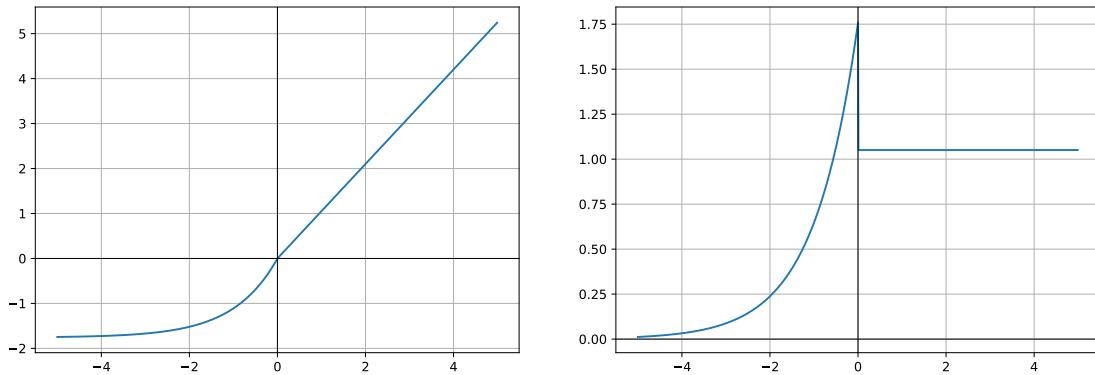
TODO: koji problem rješava

TODO: svojstva

TODO: problemi

4.2.7. Skalirana eksponencijalno-linearna jedinica (SELU)

(engl. *Scaled exponential linear unit*)



Slika 4.7: Funkcija SELU i njena derivacija

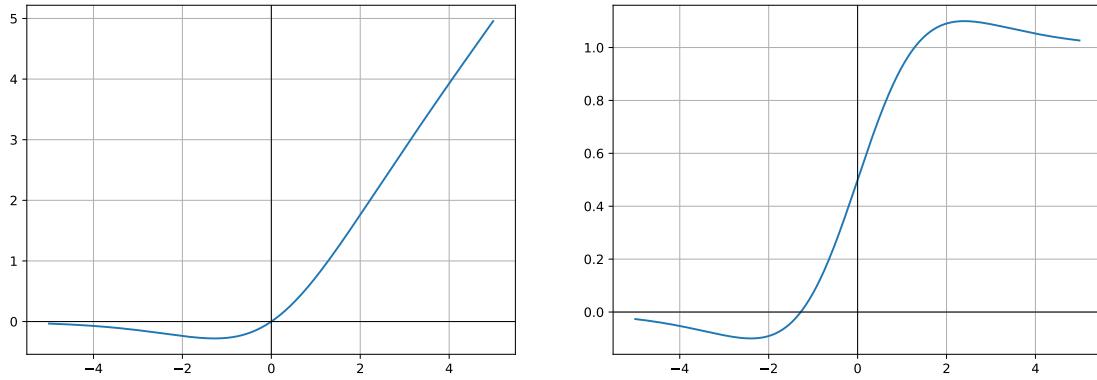
$$f(x) = \lambda \begin{cases} x, & \text{ako } x > 0 \\ \alpha(e^x - 1), & \text{inače} \end{cases} \quad f'(x) = \lambda \begin{cases} 1, & \text{ako } x > 0 \\ \alpha e^x, & \text{inače} \end{cases} \quad (4.8)$$

TODO: koji problem rješava

TODO: svojstva

TODO: problemi

4.2.8. Swish



Slika 4.8: Funkcija Swish i njena derivacija

$$\begin{aligned}f(x) &= x \cdot \sigma(\beta x) \\f'(x) &= \sigma(\beta x) \cdot (1 + \beta x \cdot (1 - \sigma(\beta x))) = \frac{e^x \cdot (e^x + x + 1)}{(e^x + 1)^2}\end{aligned}\tag{4.9}$$

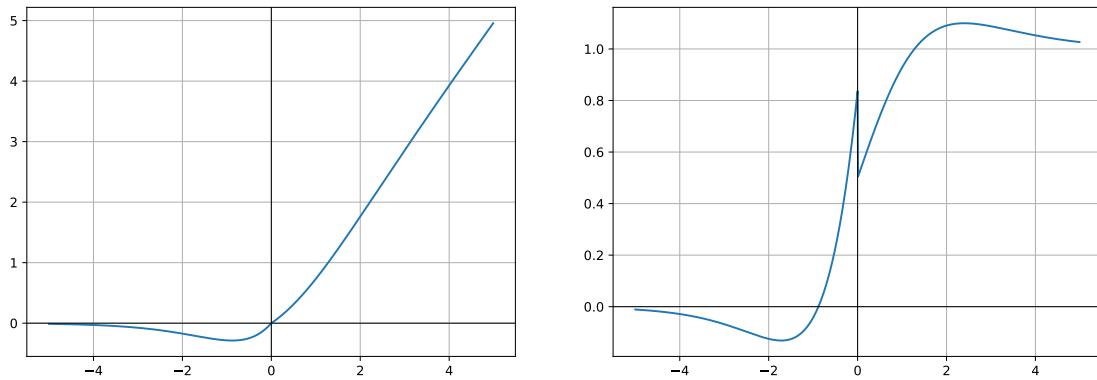
TODO: koji problem rješava

TODO: svojstva

TODO: problemi

4.2.9. ELiSH

(engl. *Exponential Linear Sigmoid Squashing*)



Slika 4.9: Funkcija ELiSH i njena derivacija

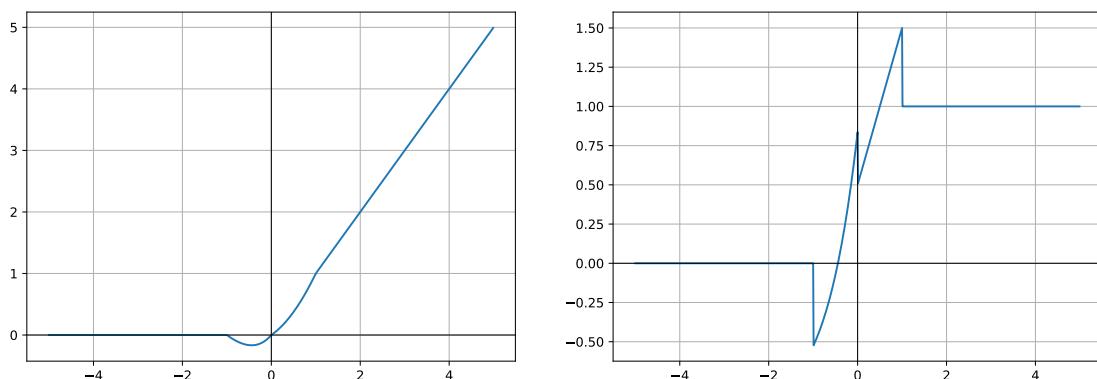
$$f(x) = \begin{cases} swish(x), & x \geq 0 \\ ELU(x) \cdot \sigma(x), & \text{inac}e \end{cases} \quad (4.10)$$

$$f'(x) = \begin{cases} swish'(x), & x \geq 0 \\ \sigma(x) \cdot (ELU'(x) + ELU(x) \cdot (1 - \sigma(x))), & \text{inac}e \end{cases}$$

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 (4.9), a negativna je umnožak funkcije ELU (4.7) i sigmoide (4.14). Obje funkcije su ručno izgrađene s ciljem iskorištanja dobrog prijenosa informacije koji ostvaruju funkcije Swish i sigmoida te sprečavanjem nestajućeg gradijenta pomoću linearne komponente u funkciji Swish. (Basirat i Roth, 2018)

4.2.10. Tvrdi ELiSH

(engl. *Hard ELiSH*)



Slika 4.10: Funkcija Tvrdi ELiSH i njena derivacija

$$f(x) = \begin{cases} x \cdot a(x), & x \geq 0 \\ ELU(x) \cdot a(x), & \text{inače} \end{cases} \quad (4.11)$$

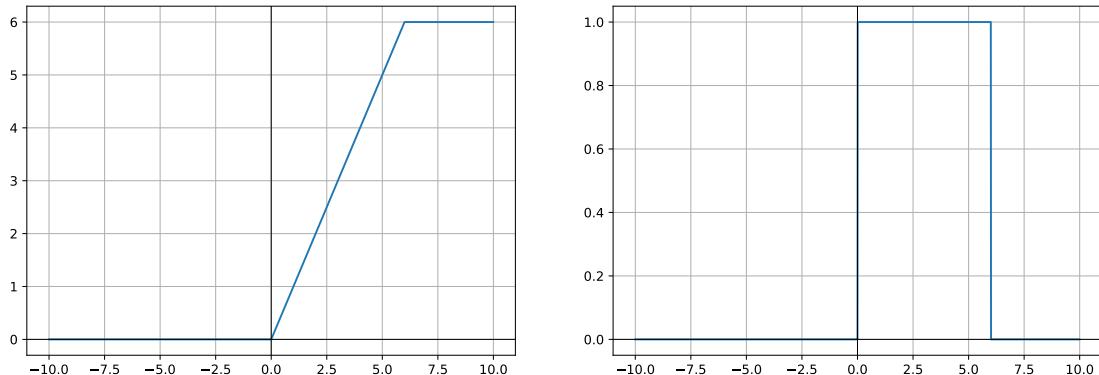
$$f'(x) = \begin{cases} a(x) + x \cdot a'(x), & x \geq 0 \\ ELU'(x) \cdot a(x) + x \cdot ELU(x) \cdot a'(x), & \text{inače} \end{cases}$$

$$a(x) = \max(0, \min(1, \frac{x+1}{2})) \approx HardSigmoid(x) \quad (4.12)$$

$$a'(x) = \begin{cases} 0.5, & |x| \leq 1 \\ 0, & \text{inače} \end{cases}$$

Ova funkcija je aproksimacija funkcije ELiSH opisane u poglavlju 4.2.9. Uvedena je za potrebe bržeg izvođenja sigmoide i nasljeđuje svojstva ELiSH funkcije. (Basirat i Roth, 2018)

4.2.11. Ograničena ispravljena linearna jedinica (ReLU-n)



Slika 4.11: Funkcija ReLU6 i njena derivacija

$$f(x) = \begin{cases} 0, & \text{ako } x < 0 \\ x, & \text{ako } x \in [0, n] \\ n, & \text{inače} \end{cases} \quad f'(x) = \begin{cases} 1, & \text{ako } x \in [0, n] \\ 0, & \text{inače} \end{cases} \quad (4.13)$$

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

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

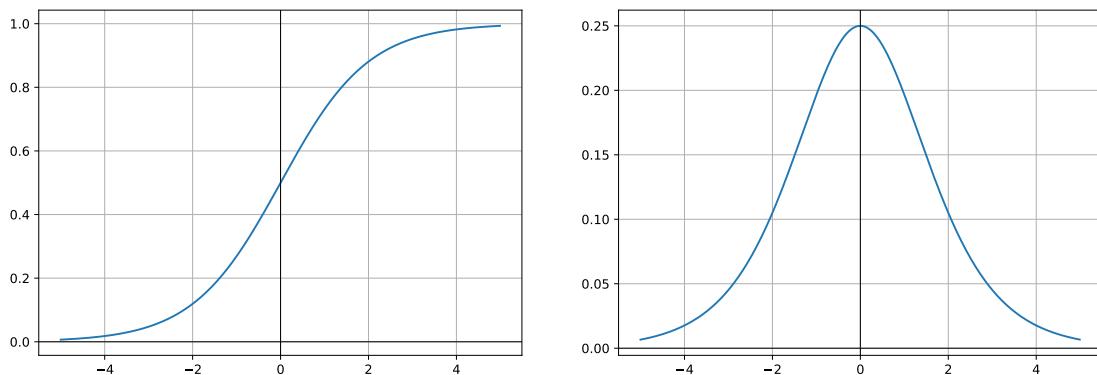
Ograničena ReLU funkcija uvedena je kako bi ograničeni Boltzmann-ov stroj (engl. *Restricted Boltzmann machine*) brže naučio rijetke značajke, koje kasnije ugađa konvolucijski klasifikator. Za razliku od ReLU koji se može interpretirati kao kompozicija

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 ReLU n 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.

4.2.12. Sigmoida (σ)

(engl. *Sigmoid*)



Slika 4.12: Sigmoida i njena derivacija

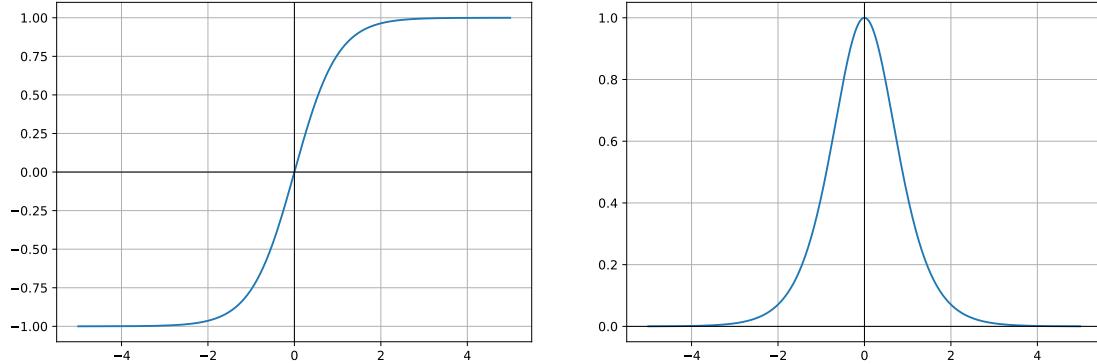
$$f(x) = \frac{1}{1 + e^{-x}} \quad f'(x) = \sigma(x)(1 - \sigma(x)) \quad (4.14)$$

TODO: koji problem rješava

TODO: svojstva

TODO: problemi

4.2.13. Tangens hiperbolni (tanh)



Slika 4.13: Funkcija tanh i njena derivacija

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad f'(x) = 1 - \tanh^2(x) \quad (4.15)$$

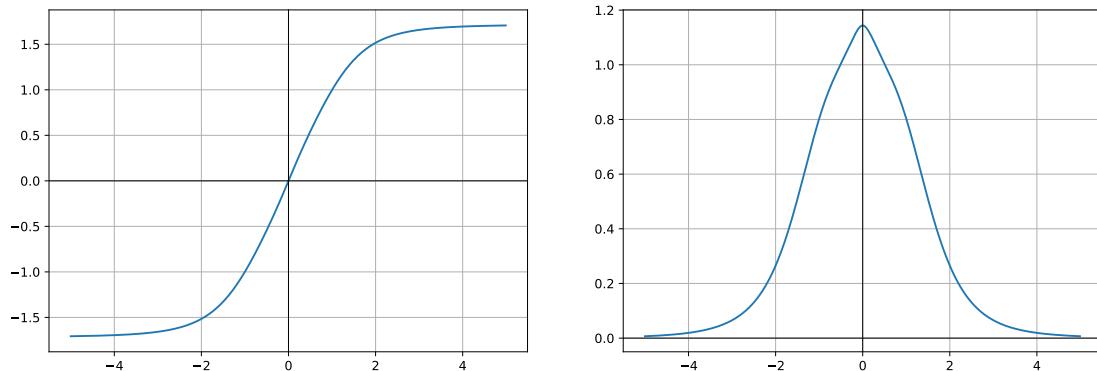
TODO: koji problem rješava

TODO: svojstva

TODO: problemi

4.2.14. Racionalna aproksimacija tanh

(engl. Rational tanh)



Slika 4.14: Racionalni tanh i njegova derivacija

$$\begin{aligned} f(x) &= 1.7159 \cdot \tanh^*(\frac{2}{3}x), \quad \tanh^*(x) = \operatorname{sgn}(x) \left(1 - \frac{1}{1 + |x| + x^2 + 1.41645 \cdot x^4} \right) \\ f'(x) &= 1.7159 \cdot \frac{2}{3} \cdot \tanh^{*\prime}(\frac{2}{3}x), \quad \tanh^{*\prime}(x) = \frac{1 + \operatorname{sgn}(x) \cdot (2x + 4 \cdot 1.41645 \cdot x^3)}{(1 + |x| + x^2 + 1.41645 \cdot x^4)^2} \end{aligned} \quad (4.16)$$

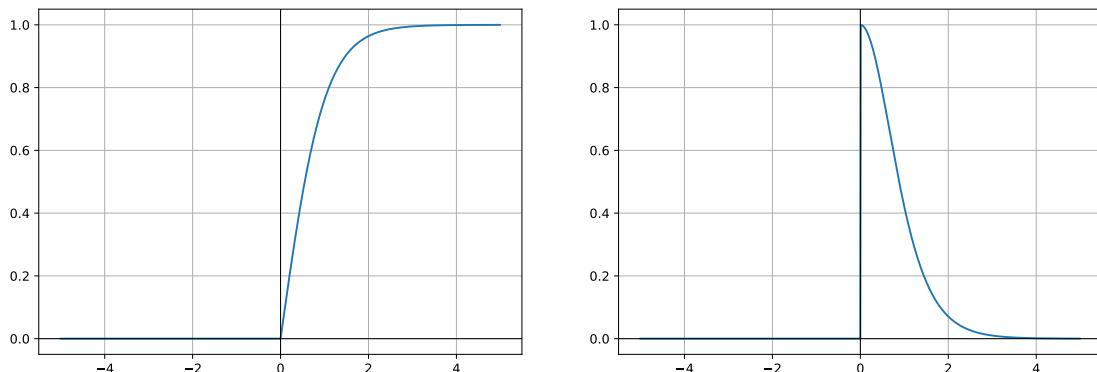
TODO: koji problem rješava

TODO: svojstva

TODO: problemi

4.2.15. Ispravljeni tanh

(engl. *Rectified tanh*)



Slika 4.15: Ispravljeni tanh i njegova derivacija

???

(4.17)

TODO: koji problem rješava

TODO: svojstva

TODO: problemi

4.2.16. Softmax

[IMAGE:]

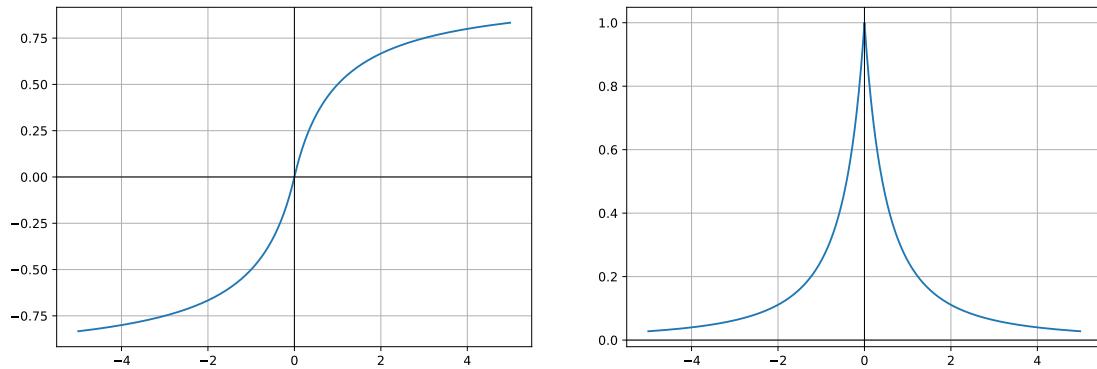
$$f(\vec{x}) = \frac{e^{\vec{x}}}{\sum_i e^{\vec{x}_i}} \quad f'(x) = \frac{e^x}{1 + e^x} \quad (4.18)$$

TODO: koji problem rješava

TODO: svojstva

TODO: problemi

4.2.17. Softsign



Slika 4.16: Softsign i njegova derivacija

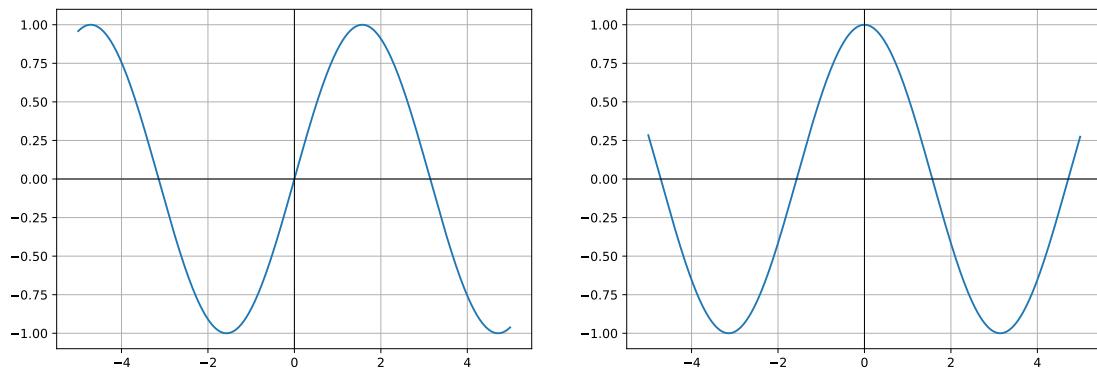
$$f(x) = \frac{x}{1 + |x|} \quad f'(x) = \frac{1 + |x| - x \cdot \text{sign}(x)}{(1 + |x|)^2} \quad (4.19)$$

TODO: koji problem rješava

TODO: svojstva

TODO: problemi

4.2.18. Sinus (sin)



Slika 4.17: Sinusoida i njegova derivacija

$$f(x) = \sin(x) \quad f'(x) = \cos(x) \quad (4.20)$$

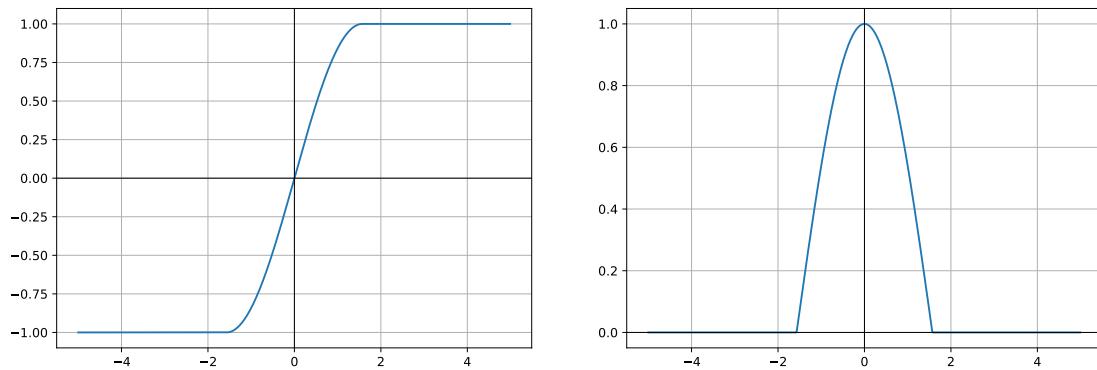
Sinus je periodička funkcija te nema teorijsku podlogu za uporabu u neuronskim mrežama, za razliku od monotonih funkcija. TODO: članak dokaz univerzalne aproksimacije monotonih fja

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 zagađuje valovitost funkcije gubitka. Dodatno, autori pokazuju da neuronska mreža zapravo ne ovisi snažno o periodičnosti funkcije tako da su rezultate usporedili s ograničenom sinusoidom (4.21). 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 4.12) i tangens hiperbolni (slika 4.13), sinusoida najveću vrijednost gradijenta daje upravo kada je aktivacija jednaka 0 (u ograničenom intervalu).

4.2.19. Ograničeni sinus (TrSin)

(engl. *Truncated sine*)

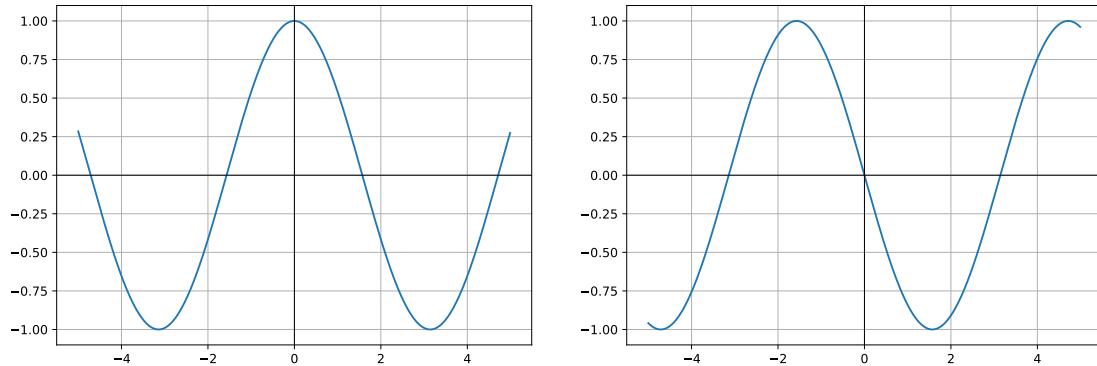


Slika 4.18: Ograničeni sinus i njegova derivacija

$$f(x) = \begin{cases} 0, & x < \frac{\pi}{2} \\ \sin(x), & |x| \leq \frac{\pi}{2} \\ 1, & \text{inače} \end{cases} \quad f'(x) = \begin{cases} \cos(x), & |x| \leq \frac{\pi}{2} \\ 0, & \text{inače} \end{cases} \quad (4.21)$$

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.

4.2.20. Kosinus (\cos)



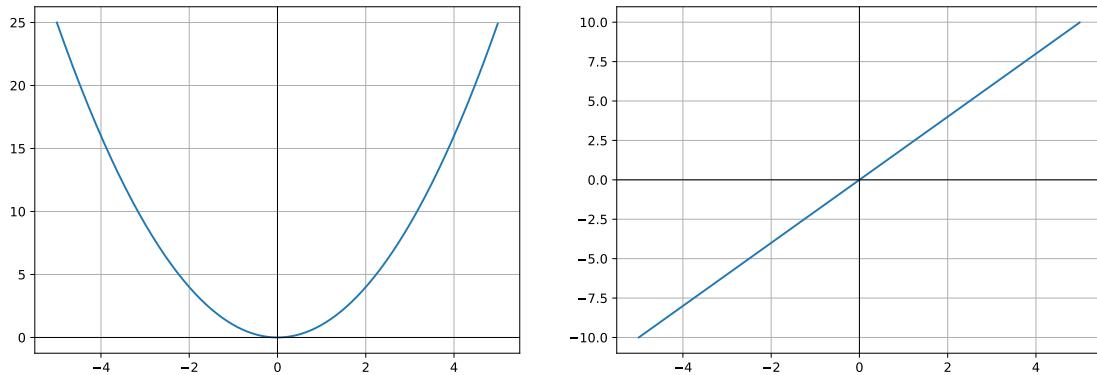
Slika 4.19: Kosinus i njegova derivacija

$$f(x) = \cos(x) \quad f'(x) = -\sin(x) \quad (4.22)$$

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).

$$\cos(x) = \sin(x + \frac{\pi}{2}) \approx \sin(x + 1.571) \quad (4.23)$$

4.2.21. Parabola x^2



Slika 4.20: Parabola i njena derivacija

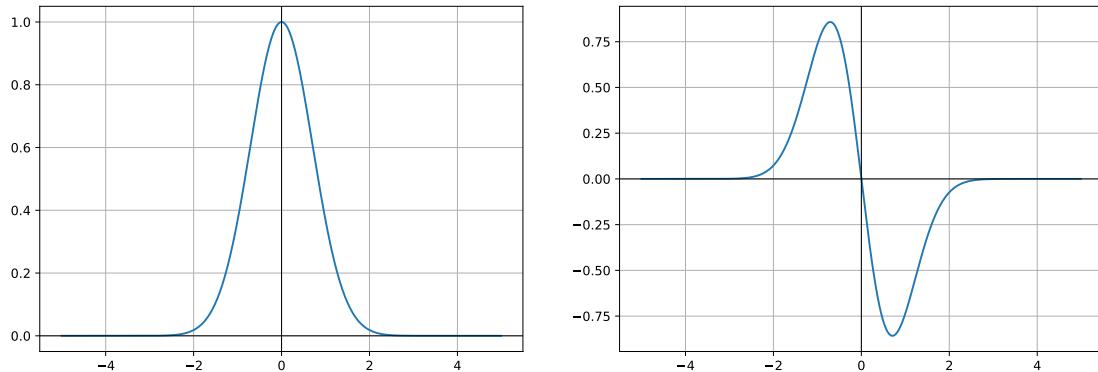
$$f(x) = x^2 \quad f'(x) = 2x \quad (4.24)$$

TODO: koji problem rješava

TODO: svojstva

TODO: problemi

4.2.22. Gaussova krivulja bez normalizacije



Slika 4.21: Gaussova krivulja bez normalizacije i njena derivacija

$$f(x) = e^{-x^2} \quad f'(x) = -2x \cdot f(x) \quad (4.25)$$

TODO: koji problem rješava

TODO: svojstva

TODO: problemi

5. Optimizacija izlazne funkcije genetskim programiranjem

5.1. Genetsko programiranje

TODO: *Opis i svojstva GP*

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

5.1.1. Simbolička regresija

TODO: *Opis SR*

5.1.2. Neuroevolucija genetskim programiranjem

TODO: *Kako predstaviti čitavu mrežu grafom (evolucija arhitekture)*

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.*

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

5.2. Pretraživanje izlaznih funkcija

TODO: *golden activation članak, kako radi detaljnije (Basirat i Roth, 2018)*

TODO: *swish članak detaljnije (?)*

5.2.1. Algoritam

TODO: *Problem konvergencije i stohastičnosti GP-a*

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

5.2.2. Skup čvorova (prostor pretraživanja)

Skup čvorova koji se koriste pri pretraživanju definiraju prostor pretraživanja. Skup čvorova je implementiran tako da razlikuje čvorove različitih stupnjeva i omogućuje operatorima selektivni, nasumično selektivni ili potpuno nasumičan pristup. Početna vrijednost konstante pri dohvatu je postojana i definira se unaprijed, a vrijednost pojedine konstante u stablima mijenja se operatorima.

Naziv	Funkcija	Stupanj	Naziv	Funkcija	Stupanj
x	x	0	elu	$ELU(x)$	1
const	$c \in \mathbb{R}$	0	gauss	e^{x^2}	1
+	$x + y$	2	lrelu	$LReLU(x)$	1
-	$x - y$	2	relu	$ReLU(x)$	1
*	$x \cdot y$	2	selu	$SELU(x)$	1
/	$\frac{x}{y+10^{-12}}$	2	sigmoid	$\frac{1}{1+e^{-x}}$	1
min	$\min(x, y)$	2	softmax	$Softmax(x)$	1
max	$\max(x, y)$	2	sotplus	$Softplus(x)$	1
abs	$ x $	1	softsign	$Softsign(x)$	1
sin	$\sin(x)$	1	swish	$Swish(x)$	1
cos	$\cos(x)$	1	tanh	$tanh(x)$	1
tan	$\tan(x)$	1	ostali	ostali	1
exp	e^x	1			
log	$\log_e(x)$	1			
pow2	x^2	1			
pow3	x^3	1			
pow	x^y	2			

Tablica 5.1: Popis čvorova korištenih pri pretrazi. U lijevoj tablici navedeni su čvorovi matematičkih funkcija, a u desnoj čvorovi popularnih aktivacijskih funkcija.

5.2.3. Operatori križanja

U nastavku opisani operatori križanja primaju dva roditelja, obavljaju križanje i vraćaju jedno dijete. Neki operatori istovremeno stvaraju dva djeteta, no vraćaju samo jedno nasumično odabranu. Ne obavlja se biranje djece prema dobroti zbog skupocijenosti postupka evaluacije, no preporuča se zbog kvalitetnije pretrage.

Zamjeni podstabla

Operator zamjene podstabla odabire u svakom roditelju po jedan čvor i zamjeni ih. Pri zamjeni čvorovi zadržavaju svoju djecu i time su efektivno zamjenjena podstabla. Operator služi za mješanje genotipa s čuvanjem podstruktura koje se pokazuju dobrima.

Rubni slučajevi su zamjena korijena stabla i zamjena listova. Pri zamjeni korijena jedno dijete postaje podstablo drugog roditelja. Pri zamjeni listova operator je ekviva-

lentan operatoru zamjene čvorova odnosno konstanti ako su obje konstante.

Zamijeni čvorove

Operator zamjene čvorova nasumično odabire čvor u stablu s manje čvorova. Zatim nasumično pronalazi čvor istog stupnja u većem stablu i zamjenjuje ih bez zamjene djece. Ako operator ne pronađe čvor istog stupnja, vraća nasumičnog roditelja. Prvo se odabire čvor iz manjeg stabla jer je veća vjerojatnost da će veće stablo sadržavati čvor istog stupnja, što je važno za unarne čvorove.

Zamijeni konstante

Operator zamjene konstanti nasumično odabire po jednu konstantu u oba roditelja i zamijeni ih. Operator je posebno koristan u kasnijem dijelu optimizacije kada pretraživanje funkcija konvergira, a pretražuju se optimalne konstante. U slučaju da barem jedan roditelj nema niti jednu konstantu, operator vraća nasumičnog roditelja.

Usrednji konstante

Operator usrednjavanja konstanti nasumično odabire po jednu konstantu u oba roditelja i jednu zamijeni njihovom aritmetičkom sredinom. Vraća se dijete sa zamijenjenom konstantom. Operator je posebno koristan u kasnijem dijelu optimizacije kada pretraživanje funkcija konvergira, a pretražuju se optimalne konstante. U slučaju da barem jedan roditelj nema niti jednu konstantu, operator vraća nasumičnog roditelja.

Vrati nasumičnog roditelja

Operator vraća nasumičnog roditelja kao dijete. Služi za osnaživanje konvergencije algoritma jer u populaciju vraća već postojeću jedinku koja dodatno prolazi kroz operator mutacije.

5.2.4. Operatori mutacije

U nastavku opisani operatori mutacije primaju jedno dijete i nad njime obave mutaciju. Operatori koji unose nove čvorove biraju čvorove iz skupa dostupnih čvorova 5.2.2.

Ubaci korijen

Operator ubacivanja korijena stabla odabire nasumičan čvor stupnja većeg od 0 iz seta i postavlja ga korijenom stabla. Ako je odabrani čvor unaran, njegovo djete postaje stari

korijen. Ako je odabrani čvor većeg stupnja prvo djete postaje korijen roditelja, a ostala njegova djeca popune se nasumičnim listovima iz skupa. Operator služi povećavanju dubine stabla i ključan je za stvaranje kompozicija funkcija.

Ubaci list

Operator ubacivanja lista odabire nasumičan čvor u stablu i zamjeni ga nasumičnim listom iz skupa. Ne postoje ograničenja na mjesto postavljanja lista pa operator može zamijeniti čitavo stablo listom. Iako je vrlo radikalni operator snažno pomaže očuvanju raznolikosti populacije. Operator služi smanjivanju dubine stabla i ključan je za pronalaženje plitkih stabala, što se odražava na brzinu izvođenja pronađenih funkcija.

Postavi vrijednost konstante

Operator odabire nasumičnu konstantu u stablu i dodjeljuje joj nasumičnu vrijednost. Vrijednost se uzorkuje iz uniformne distribucije zadanog intervala. Ako stablo ne sadrži konstante, operator ne mijenja stablo. Operator je ključan za pretraživanje prostora konstanti u kasnijem dijelu optimizacije kada pretraživanje funkcija konvergira.

Postavi cjelobrojnu vrijednost konstante

Operator je ekvivalentan operatoru postavljanja vrijednosti konstante, ali služi za pretraživanje prostora cjelobrojnih konstanti funkcija.

Pribroji vrijednost konstanti

Operator odabire nasumičnu konstantu u stablu i pribraja joj nasumičnu vrijednost. Vrijednost se uzorkuje iz normalne distribucije i skalira zadanom konstantom. Ako stablo ne sadrži konstante, operator ne mijenja stablo. Operator dijeli namjenu s operatom postavljanja vrijednosti konstante.

Izmjeni čvor

Operator odabire nasumičan čvor u stablu i zamjeni ga nasumičnim čvorom iz seta istog stupnja. Stablo se neće promijeniti ako postoji samo jedan čvor tog stupnja u setu. Operator ne mijenja veličinu stabla te služi za postizanje blažih izmjena u genotipu. Treba obratiti pažnju da se izmijene genotipa mogu značajno odraziti na fenotip.

Izmjeni podstablo

Operator odabire nasumičan čvor i na njegovo mjesto postavlja generirano podstablo. Podstablo se generira zadanim inicijalizatorom. Operator unosi snažne izmjene genotipa i služi održavanju raznolikosti populacije i globalnoj pretrazi prostora. Rubni uvijet je kada se generira stablo dubine 1, što utjecaj operatora čini identičnim operatuoru ubacivanja lista.

Ukloni korijen

Operator iz korijena stabla odabire nasumično dijete i njega postavi kao novi korijen. Ako je originalan korijen list, operator ne radi izmjene. Operator je komplementaran operatoru ubacivanja korijena te služi smanjivanju dubine stabla i ključan je za pronađenje plitkih stabala.

Ukloni unarni čvor

Operator odabire nasumični unarni čvor i zamjenjuje ga njegovim djjetetom. Ako stablo ne sadrži unarne čvorove operator ne unosi izmjene. Operator je posebno koristan pri eliminaciji dubokih kompozicija funkcija koje se znaju pojaviti pri pretraživanju, a koje nije moguće ukloniti operatorom uklanjanja korijena.

Zamijeni redoslijed djece

Operator odabire nasumičan čvor i njemu zamjeni mjesta dvaju djeteta. Ako je čvor stupnja 2 operator će im zamijeniti mjesta. Ako je čvor stupnja većeg od 2 operator će nasumično odabrati dva djeteta i njih zamijeniti. Pri tom operator može odabrati isto djete dvaput čime se stablo ne mijenja. Za sve ostale čvorove operator ne unosi izmjene. Operator je koristan za promjenu utjecaja čvorova koji ovise o redoslijedu djece (npr. operator djeljenja, oduzimanja, potenciranja i td.).

Inicijaliziraj genotip

Operator inicijalizacije genotipa zamjenjuje čitavo stablo novim generiranim stablom. Stablo se generira zadanim inicijalizatorom. Operator snažno pridonosi održavanju raznolikosti populacije i globalnoj pretrazi prostora.

6. Implementacija

6.1. Razvojna okolina i alati

Projekt je implementiran u programskom jeziku Java verzije 8, alatom IntelliJ. Razvojna okolina Deeplearning4j korištena je za učitavanje podatakovnih skupova, pretprocesiranje, izgradnju i optimizaciju neuronskih mreža te podršku izvođenja na grafičkim karticama s podrškom CUDA biblioteke. Obrada podataka i iscrtavanje grafova implementirano je u Jupyter bilježnicama u programskom jeziku Python verzije 3 korištenjem raznih biblioteka (numpy, sk-learn, matplotlib i ostalim).

Kod je dostupan na GitHub repozitoriju:

<https://github.com/lirfu/EvolvingOutputFunctions>

6.2. Organizacija koda

U implementaciji je napisano nekoliko izvršnih programa različitih namjena. Napisani su i UNIT testovi za ispitivanje ispravnosti ključnih dijelova implementacije. U nastavku su spomenuti glavni dijelovi implementacije korisni korisniku za brže snalaženje.

6.2.1. Evolucijski algoritmi

Evolucijski algoritmi i pomoćni razredi implementirani su u paketu *genetics*. Dizajn biblioteke inspiriran je bibliotekom *Evolutionary Computation Framework* (Jakobović et al.) u programskom jeziku C++.

Najsloženiji razred je apstraktni razred *Algorithm* koji čuva parametre i operatore algoritma, implementaciju inicijalizacije i pokretanja algoritma te praćenje i dohvata rezultata. Implementacije algoritma definiraju funkciju *runIteration* u kojoj se obavljaju sve operacije pri jednoj iteraciji algoritma. Implementacije algoritma nalaze se u paketu *genetics.algorithms*. Algoritam podržava oblikovni obrazac *graditelj* za

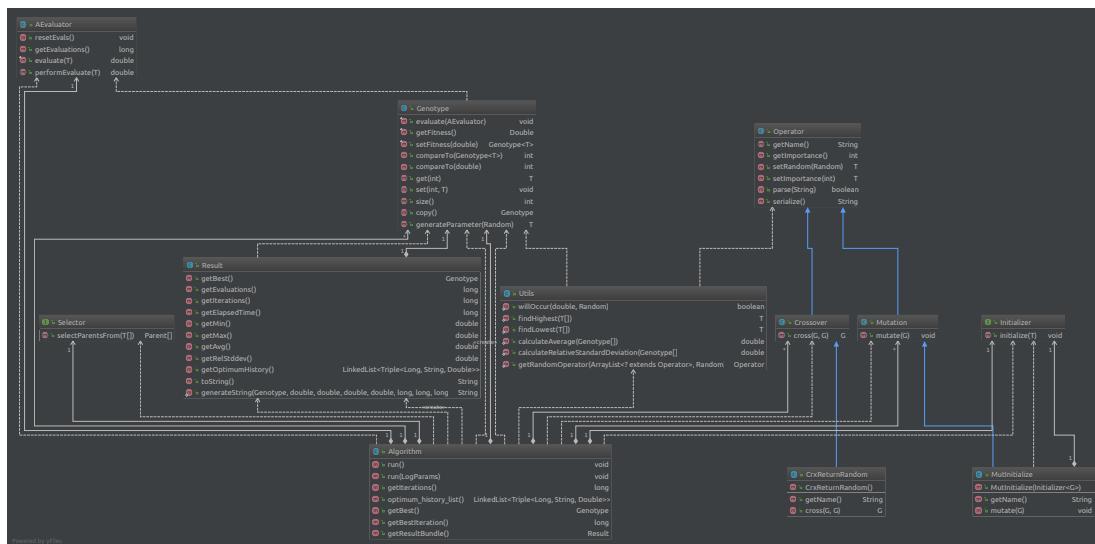
jednostavniju i pregledniju definiciju algoritma. Primjere implementacije moguće je pronaći u paketu *genetics.algorithms*.

Genotip je definiran apstraktnim razredom *Genotype* koji čuva svoju vrijednost dobrote i definira nekoliko apstraktnih metoda za rad s genotipom. Primjer implementacije genotipa je razred *symboregression.SymbolicTree* opisan kasnije. Genotip je moguće serijalizirati u tekst.

Operatori su definirani apstraktnim razredom *Operator* koji čuva vrijednost važnosti operatora te referencu na generator slučajnih brojeva. Operatori križanja *Crossover* i mutacije *Mutation* dodaju svoje apstraktne metode koje je potrebno implementirati i koje se pozivaju u algoritmu. Implementacije generičnih operatora nalaze se u korijenu paketa, dok se operatori specifični za pojedine vrste genotipa nalaze u zasebnim paketima tog genotipa (npr. *genetics.symboregression*). Parametre opeatora moguće je serijalizirati u tekst.

Metoda odabira jedinki iz populacije definirana je sučeljem *Selector* koja definira apstraktnu metodu *selectParentsFrom* koja na ulaz dobiva populaciju, a vraća polje kandidata za roditelje. Ovisno o implementaciji, algoritam odabira može vratiti samo 2 roditelja (*RouletteWheelSelector*) ili 2 roditelja i jedinku koju zamjenjujemo djetetom (*TournamentSelector*). Metode odabira implementirane su u paketu *selectors*.

Inicijalizator populacije definiran je sučeljem *Initializer*. Uvjet zaustavljanja algoritma definiran je razredom *stopconditions.StopCondition* čiji objekt se predaje algoritmu. Uvijete zaustavljanja moguće serijalizirati u tekst. Rezultat algoritma opisan je razredom *Result* kojeg je moguće serijalizirati u tekst.



Slika 6.1: Paket *genetics*

Simbolička regresija

Implementacija simboličke regresije nalazi se u paketu *genetics.symboregression*. Sadrži operatore križanja definirane paketom *crx* i operatore mutacije definirane paketom *mut*. Genotip je definiran razredom *SymbolicTree* koji sadrži referencu na korijenski čvor. Čvorovi su definirani apstraktnim razredom *TreeNode*, a skup dostupnih čvorova s pristupnim metodama definiran je razredom *TreeNodeSet*.

Izvršavanje operacija obilaskom stabla obavlja se objektom sučelja *IExecutable* koji se definira pri implementaciji čvora. Čvor sadrži i operacije za dohvati i zamjenu djece te izvršnog objekta, a koje su potrebne operatorima križanja i mutacije. Čvor može primiti i dodatan objekt koji može poslužiti čvorovima specijalne namjene.

6.2.2. Neuronske mreže

TODO: *glavne komponente*

[IMAGE: IntelliJ generirana UML paketa neurology]

6.2.3. Pomoćni mehanizmi za projekt

Zajednički razred za optimizaciju mreža

Automatska pohrana rezultata

TODO: *storagemanager*

Parametri

Hiperparametre programa moguće je zadati konfiguracijskom datotekom. Datoteka je formata ključ-vrijednost, odvojeni regularnim izrazom: [: \t]+.

TODO: *definiranje parametara kroz datoteku*

TODO: **dodavanje novih parametara*

TODO: *grid search mehanizam*

Paralelizacija

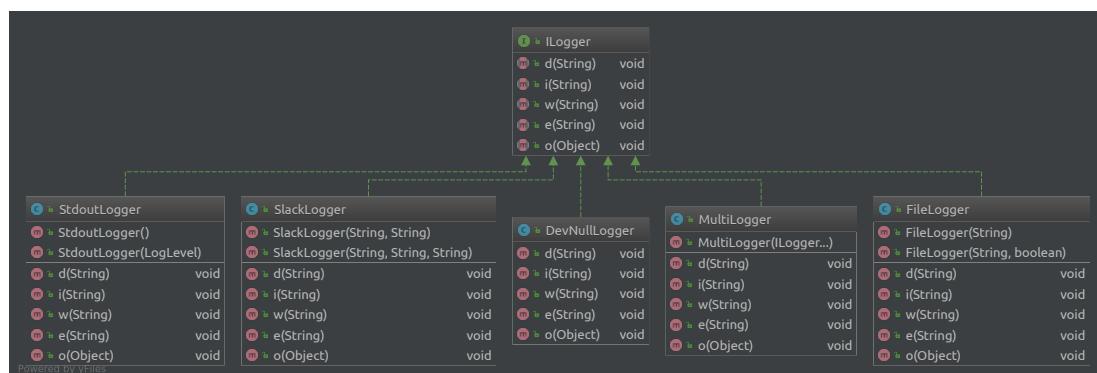
Paralelizaciju je moguće postići vrlo jednostavno paketom *utils.threading*. Prvo je potrebno instancirati razred *WorkArbiter* koji u sebi sadrži red poslova koje treba obaviti i listu radnika. Posao koji treba izvršiti definira se implementacijom sučelja *Work*. Poslovi se predaju objektu *WorkArbiter* koja ih stavlja u red. Radnici *Worker* paralelno i sinkronizirano dohvaćaju i izvršavaju poslove iz reda poslova razreda *WorkArbiter* koji ih je stvorio. S obzirom da su radnici implementirani kao zasebne dretve, navalom poslova radnici će paralelno uzimati poslove i izvršavati ih.

Čekanje pozivajuće dretve na izvršenje zadataka moguće je ostvariti pozivom metode *waitOn* koja prima uvijet čekanja *WaitCondition*. Moguće je definirati svoj uvijet (npr. čekanje dok se ne popuni spremnik) ili pričekati dok se svi poslovi ne izvrše, uvjetom koji vraća metoda *getAllFinishedCondition*.

TODO: *sinkronizacija u GA i pomoći razredi/metode*

Logiranje

TODO: *kaj sve imaš i kak se koristi*



Slika 6.2: Paket *utils.logs*

6.2.4. Izvršni programi projekta

Pohlepna pretraga uobičajenih funkcija

Izgradnja izlaznih funkcija simboličkom regresijom

7. Rezultati

7.1. DPAv4

7.1.1. Usporedba uobičajenih izlaznih funkcija

TODO: *Opis postupka pretrage*

TODO: *Tablica*

TODO: *Komentar*

7.1.2. Izgradnja izlaznih funkcija simboličkom regresijom

TODO: *Tablica*

TODO: *Komentar*

[IMAGE: *boxplot usporedba po taboo veličini*]

7.1.3. Izgradnja heterogenog rasporeda izlaznih funkcija

TODO: *Valjda će stići i ovo izvrstiti. Za najbolju prethodnu arhitekturu će pronaći najbolji raspored uobičajenih fja po slojevima mreže (npr. sin-relu).*

7.2. DPAv2

7.2.1. Usporedba uobičajenih izlaznih funkcija

U ovom poglavljtu prikazana je usporedba i rezultati uobičajenih izlaznih funkcija. Ispitivane su kombinacije arhitekture i izlazne funkcije napisane su mjere točnosti i F1 na skupu za testiranje. Za svaku kombinaciju arhitekture i izlazne funkcije provedeno

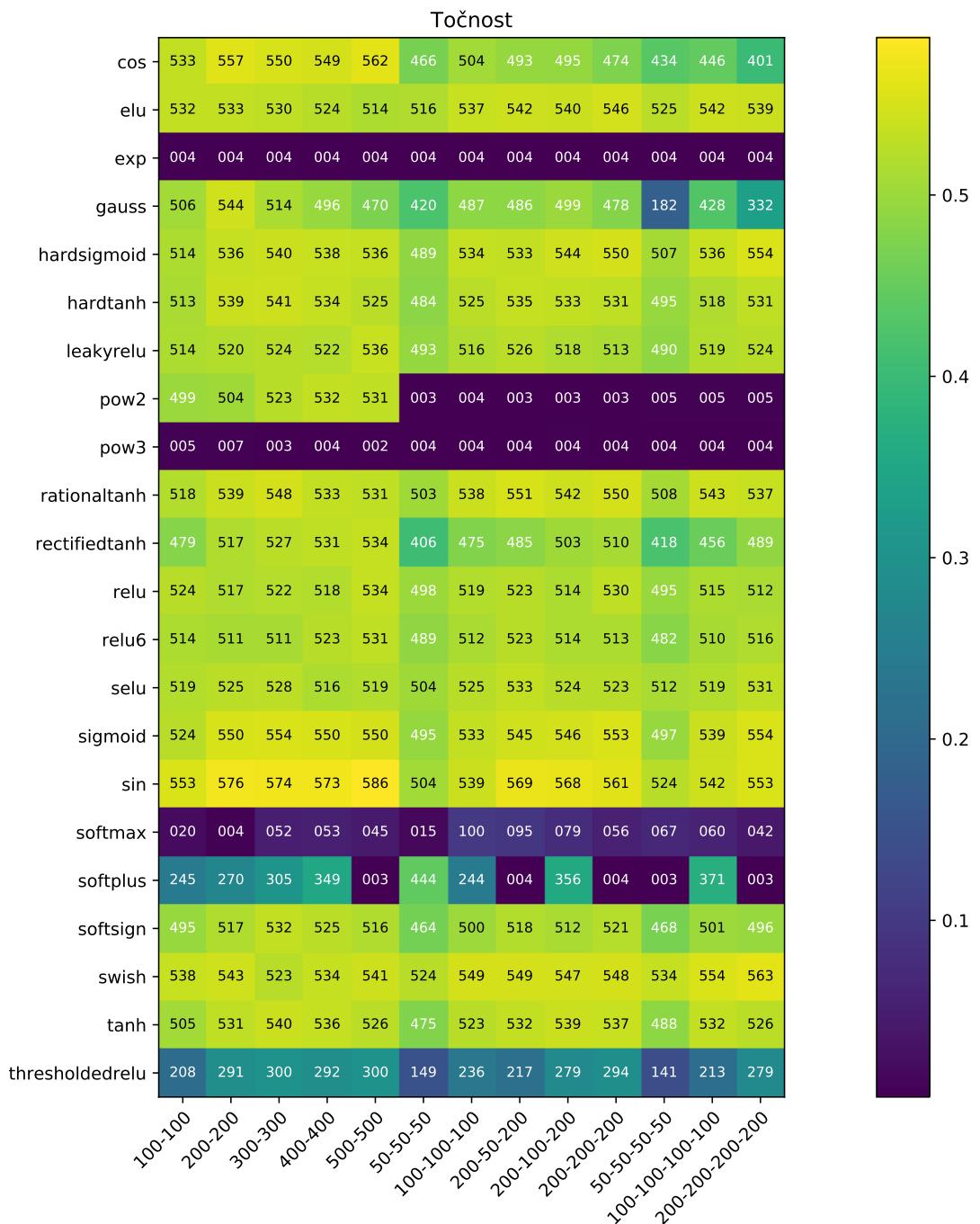
Hiperparametar	Vrijednosti
Seed	42
Veličina minigrupe	256
Normalizacija značajki	Da
Permutacija mini-grupa	Ne
Normalizacija mini-grupe	Da
Dropout	Ne
Stopa opadanja stope učenja	0.99
Broj iteracija do idućeg opadanja stope učenja	1
Maksimalan broj epoha	40
Broj uzastopnih iteracija za rano zaustavljanje	5
Minimalna relativna promjena gubitka za detekciju konvergencije	0.01
Koeficijent L2 regularizacije	$10^{-3}, 10^{-4}, 10^{-5}$
Stopa učenja	$10^{-3}, 5 \cdot 10^{-4}, 10^{-4}$

Tablica 7.1: Hiperparametri korišteni pri učenju mreža. Zarezima su odvojene vrijednosti hiperparametara koje su pretraživane po rešetci.

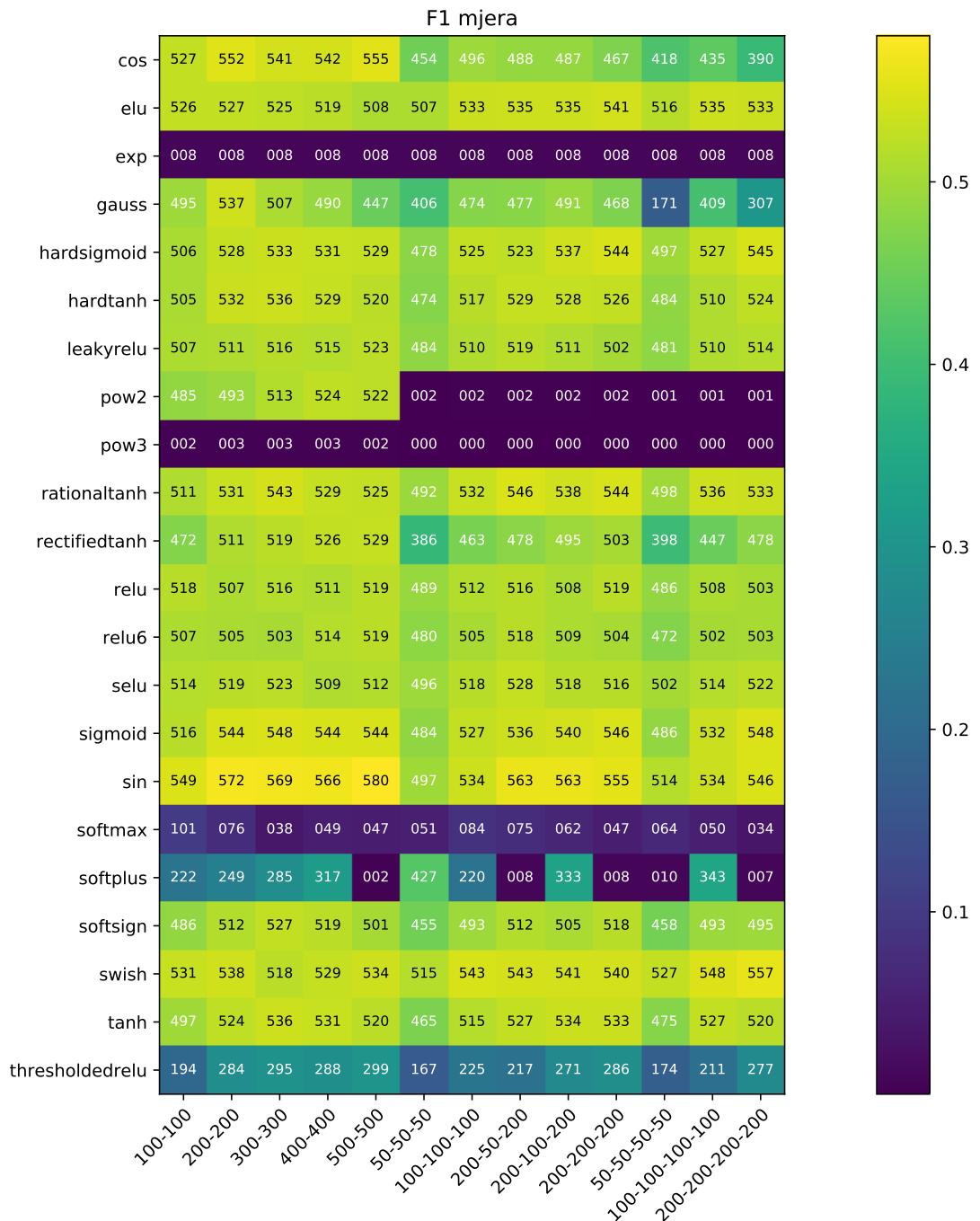
je pretraživanje hiperparametara po rešetci i napisani su najbolji postignuti rezultati. Zajednički hiperparametri i pretraživani navedeni su u tablici 7.1 (pretraživani su na dnu). Kombinacije hiperparametara su indeksirane prema tablici 7.2 za lakše snalaženje na slici najboljih kombinacija hiperparametara 7.3.

Indeks	Stopa učenja	Koeficijent L2 regularizacije
1	10^{-3}	10^{-3}
2	10^{-3}	$5 \cdot 10^{-4}$
3	10^{-3}	10^{-4}
4	10^{-4}	10^{-3}
5	10^{-4}	$5 \cdot 10^{-4}$
6	10^{-4}	10^{-4}
7	10^{-5}	10^{-3}
8	10^{-5}	$5 \cdot 10^{-4}$
9	10^{-5}	10^{-4}

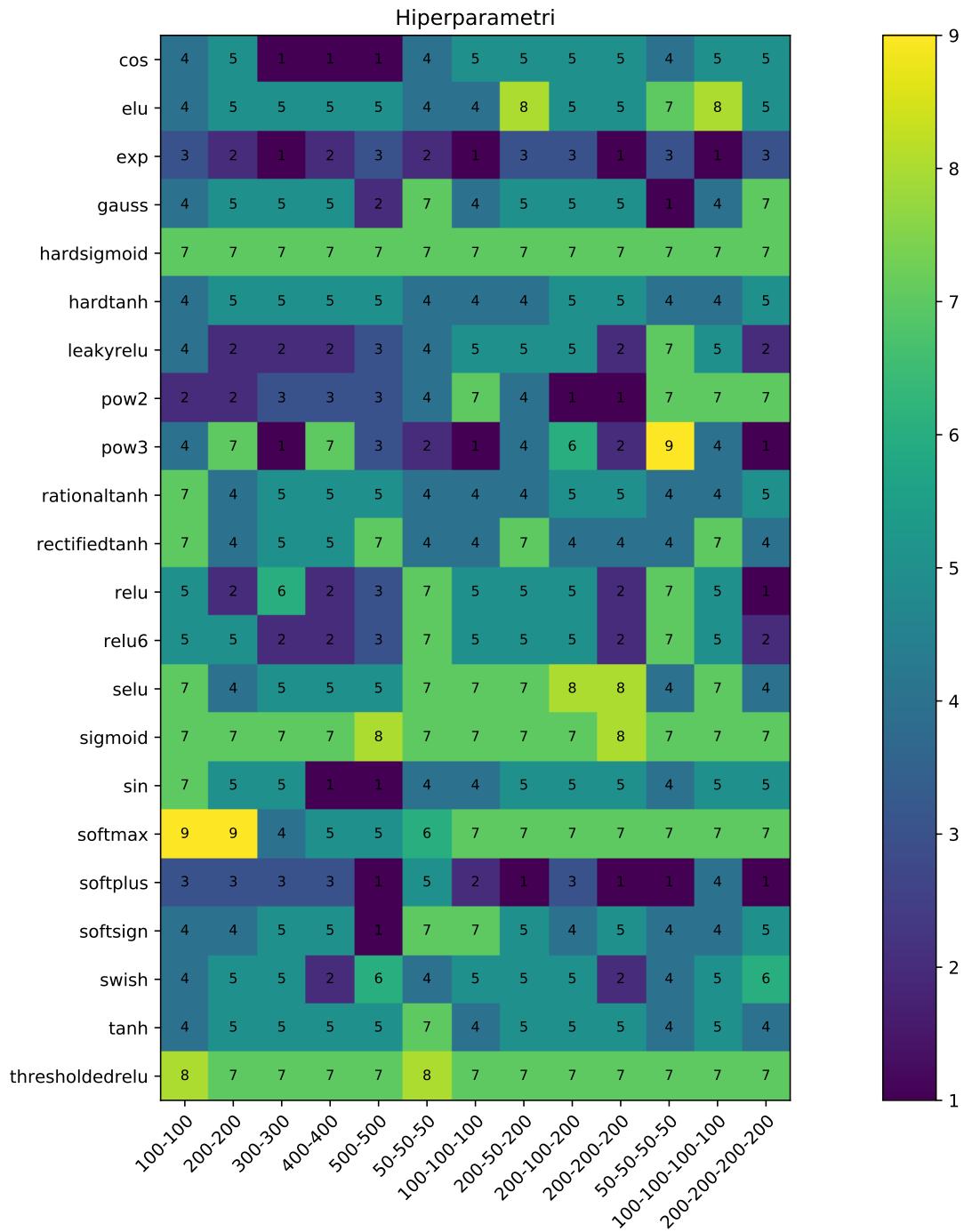
Tablica 7.2: Indeksi kombinacija hiperparametara korištenih pri pretraživanju po rešetci.



Slika 7.1: Postignuta točnost izlaznih funkcija na različitim arhitekturama. Vrijednosti u čelijama su tri najznačajnije znamenke nakon točke. Na vrijednosti mjeri 0.5 tekst mijenja boju iz bijele u crnu.



Slika 7.2: Postignuta F1 mjera izlaznih funkcija na različitim arhitekturama. Vrijednosti u čelijama su tri najznačajnije znamenke nakon točke. Na vrijednosti mjere 0.5 tekst mijenja boju iz bijele u crnu.



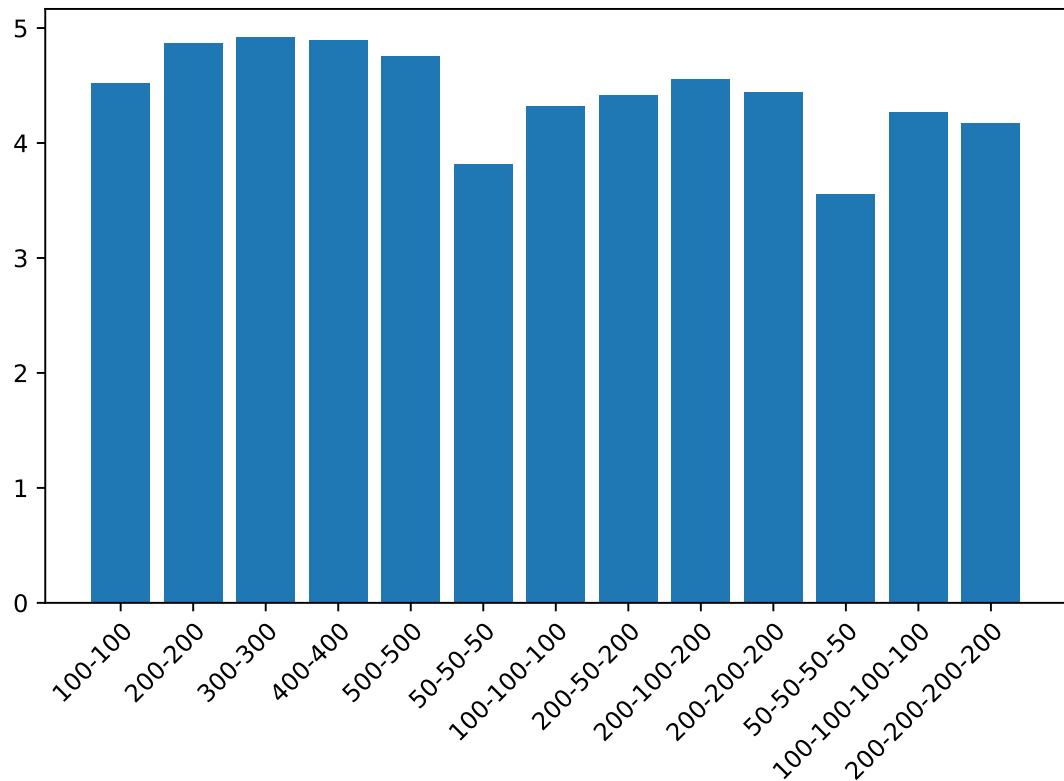
Slika 7.3: Optimalne kombinacije hiperparametara za izlazne funkcije na različitim arhitekturama. Vrijednosti u čelijama označavaju kombinacije hiperparametara navedene u tablici 7.2

[IMAGE: mesta aktivacije top 4 funkcije po arh]

TODO: Komentar

[IMAGE: mesta aktivacije bottom 4 funkcije po arh (tu je i pow2)]

TODO: Komentar



Slika 7.4: Suma kvadrata F1 mjera po svim arhitekturama. Kvadriranjem veći utjecaj na sumu imaju veće vrijednosti (želimo zanemariti loše vrijednosti). Najveću sumu ostvaruje arhitektura 300 – 300 te se upravo ona uzima kza daljnje eksperimente. Koristi se najčešća kombinacija hiperparametara za tu arhitekturu je pod indeksom 5 tablice 7.2.

7.2.2. Izgradnja izlaznih funkcija simboličkom regresijom

TODO: *Tablica*

TODO: *Komentar*

[IMAGE: *boxplot usporedba po taboo veličini*]

7.2.3. Izgradnja heterogenog rasporeda izlaznih funkcija

TODO: *Valjda će stići i ovo izvrtiti. Za najbolju prethodnu arhitekturu će pronaći najbolji raspored uobičajenih fja po slojevima mreže (npr. sin-relu).*

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.

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

Włodzisław 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.

Domagoj Jakobović, Hrvoje Ban, Vinko Bedek, Igor Bespaljko, Iva Brajer, Luka Dončić, Luka Franov, Zvonimir Fras, Danko Komlen, Ivan Kokan, Luka Krizan, Maja Legac, Tomislav Novak, Lovro Paić-Antunović, Stjepan Picek, Dražen Popović, Ángel Ferreira-Santiago, Domagoj Stanković, Ivana Stokić, i Mirjam Škarica. Evolutionary computation framework (ECF). URL <http://ecf.zemris.fer.hr/>.

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.

Eduardo D. Sontag. Vc dimension of neural networks. 1998.

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.

Xin Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, Sep. 1999. ISSN 0018-9219. doi: 10.1109/5.784219.

Marko Čupić, Bojana Dalbelo Bašić, i Marin Golub. *Neizrazito, evolucijsko i neurorачunarstvo*. 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. Usportediti učinkovitost ostvarenih postupaka s postojećim rješenjima iz literature. Radu prilожiti izvorene 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 necessary discussion and literature used.

Keywords: Keywords.