

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1966

**Optimizirane aktivacijske 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
2.3.3. Dosadašnji rezultati	4
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	18
3.1.4. Odabir hiperparametara	21
3.1.5. Svojstva	25
3.1.6. Problemi	26
4. Aktivacijske funkcije	28
4.1. Popularne aktivacijske funkcije	29
4.1.1. Funkcija identiteta	29
4.1.2. Zglobnica ili ispravljena linearna jedinica (ReLU)	30
4.1.3. Propusna ispravljena linearna jedinica (LReLU)	31
4.1.4. Ispravljena linearna jedinica s pragom (ThReLU)	31
4.1.5. Softplus	32
4.1.6. Eksponencijalno-linearna jedinica (ELU)	33
4.1.7. Skalirana eksponencijalno-linearna jedinica (SELU)	34

4.1.8. Swish	35
4.1.9. ELiSH	35
4.1.10. Tvrdi ELiSH	36
4.1.11. Ograničena ispravljena linearna jedinica (ReLU-n)	37
4.1.12. Sigmoida (σ)	38
4.1.13. Tvrda sigmoida	39
4.1.14. Tangens hiperbolni (tanh)	39
4.1.15. Tvrdi tangens hiperbolni	40
4.1.16. Racionalna aproksimacija tanh	40
4.1.17. Ispravljeni tanh	41
4.1.18. Softsign	42
4.1.19. Sinus (sin)	43
4.1.20. Ograničeni sinus (TrSin)	43
4.1.21. Kosinus (cos)	44
4.1.22. Softmax	45
5. Optimizacija aktivacijske funkcije genetskim programiranjem	46
5.1. Genetsko programiranje	46
5.1.1. Građa	46
5.1.2. Neuroevolucija genetskim programiranjem	48
5.2. Optimizacija aktivacijskih funkcija	49
5.3. Genetsko programiranje s tabu listom	50
5.3.1. Skup čvorova (prostor pretraživanja)	51
5.3.2. Operatori križanja	52
5.3.3. Operatori mutacije	53
6. Implementacija	56
6.1. Razvojna okolina i alati	56
6.2. Organizacija koda	56
6.2.1. Evolucijski algoritmi	56
6.2.2. Neuronska mreža	58
6.2.3. Pomoćni mehanizmi projekta	58
6.2.4. Izvršni programi projekta	60
7. Rezultati	62
7.1. DPAv4	62
7.1.1. Usporedba uobičajenih aktivacijskih funkcija	62

7.1.2. Izgradnja aktivacijskih funkcija simboličkom regresijom	67
7.1.3. Izgradnja heterogenog rasporeda aktivacijskih funkcija	67
7.2. DPAv2	67
7.2.1. Usporedba uobičajenih aktivacijskih funkcija	67
7.2.2. Izgradnja aktivacijskih funkcija simboličkom regresijom	71
7.2.3. Izgradnja heterogenog rasporeda aktivacijskih funkcija	71
8. Stvari koje sam probao, ali nisu ispale korisne	72
9. Buduća istraživanja	73
10. Zaključak	74
Literatura	75
A. Hiperparametri pretrage po rešetci	82

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**

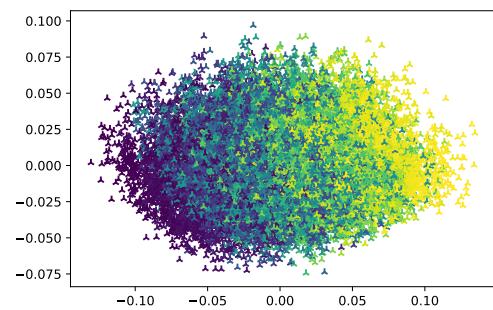
TODO: *Ne zaboravi referencu na stranicu!*

2.3.1. DPAv2

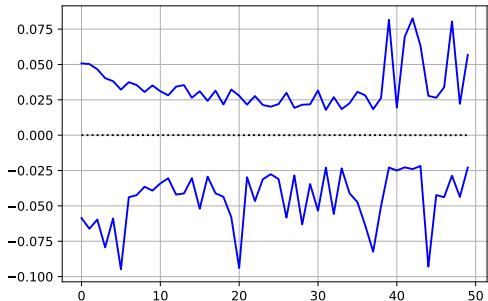
TODO: *Kad je napravljen i ko ga je radil*



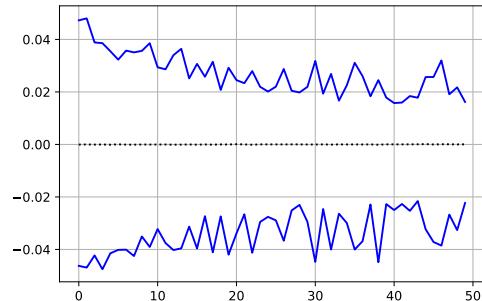
(a) PCA redukcija skupa za učenje



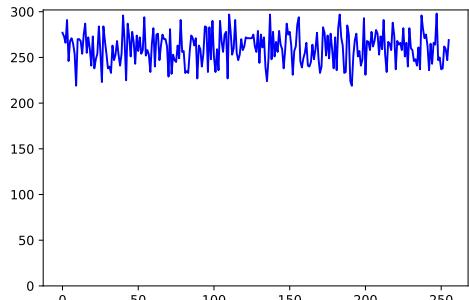
(b) PCA redukcija skupa za testiranje



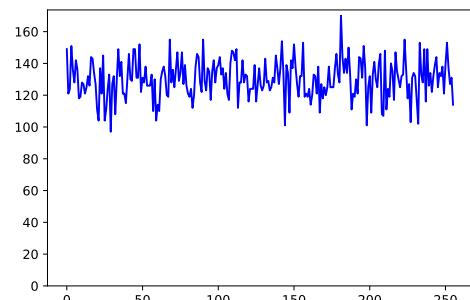
(c) Distribucija značajki skupa za učenje



(d) Distribucija značajki skupa za testiranje



(e) Distribucija oznaka skupa za učenje



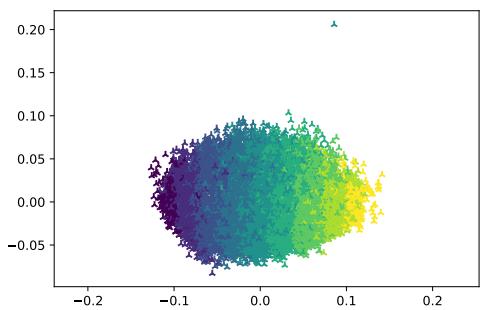
(f) Distribucija oznaka skupa za testiranje

Slika 2.1: Statistike podskupa za učenje i testiranje skupa DPAv2

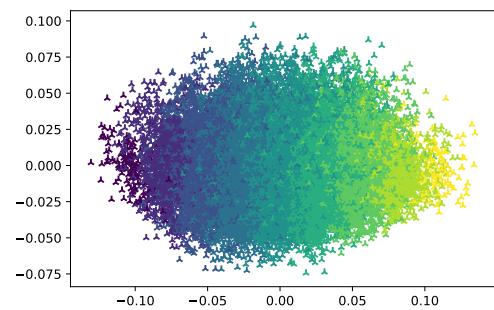
TODO: *Mjere dobrote klasifikacije*

2.3.2. DPAv4

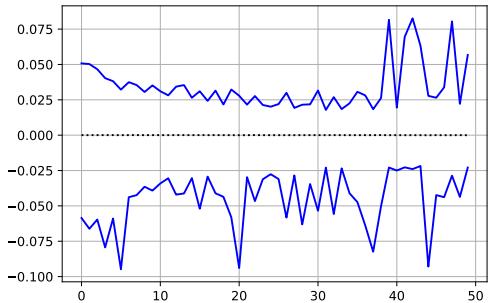
TODO: *Kada je napravljen i ko ga je radilo*



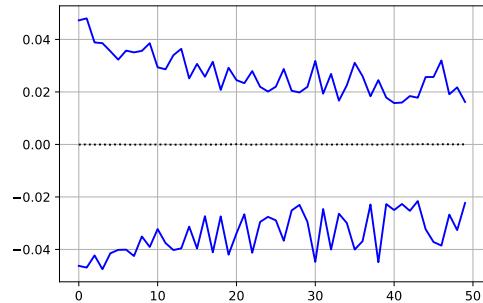
(a) PCA redukcija skupa za učenje



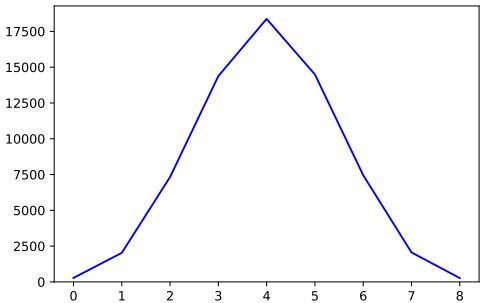
(b) PCA redukcija skupa za testiranje



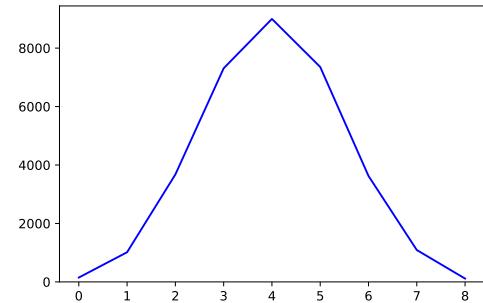
(c) Distribucija značajki skupa za učenje



(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

2.3.3. Dosadašnji rezultati

3. Klasifikator temeljen na umjetnim neuronskim mrežama

3.1. Umjetne neuronske mreže

Umjetne neuronske mreže (nadalje „neuronske mreže“) koriste se 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 te često dostižu ljudske performanse na zadanom problemu. Danas su vrlo raširene u raznim područjima od kojih su samo neka: računalni vid [27, 37], prirodna obrada jezika [31, 23] i podržano učenje [32, 11].

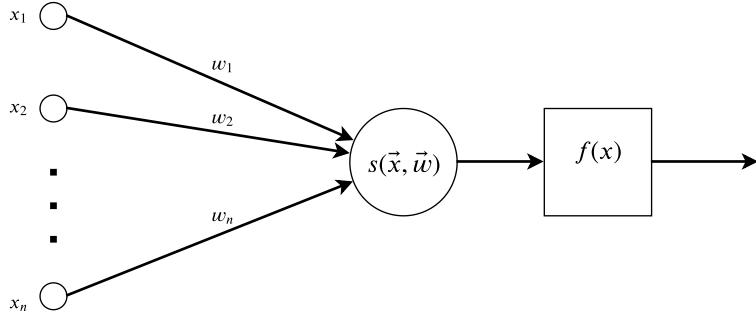
3.1.1. Građa

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 nazivaju se parametri neurona.

Način na koji se iz ulaza gradi unutarnje stanje neurona opisan je ulaznom funkcijom. Pretvorba unutarnjeg stanja neurona u izlazni signal opisana je aktivacijskom funkcijom. Ulazna i aktivacijska funkcija opisuju ponašanje neurona (3.1). Iako je ova nomenklatura dobra u većini modernih radova, ponekad razlika između ulazne i izlazne funkcije nije očigledna (kao kod radikalne bazne funkcije).

$$n(x) = (f \circ s)(x) = f(s(x)) \quad (3.1)$$

U literaturi postoje nekonzistencije oko pojma aktivacijske funkcije. U pregled-



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

nim znanstvenim radovima [6, 7, 8] pojam aktivacijske funkcije predstavlja ulaznu funkciju, što je neispravno prema brojnim današnjim radovima. Neispravno je i sa stajališta računalne neuroznanosti prema kojoj aktivacijska funkcija označava funkciju učestalosti okidanja neurona u ovisnosti o ukupnom statickom ulazu [5, str. 234]. U starijoj literaturi pojam aktivacijske i prijenosne funkcije često se koristi ekvivalentno [47, 17]. U prijedlogu standardizacije Eberhart [9] aktivacijska funkcija je definirana: „*Algoritam za računanje vrijednosti aktivacije neurona kao funkcije njenog ukupnog ulaza. Ukupan ulaz je tipično težinska suma ulaza u neuron*“. S obzirom na nekonzistencije u literaturi i iz potrebe razlikovanja funkcija koje prikupljaju i odašilju signale iz neurona u ovom se radu koristi gore spomenuta nomenklatura (ulazna i aktivacijska funkcija).

Najpopularnije ulazne funkcije jesu afina funkcija i unakrsna korelacija. Afina funkcija (3.2) 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.

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

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 [27], no može se koristiti u problemima koji zahtijevaju obradu sljedova podataka. Jedan takav primjer je klasifikacija teksta pri kojoj slijed riječi gradi informaciju koja može služiti za detekciju sentimenta i ostale primjene [23].

U radu Turian et al. [45] klasičnoj afinoj funkciji pribraja se L2 norma dodatne linearne funkcije čije elemente nazivaju *kvadratnim filterima*. Empirijski pokazuju da

dodavanje kvadratnih filtara pospješuje performanse modela.

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(poglavlje 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 nавести samo redoslijed slojeva, no ponekad je potrebno definirati i način povezivanja slojeva npr. pri uporabi preskočnih veza [39, 19, 20]. Prvi sloj služi za postavljanje ulaza mreže i naziva se ulaznim slojem mreže. Posljednji sloj mreže služi za ekstrakciju izlaza te mjerjenje kakvoće mreže i naziva se 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 kaže se 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.

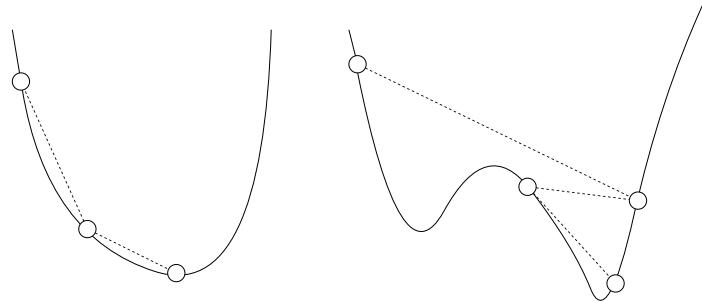
3.1.2. Optimizacija umjetne neuronske mreže

Optimizacijom parametara neuronska mreža prilagođava se danom zadatku, odnosno kaže se da mreža 'uči'. Optimizacija parametara najčešće se izvodi gradijentnim spustom, uz pretpostavku derivabilnosti svih komponenata neuronske mreže. Kada ta pretpostavka ne vrijedi koriste se algoritmi kombinatorne optimizacije poput algoritma roja čestica koji se spominje u Čupić et al. [48]. U ovom radu neuronske mreže optimiraju se gradijentnim spustom.

Gradijentni spust

Gradijentni spust je algoritam pronalaska minimuma funkcije vođen gradijentom te funkcije. Za zadanu početnu točku iterativno se pomiče u smjeru suprotnom od gradijenta funkcije u toj točki dok ne zadovolji neki od uvjeta zaustavljanja. Na strmim funkcijama gradijent je često prevelik i može izazvati oscilaciju ili divergenciju (slika 3.3). Stoga se gradijent pri pomaku skalira koeficijentom pomaka μ . Dobro odabran koeficijent pomaka može osigurati bržu konvergenciju, a kod višemodalnih funkcija i

pronalazak boljeg optimuma (slika 3.2).



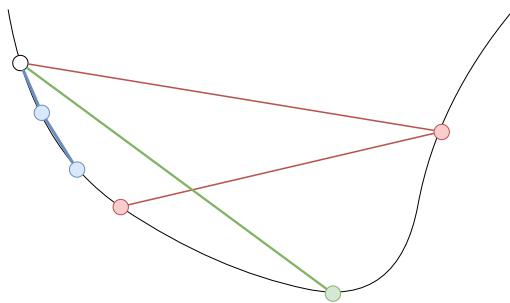
Slika 3.2: Za razliku od unimodalnih, višemodalne funkcije mogu navesti gradijentni spust na nemonotoni spust.

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 3.4).

```

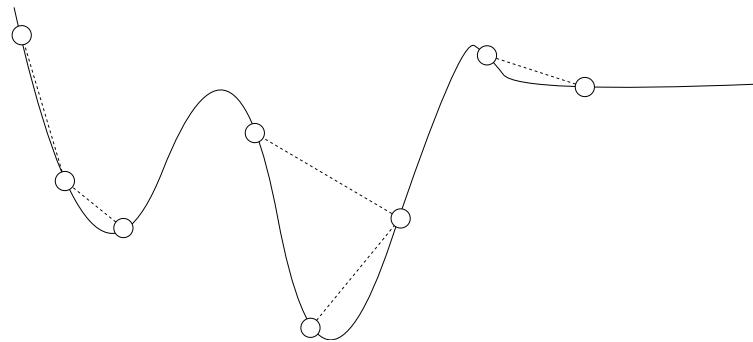
Input: funkcija  $f(\vec{x})$ 
Input: početna točka  $\vec{x}_0$ 
Input: koeficijent pomaka  $\eta$ 
Input: broj iteracija  $n$ 
for  $n$  iteracija do
     $\vec{g}_i \leftarrow \vec{\nabla}_{\vec{x}} f(\vec{x}_i)$ 
     $\vec{x}_{i+1} \leftarrow \vec{x}_i - \eta \cdot \vec{g}_i$ 
end
Result: pronađena optimalna točka  $\vec{x}_i$ 
```

Algorithm 1: Gradijentni spust



Slika 3.3: Prikazan je utjecaj premalog (plavo), prevelikog (crveno) i optimalnog (zeleno) koeficijenta spusta kroz dvije iteracije. Odabir koeficijenta utječe na brzinu spusta i kvalitetu konačnog rješenja.

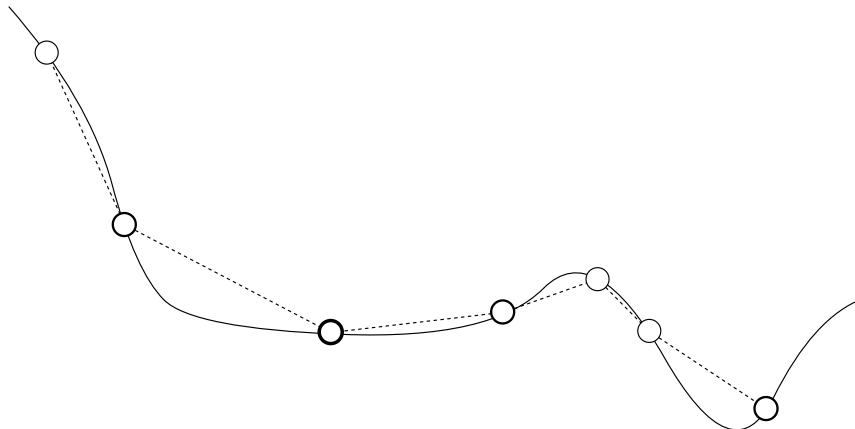
Broj iteracija definira broj pomaka od početne točke, što definira i trajanje algoritma. Generalno želi se skratiti vrijeme pretraživanja te povećati koeficijent spusta kako bi se koristilo manje pomaka. No u praksi se najčešće nailazi na višemodalne funkcije sa strmim regijama koje izazivaju oscilacije 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.).



Slika 3.4: Prikazan je utjecaj odabira početne točke na kvalitetu pronađenog optimuma. Loše odabrana točka (lijevo i desno) može navesti algoritam izvan poželjne regije (sredina) i zaustaviti algoritam u lokalnom optimumu.

Problem se javlja ako algoritam odluta u visoravan na kojoj su gradijenti vrlo mali, a sama regija je s obzirom na pomake velika (slika 3.4, desno). Kad gradijent postane neupotrebljivo malen kaže se da je *nestao*. 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 visoravnji pomičući ga u smjeru koji je akumuliran. Kako moment ne bi izvukao algoritam iz optimuma, dodaje mu se koeficijent *zaboravljanja* kojim se stari vektor momenta djelomično zaboravlja u korist novog vektora pomaka (jednadžba 3.3). Moment može pomoći i pri zaobilaženju lokalnih optimuma (slika 3.5).

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



Slika 3.5: Prikazan je utjecaj momenta na savladavanje platoa i izbočina. Debljina kružnice predviđava količinu momenta.

Algoritam je primjenjiv na funkcije proizvoljne dimenzionalnosti uz pretpostavku derivabilnosti u svakoj točki. Za proizvoljnu realnu funkciju, uz dobro odabrane hiperparametre, algoritam će konvergirati u jedan od lokalnih optimuma, no algoritam generalno nema garanciju konvergencije u globalni optimum. Garanciju pronalaska globalnog optimuma nudi jedino za trivijalne unimodalne funkcije uz odgovarajuće hiperparametre algoritma (slika 3.2).

Problem odabira hiperparametara proizlazi iz činjenice da u generalno praksi funkcija koja se minimizira nije definirana (već samo skup primjera te funkcije) te se ne može jasno vizualizirati postupak pretrage. Čak i da je definirana funkcija najčešće nije poznata vrijednost globalnog optimuma, a pohlepna pretraga je netraktabilna. Unatoč tome, gradijentni spust efikasno i učinkovito pronalazi optimume koji su dovoljno dobi za većinu praktičnih primjena [37].

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 (poglavlje 3.1.2) te definira što je ishod učenja.

Najčešće nije poznata definicija funkcije gubitka na čitavoj promatranoj domeni već su poznati samo primjeri te funkcije u podatcima koji su izmjereni i koji se smatraju reprezentativnim. Ovdje se prepostavlja da će neuronska mreža ostvariti svojstvo generalizacije, koje je detaljnije objašnjeno u poglavljju 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 aktivacijsku 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 definira se kao inverz funkcije izglednosti da neuronska mreža modelira funkciju predstavljenu primjerima podatkovnog skupa. Funkcija gubitka se minimizira 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, funkcija modela zapisuje se skraćeno kao aproksimacija odnosno predikcija funkcije izlaza podatkovnog skupa te se ovisnost o ulazima podrazumijeva.

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

S obzirom da se želi procijeniti kvaliteta modela na čitavom podatkovnom skupu važno je očekivanje funkcije gubitka (3.6).

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

U problemima regresije, prepostavlja se da izlazi mreže prate Gaussovu distribuciju s jediničnom kovarijacijskom matricom. Njenim uvrštavanjem u negativnu log. izglednost (3.4) dobiva se funkcija 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) dobiva se 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 dobiva se 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 prilagodljivom parametru mreže. Takva formulacija problema omogućuje da svaki parametar mreže ugada se gradijentnim spustom. Dakle, za parametriziranu funkciju $f(x; \theta)$ traže se parametri 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 koja se optimizira 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, dobiva se sljedeći izraz:

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

Derivacija funkcije gubitka za klasifikaciju može se drastično pojednostaviti ako se za funkciju izlaznog sloja mreže odabere funkcija *softmax* (3.14), a izlazi se kodiraju *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:

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

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

Želi li se mrežu učiti gradijentnim spustom, svaki parametar mreže treba imati pristup gradijentu funkcije gubitka. S obzirom da je neuron parametrizirana funkcija, pomoću koje se ulančavanjem gradi mreža, dovoljno je pokazati da pojedini neuron osigurava svojim parametrima pristup gradijentu te da gradijent šalje svojim prethodnicima s kojima je povezan. Da je to ostvarivo dokazuju 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 korištena je funkcija gubitka na jednom podatkovnom paru, no želi se minimizirati očekivanje funkcije gubitka (3.6) na cijelom podatkovnom skupu jer se želi izgraditi model 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 bi se čitavi morali držati u brzoj memoriji. Ovo predstavlja velik problem pri računanju s grafičkim karticama. Očekivanje gubitka služi za procjenu smjera i veličine gradijenta. Ako se gradijent procjenjuje na čitavom podatkovnom skupu on će nas dovesti do prvog optimuma koji najčešće nije dobar, odnosno ovisi o odabiru početne točke.

S druge strane, ako se gradijent računa na pojedinom podatkovnom paru kao u (3.10) ostvareno je stohastičko kretanje jer će svaki primjer usmjeriti gradijent u drugom smjeru. Posljedica toga je duže vrijeme pretrage, no veća otpornost na loše lokalne optimume. Iako će algoritam generalno konvergirati, postupak je zahtjevan jer za svaki primjer podatkovnog skupa treba provesti korak algoritma. Ovaj problem također ograničava svojstvo ubrzanja grafičkim karticama koje nude masivnu paralelizaciju operacija nad podatcima.

Iz navedenih razloga želi se vršiti procjena gradijenta na "nekoliko" primjera. Podskup takvih primjera naziva se **mini-grupa** (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 nije moguće znati je li odabrani raspored idealan za model koji se evaluira. Pristranost se može ublažiti miješanjem podatkovnog skupa između epoha, no skupocjenost 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 su objašnjena dva pristupa: klasični s koeficijentom pomaka (algoritam

1) i s dodatkom momenta (3.3).

Optimizator Adam (engl. *adaptive moments*) dodatno prati povijest momenta drugog reda kojom prilagođava stopu učenja kako bi smanjio razlike između parametara koji su povijesno dobivali velike ili male gradijente, čime se izglađuje prostor parametara. Povijest drugog momenta prate AdaGrad i RMSProp, no Adam dodatno prati povijest prvog momenta i uključuje centriranje momenata. Adam se pokazao vrlo robustnim optimizatorom i vrlo često se koristi za optimizaciju dubokih neuronskih mreža. [14, poglavlje 8.5]

Promjenjiva stopa učenja

Tijekom optimizacije gradijentnim spustom uz fiksnu stopu učenja optimizacija će početi oscilirati oko iste vrijednosti, odnosno srednja vrijednost gubitka je konvergirala. Oscilacija se javlja jer je stopa učenja suviše velika za regije u koje vode različiti gradijenti mini-grupa. U takvim situacijama je potrebno smanjiti stopu učenja, no problem je detekcija takvih situacija. Optimizator Adam prilagođava gradijente skaliranjem, što se može interpretirati i prilagođavanjem stope učenja. Unatoč tome, ponekad je potrebno koristiti dodatne mehanizme smanjivanja stope učenja.

Pristup koji se koristi u ovom radu je zadalu početnu stopu u svakoj iteraciji pomnožiti s konstantom manjom od 1, odnosno stopa se konstantno smanjuje. Korištenje stohastičkog gradijentnog spusta postupak vodi pretragu na sve finije regije. Postupak je sličan postupku simuliranog kaljenja gdje se smanjivanjem temperature sustava smanjuje njegova stohastičnost i postupak konvergira u sve finije regije rješenja. Iako bi u teoriji Adam trebao biti dovoljan, preliminarni eksperimenti pokazuju da dodani mehanizam pomaže učenju mreže.

Dodatno, pokazalo se korisnim koristiti monotono smanjivanje stope učenja tijekom učenja. Smanjivanje stope učenja pri stohastičkom gradijentnom spustu smanjuje varijancu gradijenata te omogućava stabilnije i preciznije učenje. U radu se koristi eksponencijalno opadanje stope učenja po epohama. Prije svake epohe, stopa učenja se izračuna prema formuli (3.23), gdje je e indeks epohe.

$$\eta_e = \eta_0 \cdot 0.99^e, \quad e \in [0, N] \quad (3.23)$$

Inicijalizacija parametara

Inicijalizacija parametara mreže odgovara odabiru početne točke pri gradijentnom spustu. Dobrom inicijalizacijom može se osigurati pronalazak boljeg optimuma te u kritičnim slučajevima i osigurati konvergencija mreže. S obzirom da neuronske mreže posjeduju

jasnu strukturu (što je posebno izraženo u slojevitim arhitekturama) inicijalizacija ima dodatne utjecaje. U potpuno povezanim slojevima želi se postići 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. Kaže se da je došlo do **koadaptacije neurona**.

Inicijalizacija slojevitim nenadziranim učenjem ograničenog Boltzmann-ovog stroja (engl. *restricted Boltzmann machine*) pokazuje se dobrom [26]. 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 normalne distribucije s varijancom koja ovisi o broju ulaznih i izlaznih neurona. [12]

$$W_{i,j} \sim \mathcal{N}(0, \frac{2}{n_{ulaza} + n_{izlaza}}) \quad (3.24)$$

Normalizacija značajki

Podatci najčešće nisu uzorkovani iz normalne distribucije. To stvara problem jer prisljava mrežu da uči srednju vrijednost i varijancu podataka što usporava učenje i stvara numeričke probleme. Stoga je korisno prije korištenja svesti značajke na normalnu distribuciju sa sredinom 0 i varijancom 1. To se izvodi računanjem koeficijenata distribucije te oduzimanjem sredine i dijeljenjem s varijansom.

Koeficijenti se računaju iz značajki podatkovnog skupa za učenje (3.25, 3.26). Koeficijent varijance ovisi o koeficijentu sredine pa se njihovo računanje ne može parallelizirati, no to ne predstavlja velik problem s obzirom da se računaju samo jednim prolazom kroz skup. Prije no što se značajke predaju modelu normaliziraju se formulom (3.27).

$$\vec{\mu} = \frac{1}{N} \sum_{\vec{x} \in \mathcal{D}} \vec{x} \quad (3.25)$$

$$\vec{\sigma}^2 = \frac{1}{N-1} \sum_{\vec{x} \in \mathcal{D}} (\vec{x} - \vec{\mu})^2 \quad (3.26)$$

$$\vec{z} = \frac{\vec{x} - \vec{\mu}}{\vec{\sigma}^2} \quad (3.27)$$

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

Normalizacija nad grupom

(engl. *batch normalization*)

Pri optimizaciji neuronskih mreža u jednoj iteraciji se optimiziraju svi parametri. Problem pri tome je što predikcija optimizatora za parametre jednog sloja prepostavlja da će ostali slojevi ostati netaknuti. To uzrokuje promjenu srednje vrijednosti i varijance odziva koja se akumulira prolaskom kroz mrežu. Iz tog razloga koristi se normalizacija grupom koja standardizira odzive neurona po svakoj mini-grupi. Kako to može smanjiti ekspresivnost neurona, koristi se naknadna afina transformacija s učenim parametrima.

$$\mu = \frac{1}{N} \sum_{i=1}^N h_i \quad \sigma^2 = \frac{1}{N} \sum_{i=1}^N (h_i - \mu)^2 \quad (3.28)$$

$$\hat{h}_i = \frac{h_i - \mu}{\sqrt{\sigma^2 + \epsilon}} \quad y_i = \gamma \cdot h_i + \beta \quad (3.29)$$

3.1.3. Regularizacija

Decizijska granica opisuje točke u prostoru značajki za koje model dvoji između dviju ili više klase. Geometrijski, decizijska granica može se interpretirati kao presjek dviju ili više ploha u hiperprostoru. Pri optimizaciji model prilagođava decizijsku granicu kako bi što bolje razdvojio 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 se kaže da je model počeo učiti šum odnosno da je **prenaučen**. Na prenaučenost su posebno osjetljivi složeni modeli kao što su neuronske mreže.

Kako bi se ublažila prenaučenost modela koriste se tehnike regularizacije. Pri tome se ne smije pretjerati jer suviše snažna regularizacija može ograničiti sposobnost učenja te model može postati **podnaučen**. Stoga je potrebno pronaći dobar omjer između složenosti modela i jačine regularizacije kako bi dobiveni model dobro **generalizirao**. Pri uporabi neuronskih mreža obično se odabire vrlo 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 .

$$\tilde{L}(\theta \mid x, y) = L(\theta \mid x, y) + \alpha \Omega(\theta) \quad (3.30)$$

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.

$$\Omega(\vec{\theta}) = \frac{1}{2} \|\vec{w}\|_2^2 = \frac{1}{2} \sum_i w_i^2 \quad (3.31)$$

$$\vec{\theta}_{i+1} \leftarrow \vec{\theta}_i - \eta \cdot \vec{g}_i + \alpha \cdot \theta \quad (3.32)$$

Rano zaustavljanje

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

Postupak ranog zaustavljanja tretira broj epoha kao hiperparametar koji se pretražuje po liniji. No umjesto da se za svaku vrijednost broja epoha nanovo trenira model, ovdje se trenira jednom i odabire vrijednost za koju model generalizira najbolje. Pri mjerenu generalizacije modela koristi se unakrsna validacija (poglavlje 3.1.4).

Input: model m

```
while (+ +  $i < I \wedge t < T$ ) do
    uči model jednu epohu  $m$ 
    izmjeri gubitak treniranja  $L_{train}$ 
    izmjeri gubitak validacije  $L_{validate}$ 
    if  $L_{train} \in \{NaN, \pm\infty\}$  then
        | izađi iz petlje
    end
    if ( $\left| \frac{L_{train} - L_{train}^*}{L_{train}^*} \right| \leq \delta$ )  $\vee (L_{train} \geq L_{train}^*) \vee (L_{validate} \geq L_{validate}^*)$  then
        |
    else
        |  $t = 0$ 
    end
    if  $L_{validate} < L_{validate}^*$  then
        |  $i^* = i$ 
    end
     $L_{train}^* = \max\{L_{train}^*, L_{train}\}$ 
     $L_{validate}^* = \max\{L_{validate}^*, L_{validate}\}$ 
end
```

Result: pronađen optimalan broj iteracija i^*

Algorithm 3: Algoritam ranog zaustavljanja s dodatnim mehanizmima korišten u radu

U ovom radu detekcija pretreniranosti radi se usporedbom s najboljom pronađenom vrijednosti funkcije gubitka na skupu za validaciju. Ako vrijednost gubitka raste n uzastopnih epoha, to je pokazatelj da se mreža počela pretrenirati. Kao dodatan mehanizam ranijeg zaustavljanja, koji služi za uštedu na vremenu provedenom učenjem, koriste se detekcije divergencije i konvergencije. Divergencija je detektirana ako vrijednost gubitka na skupu za učenje raste, a konvergencija ako se taj gubitak mijenja za manje od $p\%$ najboljeg pronađenog gubitka na skupu za učenje kroz n uzastopnih epoha. Sva tri mehanizma doprinose brojaču epoha i kada brojač dostigne vrijednost n učenje se prekida i odabire se epoha s minimalnim gubitkom na skupu za validaciju. Ako ni jedan mehanizam nije aktiviran, brojač se ponovno postavi na 0.

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 [19, 20]. Preskočne veze omogućavaju direktniji prijenos gradijenta što pomaže kod problema nestajućeg gradijenta u dubokim mrežama (poglavlje 3.1.6). Arhitektura može omogućiti dodatnu paralelizaciju inferencije i učenja tako da se teške operacije raspodijele na više uređaja, a rezultati spoje samo kada je to nužno [27].

Aktivacijske 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 poglavljju 4.1.2. U povratnim neuronskim mrežama za rekurzivne slojeve popularan je tangens hiperbolni opisan u poglavljju 4.1.14. U poglavljju 4 navedene su i opisane brojne funkcije te je njihov utjecaj ispitana u poglavljju 7.

Procjena generalizacije i odabir modela

Skup podataka kojim se uči model najčešće ne opisuje stvarnu funkciju potpuno, već sadrži primjere koji se smatraju reprezentativnim i koji su dovoljni za njeno modeliranje. Kako bi se procijenilo koliko dobro naš model procjenjuje stvarnu funkciju model se ispituje na podatcima koji nisu korišteni prilikom učenja, odnosno na neviđenim podatcima. Ta se metoda zove **unakrsna validacija** (engl. *crossvalidation*), a skupovi se nazivaju **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 će se nepotpuno ili pristrano provesti testiranje 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 model se optimizira, a skupom za validaciju ispituje se generalizaciju modela. Između kombinacija hiperparametara odabire se ona za koju model najbolje generalizira na skupu za validaciju te se njome nanovo optimizira model na originalnom skupu za učenje i dobiva konačna mjera generalizacije 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 4.

```

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 4: Ugniježđena unakrsna provjera

Mjere generalizacije

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

$\hat{y} \setminus y$	\top	\perp	
\top	TP	FP	(3.33)
\perp	FN	TN	

Pri binarnoj klasifikaciji definirana je **matrica zabune** (3.33) koja sadrži četiri

elementa (3.34) koja definiraju vrstu pogotka i pogreške.

$$\begin{aligned}
 \text{Stvarno pozitivni: } TP &= \sum_{(x,y) \in \mathbb{D}} \mathbb{1}\{h(x) = \top \wedge y = \top\} \\
 \text{Stvarno negativni: } TN &= \sum_{(x,y) \in \mathbb{D}} \mathbb{1}\{h(x) = \perp \wedge y = \perp\} \\
 \text{Lažno pozitivni: } FP &= \sum_{(x,y) \in \mathbb{D}} \mathbb{1}\{h(x) = \top \wedge y = \perp\} \\
 \text{Lažno negativni: } FN &= \sum_{(x,y) \in \mathbb{D}} \mathbb{1}\{h(x) = \perp \wedge y = \top\}
 \end{aligned} \tag{3.34}$$

Iz tih skupova se tada grade složenije mjere. **Točnost** je mjera kojom se iskazuje postotak točno pogodjenih primjera:

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.35}$$

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 bit će nezamjetna. Stoga se kod nebalansiranih setova češće koristi **F₁ mјera**, koja uzima u obzir **preciznost** klasifikatora u razlikovanju pozitivnih primjera od negativnih (3.36) i njegov **odziv** odnosno obuhvat svih pozitivnih primjera testnog skupa (3.37). F₁ mјera je definirana kao harmonijska sredina između preciznosti i odziva (3.38). Postoji i generalizirana mјera F_β koja dodjeljuje veću težinu preciznosti ili odzivu (3.39), no u ovom radu koristi se samo F₁ koja pridjeljuje jednaku težinu. Harmonijska sredina se koristi jer je najstroža između Pitagorinih mјera za sredinu kao što prokazuje slika 3.6.

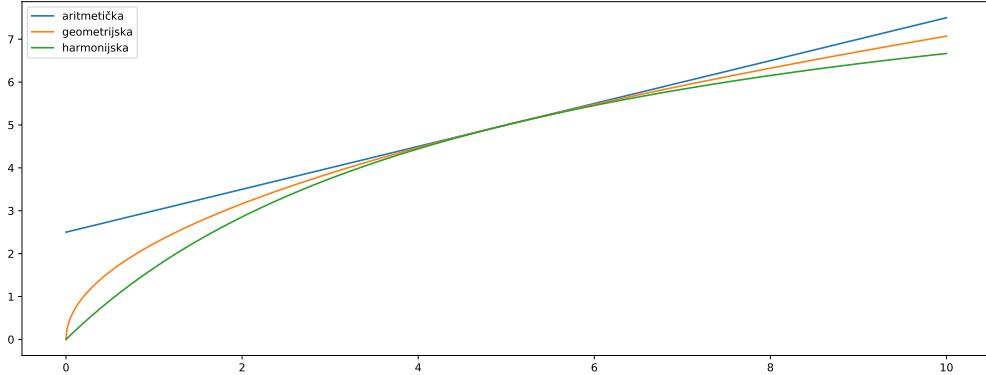
$$\text{Preciznost: } P = \frac{TP}{TP + FP} \tag{3.36}$$

$$\text{Odziv: } R = \frac{TP}{TP + FN} \tag{3.37}$$

$$F_1 : \quad F_1 = 2 \cdot \frac{P \cdot R}{P + R} \tag{3.38}$$

$$F_\beta : \quad F_\beta = (1 + \beta^2) \cdot \frac{P \cdot R}{\beta^2 \cdot P + R} \tag{3.39}$$

Mjere binarne klasifikacije mogu se primijeniti pri **višeklasnoj klasifikaciji**, no matrica zabune je dimenzija $C \times C$ gdje je C broj klasa. Elementi matrice računaju se



Slika 3.6: Pitagorine mjere za sredinu između dviju vrijednosti. Po apscisi su brojevi koji se uspoređuju s brojem 5, a po ordinati vrijednosti mjere.

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

$$\begin{aligned}
 \text{Stvarno pozitivni: } & TP_i = \sum_{(x,y) \in \mathbb{D}} \mathbb{1}\{h(x) = C_i \wedge y = C_i\} \\
 \text{Stvarno negativni: } & TN_i = \sum_{(x,y) \in \mathbb{D}} \mathbb{1}\{h(x) \neq C_i \wedge y \neq C_i\} \\
 \text{Lažno pozitivni: } & FP_i = \sum_{(x,y) \in \mathbb{D}} \mathbb{1}\{h(x) = C_i \wedge y \neq C_i\} \\
 \text{Lažno negativni: } & FN_i = \sum_{(x,y) \in \mathbb{D}} \mathbb{1}\{h(x) \neq C_i \wedge y = C_i\}
 \end{aligned} \tag{3.40}$$

Za izračun složenijih mjera poput točnosti i F_1 mjere mora se računati prosjek po klasama. Koriste se 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 [33].

$$Acc^M = \sum_{i=1}^K \frac{Acc_i}{K} \quad P^M = \sum_{i=1}^K \frac{P_i}{K} \quad R^M = \sum_{i=1}^K \frac{R_i}{K} \quad F_1^M = \sum_{i=1}^K \frac{F_{1;i}}{K} \tag{3.41}$$

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.

$$\begin{aligned}
 Acc^\mu &= \frac{\sum_i (TP_i + TN_i)}{\sum_i (TP_i + TN_i + FP_i + FN_i)} = Acc^M \\
 FP = FN \implies P^\mu = R^\mu = F_1^\mu &= \frac{\sum_i TP_i}{\sum_i (TP_i + FP_i)}
 \end{aligned} \tag{3.42}$$

Pretraživanje po rešetci

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

Iako se optimalni hiperparametri 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 iskaže kako doći do tih parametara što optimizaciju neuronske mreže čini netrivijalnom [14, poglavje 6.4.1].

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 koja se aproksimira kako bi se smanjio potreban broj neurona. Primjer su funkcije simetrične oko neke osi. Ako skriveni slojevi mreže vrše preklapanje te funkcije preko pravca u njenom hiperprostoru, uzastopnim preklapanjem dobiva se sve jednostavnija funkcija. Preklapanje mogu vršiti po dijelovima linearne aktivacijske funkcije poput ReLU i Maxout. Dakako, ne postoji garancija da stvarna funkcija zadovoljava svojstvo simetrije, no u praksi dublje mreže generaliziraju bolje [27, 39, 19, 20]. Postoje i druge interpretacije utjecaja dubine, poput svojstva dekompozicije zadatka na manje cjeline ili interpretacije neuronske mreže kao računalnog programa, no one nadilaze temu ovog rada. [14, poglavje 6.4.1]

Generalizacija

Model sa svojstvom generalizacije dobro modelira stvarnu funkciju te ispravno radi i na neviđenim primjerima. Ovo je posebno važno pri klasifikaciji slika gdje za pojedinoj klasu (npr. automobil) postoji neprebrojivo mogućih slika s različitim modelom, bojom ili kutom gledanja automobila. No dostupno je svega nekoliko stotina tisuća primjera koji se smatraju reprezentativnim za tu klasu i njima se želi izgraditi klasifikator koji je robustan na većinu perturbacija slike. Kako bi model što bolje naučio stvarnu funkciju potreban je velik broj označenih uzoraka. S obzirom da je označavanje podataka često vrlo skupocjeno, istražuju se metode automatizacije poput polu-nadziranog učenja i aktivnog učenja [11]. Augmentacija podataka raznim metodama stvara više primjera iz jednog uzorka stvarne funkcije čime se proširuje podatkovni skup bez potrebe za označavanjem te podiže generalizacijsku moć modela. No, čak i uz navedene metode model i dalje neće dovoljno dobro generalizirati stvarnu funkciju što potvrđuje postojanje neprijateljskih primjera [43].

3.1.6. Problemi

Odabir hiperparametara

Arhitektura, prijenosne funkcije i parametri definiraju neuronsku mrežu te njihov pravilan odabir značajno utječe na performanse neuronske mreže. Nažalost nije ih moguće optimalno odabrati u zatvorenom obliku, već se to svodi na problem pretraživanja kao što je pretraživanje po rešetci (poglavlje 3.1.4). Arhitekture mogu poprimiti vrlo složene oblike kao što je predstavljeno radovima Srivastava et al. [39], He et al. [19], Huang et al. [20], Szegedy et al. [42] i Redmon et al. [37] te se najčešće grade ručno s maksimalnom traktabilnom složenošću mreže. Detaljnije o odabiru aktivacijskih funkcija napisano je u poglavlju 4.

Nestajući gradijent

Prolaskom kroz duboku mrežu gradijent se množi matricama težina \underline{W}_i . Uzastopno množenje može dovesti do problema ako su svojstvene vrijednosti matrice udaljene od ± 1 . Rastavom matrice u umnošku ponovljenom t puta (3.43) vidljiv je problem. Svojstvene vrijednosti veće od 1 će rasti, a one manje od 1 će se smanjivati. [14, poglavlje 8.2.5]

$$\underline{W}^t = (\underline{V} \cdot \text{diag}(\lambda) \cdot \underline{V}^{-1})^t = \underline{V} \cdot \text{diag}(\lambda)^t \cdot \underline{V}^{-1} \quad (3.43)$$

Problem eksplodirajućeg gradijenta rješava se odrezivanjem gradijenta koji je prevelik. Problemu nestajućeg gradijenta moguće je pristupiti uvođenjem preskočnih veza u arhitekturu [39, 19, 20] te odabirom aktivacijske funkcije s linearnim svojstvima [1].

4. Aktivacijske funkcije

Aktivacijska funkcija služi unošenju nelinearnosti u neuronsku mrežu i ima utjecaj na njeno učenje. Prilikom inferencije aktivacijska funkcija stvara nelinearnosti u decizijskoj granici koje su parametrizirane parametrima neurona. U slojevitim arhitekturama aktivacijske funkcije se primjenjuju uzastopno s različitim parametrima neurona. Time vrše nelinearnu projekciju prostora iz jedne dimenzije u drugu te utječu na jednostavnost i kvalitetu konačne klasifikacije. Prilikom učenja neuronske mreže važan je utjecaj derivacije aktivacijske funkcije na gradijent pri širenju unatrag. Prolaskom unatrag gradijent se množi s derivacijom aktivacijske funkcije (3.18) što može izazvati nestajanje ili eksploziju gradijenta.

Za odabir aktivacijske funkcije ne postoji jasno pravilo. U praksi se odabiru po empirijskim rezultatima i brzinom izvođenja. Iako neke funkcije imaju teorijski potkrijepljena svojstva to ih u praksi ne čini najboljima. Tipično se odabire *ReLU*, a za skrivene slojeve rekurzivnih modela odabire se *tanh*.

Periodične aktivacijske funkcije

U rezultatima pretrage ovog rada i Ramachandran et al. [36] često pojavljuju se periodične funkcije. Stoga su u nastavku opisani radovi na temu svojstva periodičkih funkcija u neuronskim mrežama koje je autor uspio pronaći.

Periodične aktivacijske funkcije stvaraju brojne lokalne optimume u funkciji gubitka koji otežavaju učenje neuronskih mreža. S obzirom da se mreže treniraju 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 razliku u performansama iako oko 40% ulaza neurona završava izvan intervala monotonosti, što upućuje da modeli ne ovise snažno o periodičnosti funkcije. U primjeni na rekurziv-

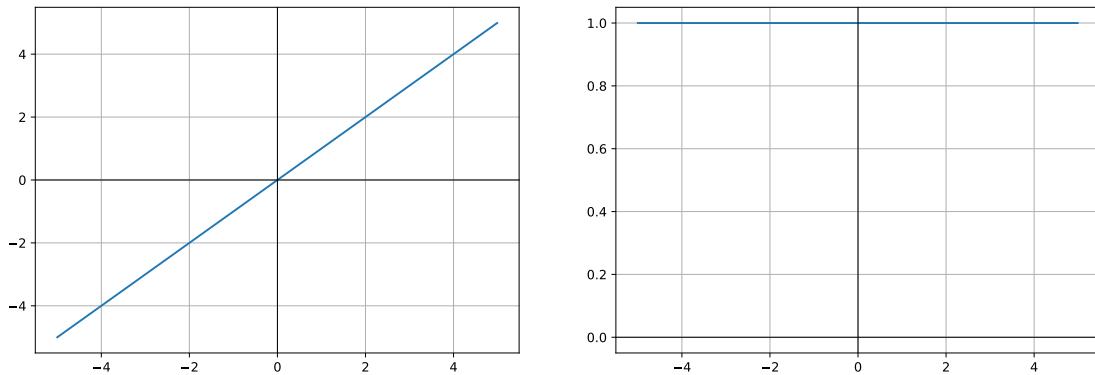
nim modelima ostvaruje prednost nad monotonim tangensom hiperbolnim, no autori navode teškoću optimizacije u nekim situacijama. [35]

4.1. Popularne aktivacijske funkcije

U nastavku su navedene popularne aktivacijske 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.

4.1.1. Funkcija identiteta

(engl. *Identity function*)



Slika 4.1: Funkcija identiteta i njena derivacija

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

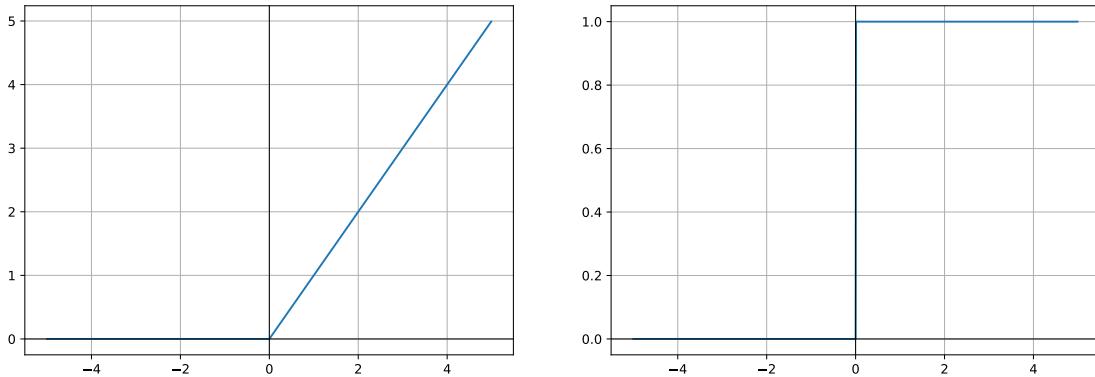
Funkcija identiteta je jednostavna i brza, no njome neuronska mreža može naučiti samo linearne funkcije. Ako se primjeni na dva uzastopna sloja neuronske mreže, konačna će funkcija ponovno biti linearna što znači da se mreža ne može koristiti na nelinearnim podatcima. Derivacija funkcije je konstanta 1 što znači da je gradijent prolaskom kroz nju očuvan. Funkcija identiteta često se koristi kao funkcija aktivacijske sloja kod regresije, no u dubokim arhitekturama [19, 20] i novijim aktivacijskim

funkcijama [36, 1] koristi se paralelno s nelinearnim funkcijama.

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

4.1.2. Zglobnica ili ispravljena linearna jedinica (ReLU)

(engl. *Rectified linear unit*)



Slika 4.2: Funkcija ReLU i njena derivacija

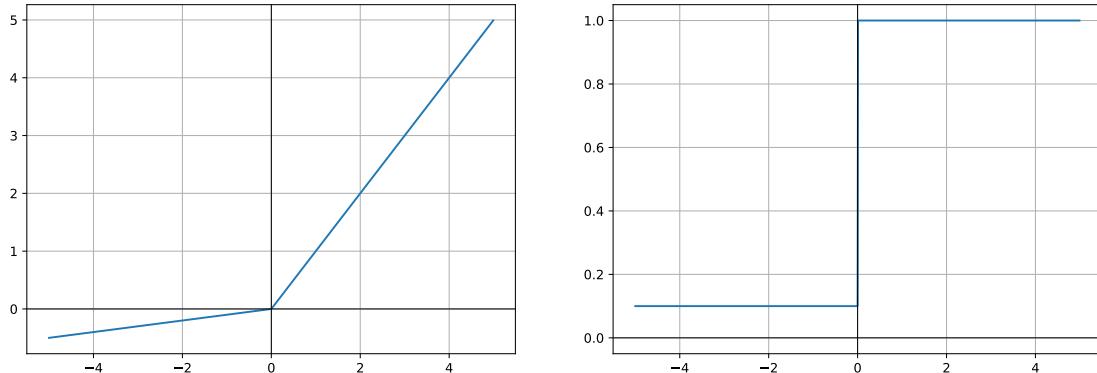
$$\begin{aligned}
 f(x) &= \begin{cases} x, & \text{ako } x > 0 \\ 0, & \text{inače} \end{cases} & f'(x) &= \begin{cases} 1, & \text{ako } x > 0 \\ 0, & \text{inače} \end{cases} \\
 &= \max(0, x) & &
 \end{aligned} \tag{4.3}$$

Funkcija ReLU svoje korijene povlači iz rada bioloških neurona, a u dubokom učenju je zamijenila dotadašnju sigmoidu i tangens hiperbolni. Zahvaljujući prizemljenosti na negativnoj domeni ReLU omogućava neuronskoj mreži učenje rijetke reprezentacije. Njihovim kombiniranjem u dubokoj mreži dobiva se model s eksponentijalno puno linearnih regija koje dijele zajedničke parametre [34]. Zahvaljujući linearnosti na pozitivnoj domeni funkcija ne doprinosi nestajanju ni eksploziji gradijenta, a sam izračun funkcije i derivacije je vrlo brz i efikasan. [13]

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 [13]. Drugi problem je u neograničenosti funkcije u pozitivnoj domeni, no on se rješava regularizacijom koja ograničava veličinu ulaza u neuron [13].

4.1.3. Propusna ispravljena linearna jedinica (LReLU)

(engl. *Leaky ReLU*)



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

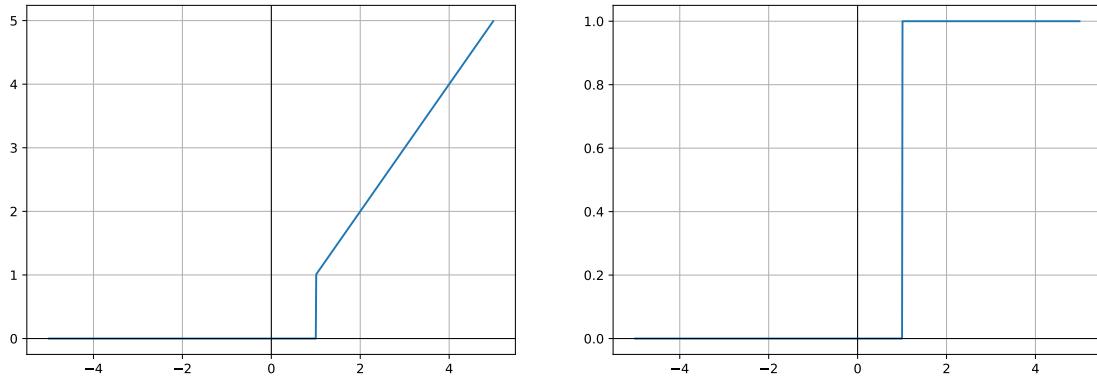
$$f(x) = \begin{cases} x, & \text{ako } x > 0 \\ \alpha x, & \text{inače} \end{cases} \quad f'(x) = \begin{cases} 1, & \text{ako } x > 0 \\ \alpha, & \text{inače} \end{cases} \quad (4.4)$$

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

Problem funkcije ReLU (poglavlje 4.1.2) je što postoji mogućnost pojave mrtvih neurona, koji su uvijek u neaktivnom području i kroz njih ne teče gradijent. Taj problem efektivno smanjuje kapacitet modela pod cijenu nepotrebnog memorijskog opterećenja i smanjene brzine izvođenja. Iz navedenih razloga uvedena je propusna ReLU funkcija koja u "neaktivnom" području propušta mali gradijent. Rezultati na zadatku prepoznavanja govora pokazuju da je propusna ReLU funkcija ekvivalentna standardnom ReLU, no rezultati su prikazani na relativno plitkim arhitekturama (do 4 skrivena sloja). Tipična vrijednost parametra α je 0.01. [28]

4.1.4. Ispravljena linearna jedinica s pragom (ThReLU)

(engl. *Thresholded ReLU*)

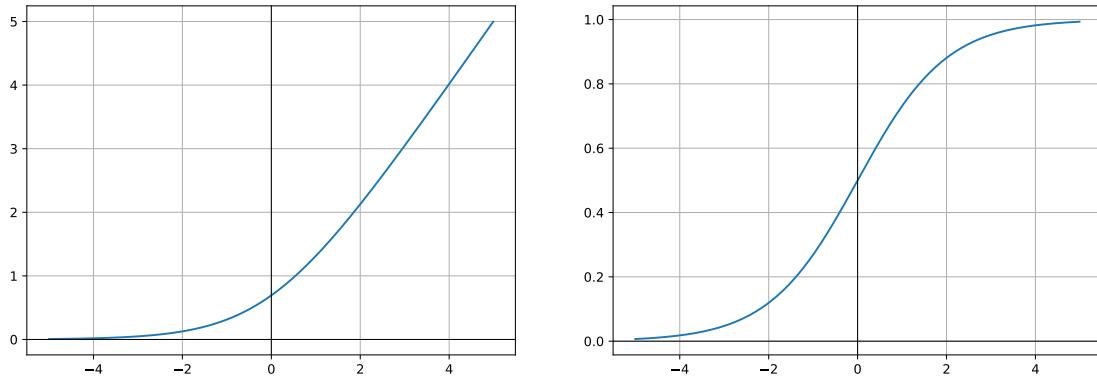


Slika 4.4: Funkcija ThReLU i njena derivacija

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

Funkcija je generalizacija funkcije *ReLU* (poglavlje 4.1.2) kojoj je vrijednost praga 0. Predložena je kao rješenje za pojavu negativnih pragova pri učenju autoenkodera s metodama regularizacije kao što su sažimajući autoencoder (engl. *contractive autoencoder*) i autoencoder za otklanjanje šuma (engl. *denoising autoencoder*). Primjenom funkcije bez regularizacije postiže se usporedivi ili bolji rezultati. Za vrijednost praga tipično se uzima $\theta = 1$. [30]

4.1.5. Softplus



Slika 4.5: Softplus i njegova derivacija

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

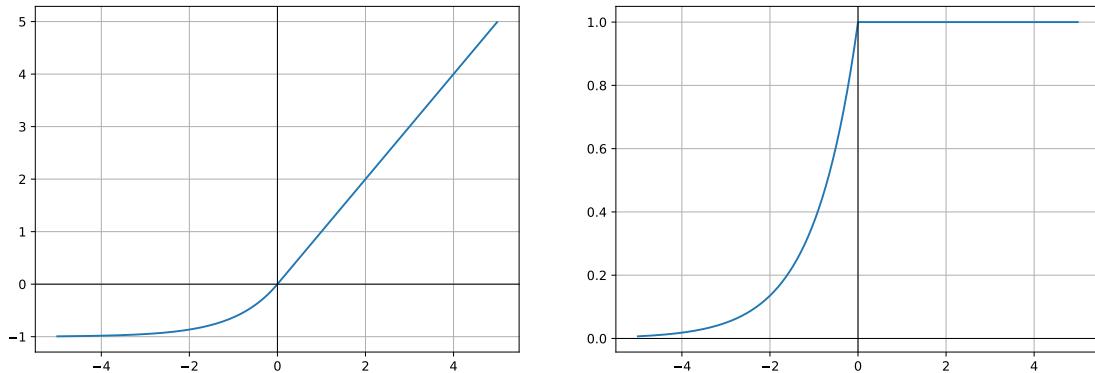
Funkcija *Softplus* je strogo pozitivna monotono rastuća glatka funkcija nalik na *ReLU*. Vrlo je bliska beskonačnom zbroju binomnih jedinica odmaknutih za 1 koje se mogu koristiti u ograničenim Boltzmann-ovim strojevima. [34]

$$\sum_{i=1}^{\infty} \sigma(x - i + 0.5) \approx \log(1 + e^x) \quad (4.7)$$

U odnosu na *ReLU*, funkcija nema problem nestajućeg gradijenta jer propušta gradijent u negativnoj domeni i ostvaruje veće vrijednosti od *ReLU*. To je svojstvo posebna prednost nad *ReLU* i sigmoidom pri korištenju tehnike Dropout kojom se blokira doprinos velikog broja neurona. Unatoč teoretskim prednostima funkcija rijetko postiže zadovoljive rezultate. [18]

4.1.6. Eksponencijalno-linearna jedinica (ELU)

(engl. *Exponential linear unit*)



Slika 4.6: Funkcija ELU i njena derivacija

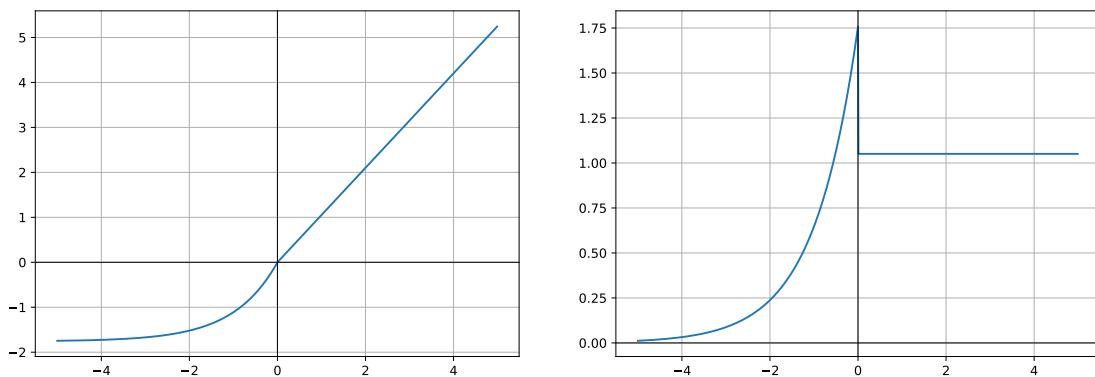
$$f(x) = \begin{cases} x, & \text{ako } x > 0 \\ \alpha(e^x - 1), & \text{inače} \end{cases} \quad f'(x) = \begin{cases} 1, & \text{ako } x > 0 \\ \alpha e^x, & \text{inače} \end{cases} \quad (4.8)$$

Funkcija *ELU* je linearna na pozitivnoj domeni, a u negativnoj domeni postepeno prelazi u zasićenje. Uvedena je kako bi umanjila efekt otklona izlaza neurona, koja nastaje kada je srednja vrijednost izlaza otklonjena od 0. Aktivacije koje nisu centrirane usporavaju učenje mreže i stoga ih je potrebno centrirati. Jedan pristup je centriranje normalizacijom grupa, a alternativni je odabirom aktivacijske funkcije. *ReLU* nije centriran jer je u pozitivnoj domeni linearan, a u negativnoj 0. Tangens hiperbolni je primjer centrirane funkcije. *ELU* je djelom linearan pa nema problem nestajućeg

gradijenta te brže prelazi u zasićenje i time smanjuje razlike između neaktivnih neurona za različite ulazne argumente što ga čini otpornim na šum. Hiperparametar α određuje negativnu vrijednost zasićenja i tipično se postavlja na 1. Funkcija postiže bolje rezultate od $ReLU$ s normalizacijom grupe. Kada se koristi s normalizacijom grupe ostvaruje nešto lošije rezultate, no i dalje bolje od $ReLU$ i izvedenica. [3]

4.1.7. Skalirana eksponencijalno-linearna jedinica (SELU)

(engl. *Scaled exponential linear unit*)

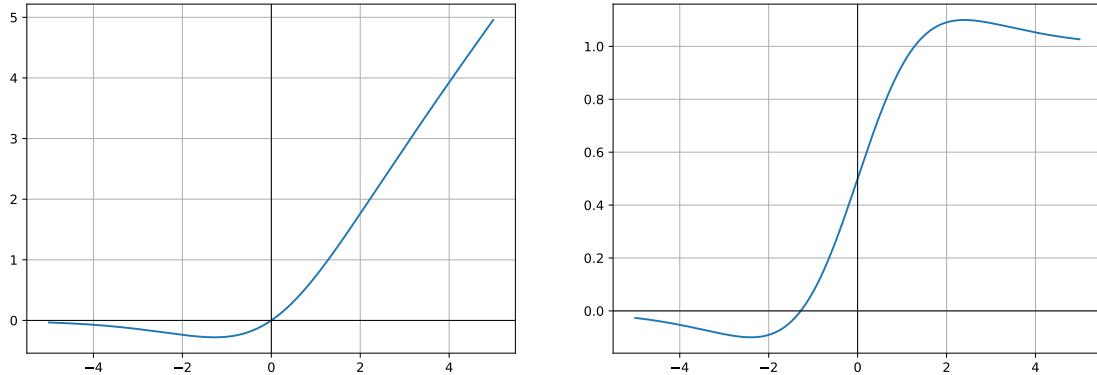


Slika 4.7: Funkcija SELU i njena derivacija

$$f(x) = \lambda \begin{cases} x, & \text{ako } x > 0 \\ \alpha(e^x - 1), & \text{inače} \end{cases} \quad f'(x) = \lambda \begin{cases} 1, & \text{ako } x > 0 \\ \alpha e^x, & \text{inače} \end{cases} \quad (4.9)$$

Aktivacijska funkcija $SELU$ izgrađena je za ostvarivanje samo-normalizirajuće neuronske mreže kojom je moguće izgraditi duboke potpuno povezane modele bez potrebe za eksplisitnim tehnikama normalizacije. Funkcija nasljeđuje svojstvo otpornosti na nestajuće i eksplodirajuće gradijente od funkcije ELU (poglavlje 4.1.6). Primjenom funkcije na mreži s prilagođenom inicijalizacijom i regularizacijom ostvaruju se rezultati bolji od ekvivalentnih mreža s primjenjenim tehnikama normalizacije (normalizacija težina, slojeva ili grupom) te specijaliziranih mreža (HighwayNet [39], ResNet [19]) na brojnim podatkovnim skupovima. Performanse samo-normalizirajućih mreža potkrijepljene su teorijom te su definirane vrijednosti parametara $\alpha \approx 1.6733$ i $\lambda \approx 1.0507$ za normalizirane ulaze u sloj. [24]

4.1.8. Swish



Slika 4.8: Funkcija Swish i njena derivacija

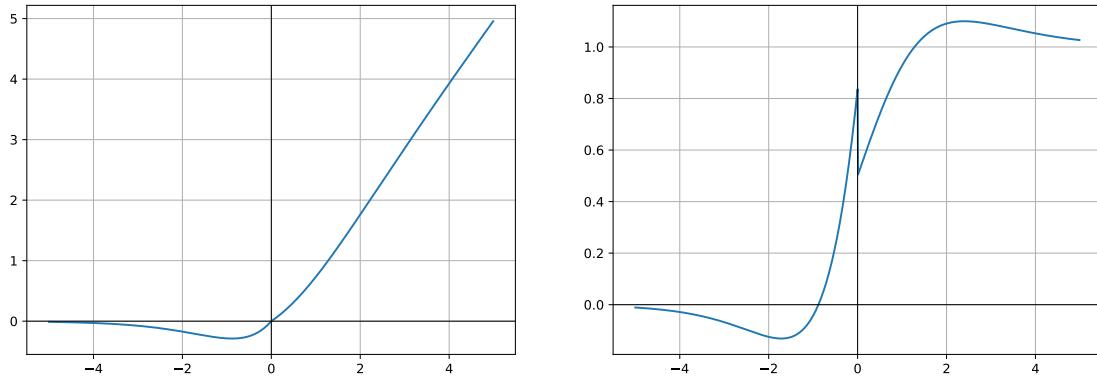
$$f(x) = x \cdot \sigma(\beta x) \quad f'(x) = \sigma(\beta x) \cdot (1 + \beta x \cdot (1 - \sigma(\beta x))) \quad (4.10)$$

β može biti definiran ili je učeći parametar

Funkcija Swish je pronađena pretragom temeljenom na potpornom učenju. Funkcija zadovoljava strukturu kompozicije binarne (umnožak) i dviju unarnih operacija (identitet i sigmoida). Ispitivanje funkcije na nekoliko podatkovnih skupova i na nekoliko dubokih arhitektura s preskočnim vezama ostvaruje konzistentno bolje rezultate u odnosu na nekoliko dotadašnjih najboljih funkcija bez mijenjanja originalnih hiperparametara. Udobina funkcije na negativnoj domeni pokazuje se važnom jer se većina izlaza nalazi upravo tamo. Parametar β interpolira funkciju između linearne funkcije ($\beta = 0$) i funkcije $ReLU$ ($\beta \rightarrow \infty$). Parametar β može biti učeći parametar koji pospješuje učenje nekih arhitektura, no tipično se postavlja na fiksnu vrijednost 1. [36]

4.1.9. ELiSH

(engl. *Exponential Linear Sigmoid Squashing*)



Slika 4.9: Funkcija ELiSH i njena derivacija

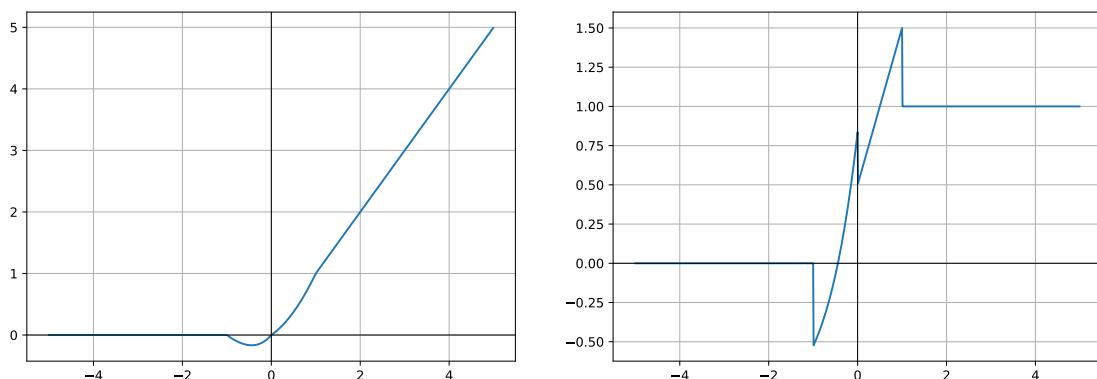
$$f(x) = \begin{cases} swish(x), & x \geq 0 \\ ELU(x) \cdot \sigma(x), & \text{inac} \end{cases} \quad (4.11)$$

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

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 (4.10), a negativna je umnožak funkcije ELU (4.8) i sigmoide (4.15). 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. [1]

4.1.10. Tvrdi ELiSH

(engl. *Hard ELiSH*)



Slika 4.10: Funkcija Tvrdi ELiSH i njena derivacija

$$f(x) = \begin{cases} x \cdot a(x), & x \geq 0 \\ ELU(x) \cdot a(x), & \text{inače} \end{cases} \quad (4.12)$$

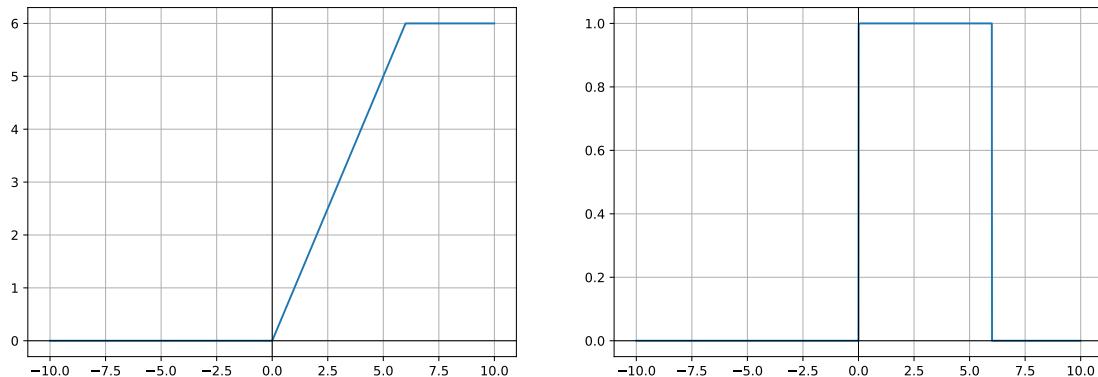
$$f'(x) = \begin{cases} a(x) + x \cdot a'(x), & x \geq 0 \\ ELU'(x) \cdot a(x) + x \cdot ELU(x) \cdot a'(x), & \text{inače} \end{cases}$$

$$a(x) = \max(0, \min(1, \frac{x+1}{2})) \approx HardSigmoid(x) \quad (4.13)$$

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

Ova funkcija je aproksimacija funkcije ELiSH opisane u poglavlju 4.1.9. Uvedena je za potrebe bržeg izvođenja sigmoide i nasljeđuje svojstva ELiSH funkcije. [1]

4.1.11. Ograničena ispravljena linearna jedinica (ReLU-n)



Slika 4.11: Funkcija ReLU6 i njena derivacija

$$f(x) = \begin{cases} 0, & \text{ako } x < 0 \\ x, & \text{ako } x \in [0, n] \\ n, & \text{inače} \end{cases} \quad f'(x) = \begin{cases} 1, & \text{ako } x \in [0, n] \\ 0, & \text{inače} \end{cases} \quad (4.14)$$

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

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

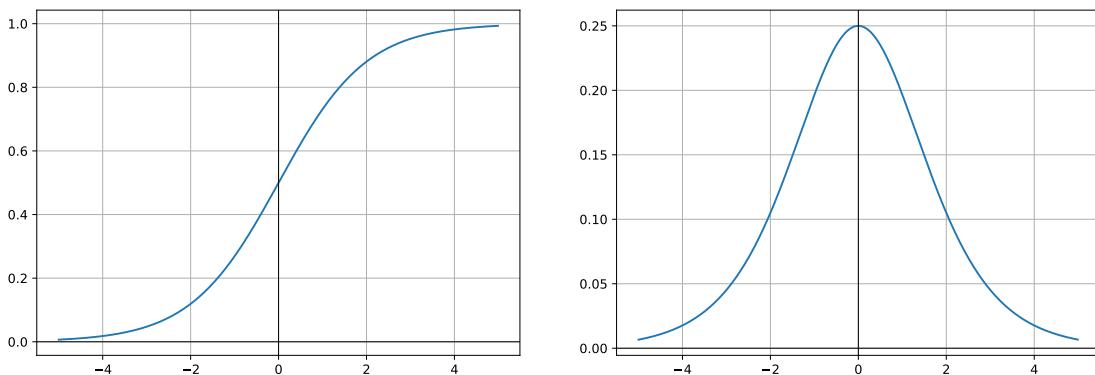
Ograničena ReLU funkcija uvedena je kako bi ograničeni Boltzmann-ov stroj (engl. *Restricted Boltzmann machine*) brže naučio rijetke značajke, koje kasnije ugađa konvolucijski klasifikator. Za razliku od ReLU koji se može interpretirati kao kompozicija

beskonačno mnogo identičnih Bernoullijevih jedinica translatiranih po domeni, ograničeni ReLU interpretira se kao kompozicija n Bernoullijevih jedinica. Za vidljivi sloj autori koriste ReLU1, a za skrivene slojeve ReLU6. [26]

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.

4.1.12. Sigmoida (σ)

(engl. *Sigmoid*)



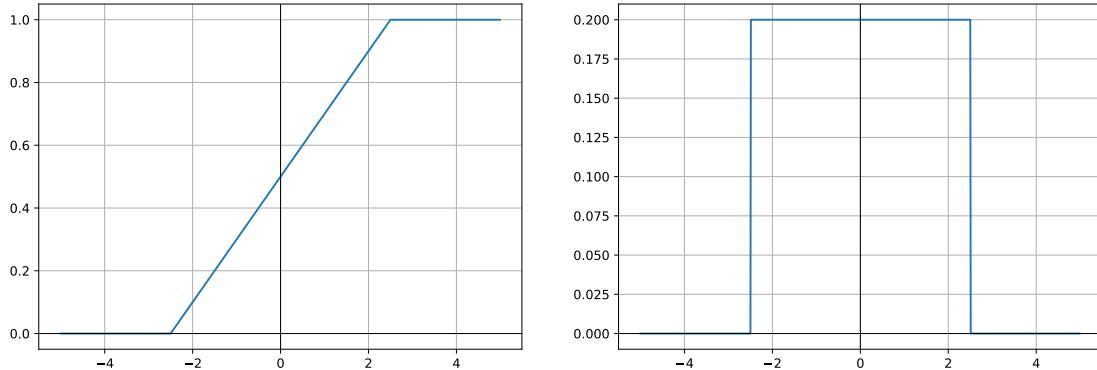
Slika 4.12: Sigmoida i njena derivacija

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

Sigmoida kao aktivacijska funkcija svoje korijene vuče iz biologije [5, str. 16]. Funkcija je kontinuirana i strogo pozitivna sa zasićenjem u 1 na pozitivnoj domeni i zasićenjem u 0 na negativnoj domeni. Funkciju se može smatrati relaksiranom funkcijom skoka. Za velike parametre neurona sigmoida se približava funkciji skoka te za malu perturbaciju ulaza oko 0 daje veliku promjenu izlaza što otežava postupak učenja. Derivacija sigmoide je zvonolika funkcija s vrhom u 0.25. Množenjem gradijenta s njenom derivacijom (3.17) gradijent se brzo smanjuje što pridonosi pojavi nestajućeg gradijenta. Iz tog razloga je gotovo nemoguće učiti duboke modele sa sigmoidom te su razvijene nove funkcije s ciljem očuvanja gradijenta.

4.1.13. Tvrda sigmoida

(engl. *Hard sigmoid*)



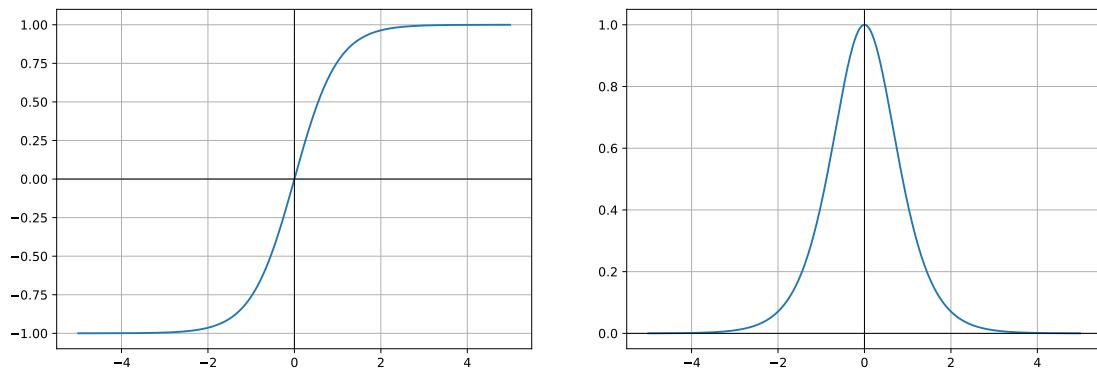
Slika 4.13: Tvrda sigmoida i njena derivacija

$$f(x) = \min(1, \max(0, 0.2x + 0.5)) \quad f'(x) = \begin{cases} 0.2, & \text{ako } |x| \leq 2.5 \\ 0, & \text{inače} \end{cases} \quad (4.16)$$

Funkcija je tvrda inačica sigmoide nastala Taylor-ovim razvojem prvog reda oko nule. Funkcija omogućava učenje oštrih granica, pod cijenu zaustavljanja gradijenta u regijama zasićenja. U regije zasićenja funkcije moguće je dodati šum koji sprječava zaustavljanje gradijenta i pomaže optimizaciji. Kako učenje napreduje, šum se postepeno smanjuje kako bi se osigurala konvergencija algoritma. [16]

Iako se u ovom radu pri primjeni tvrde funkcije ne dodaje šum, funkcija ostvaruje bolje rezultate od popularne funkcije *ReLU*.

4.1.14. Tangens hiperbolni (tanh)



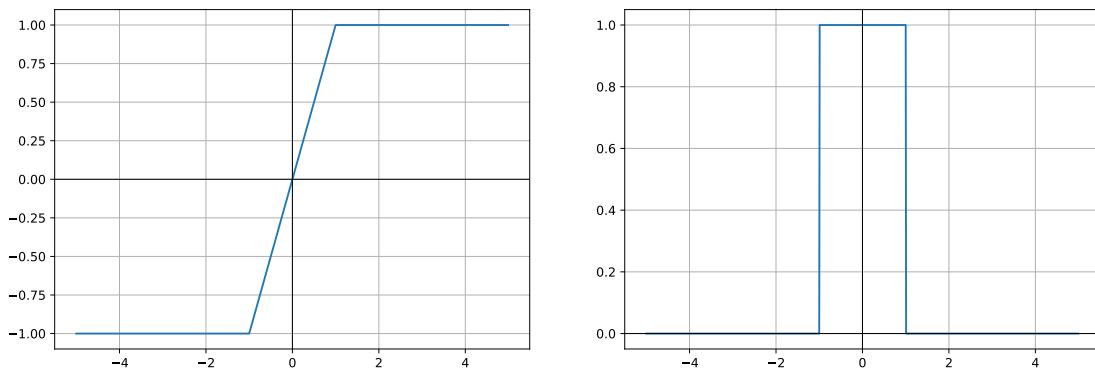
Slika 4.14: Funkcija tanh i njena derivacija

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad f'(x) = 1 - \tanh^2(x) \quad (4.17)$$

Tangens hiperbolni je sličan sigmoidi, no nije strogo pozitivan. Zasićenja u ± 1 čine ga pogodnim za rekurzivne modele koji zahtijevaju sposobnost dodavanja i oduzimanja vrijednosti između slojeva. Derivacija je oštira i poprima veće vrijednosti u odnosu na sigmoidu, s vrhom u 1, što je pogodnije za prijenos gradijenta pri učenju dubokih i rekurzivnih modela.

4.1.15. Tvrdi tangens hiperbolni

(engl. *Hard tanh*)



Slika 4.15: Tvrdi tanh i njegova derivacija

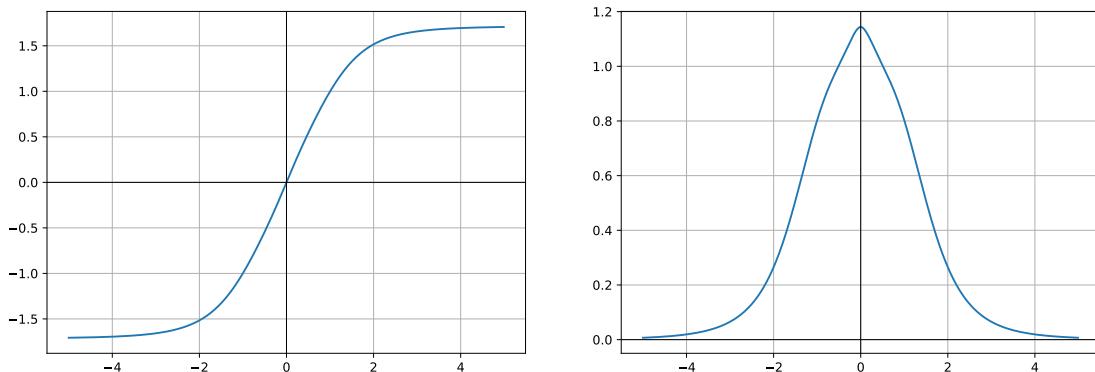
$$f(x) = \begin{cases} -1, & \text{ako } x < 1 \\ x, & \text{ako } |x| \leq 1 \\ 1, & \text{inače} \end{cases} \quad f'(x) = \begin{cases} 0, & \text{ako } x < 1 \\ 1, & \text{ako } |x| \leq 1 \\ 0, & \text{inače} \end{cases} \quad (4.18)$$

Tvrdi tangens hiperbolni je linearizirana aproksimacija tangensa hiperbolnog korištena u radu Collobert [4]. Iako nema glatkoću originalne funkcije i ne propušta gradijent izvan intervala $[-1, 1]$, jednako dobro pomaže generalizaciji mreže. Koristan je zbog veće brzine izvođenja funkcije i derivacije u odnosu na tangens hiperbolni.

U radu Gülçehre et al. [16] u regije zasićenja dodaje se šum kao što je opisano u poglavlju 4.1.13. Iako se u ovom radu ne dodaje šum kao ni kod tvrde sigmoide, funkcija ostvaruje rezultate bolje od popularne funkcije *ReLU*.

4.1.16. Racionalna aproksimacija tanh

(engl. *Rational tanh*)



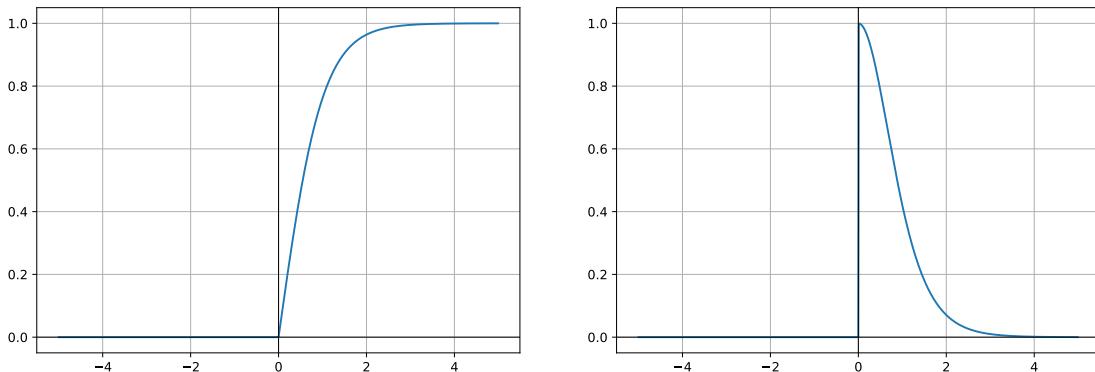
Slika 4.16: Racionalni tanh i njegova derivacija

$$\begin{aligned}
 f(x) &= 1.7159 \cdot \tanh^*(\frac{2}{3}x), & \tanh^*(x) &= \text{sgn}(x)(1 - \frac{1}{1 + |x| + x^2 + 1.41645 \cdot x^4}) \\
 f'(x) &= 1.7159 \cdot \frac{2}{3} \cdot \tanh^{*\prime}(\frac{2}{3}x), & \tanh^{*\prime}(x) &= \frac{1 + \text{sgn}(x) \cdot (2x + 4 \cdot 1.41645 \cdot x^3)}{(1 + |x| + x^2 + 1.41645 \cdot x^4)^2}
 \end{aligned} \tag{4.19}$$

Racionalna aproksimacija tangensa hiperbolnog prati originalnu funkciju s relativnom greškom manjom od 1.8%, no uvelike ubrzava njegovo računanje. Umjesto zahtjevnog računanja eksponenta prirodne konstante ova aproksimacija zahtjeva samo 11 naredbi (uzastopno kvadriranje ulaza). Funkcija se pokazala boljom od *ReLU* na problemu detekcije lica u slikama. Odabir konstanti kojima su skalirani ulaz i izlaz funkcije nije opisan. [22]

4.1.17. Ispravljeni tanh

(engl. *Rectified tanh*)



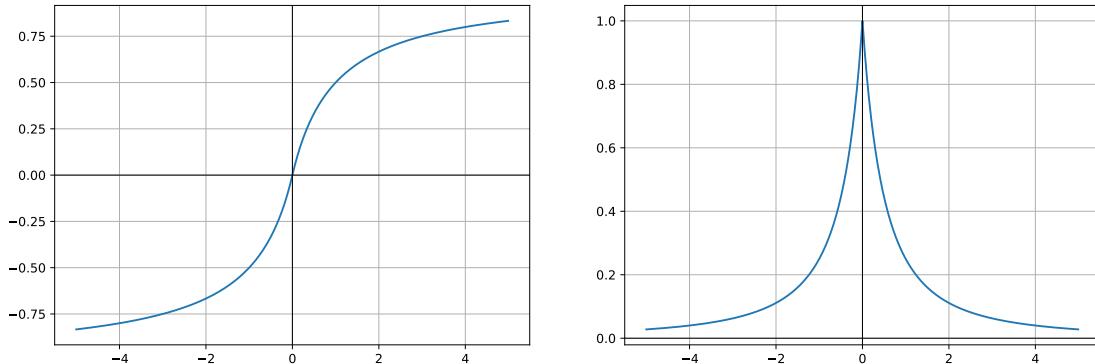
Slika 4.17: Ispravljeni tanh i njegova derivacija

$$f(x) = \begin{cases} \tanh(x), & \text{ako } x > 0 \\ 0, & \text{inače} \end{cases} \quad f'(x) = \begin{cases} 1 - \tanh^2(x), & \text{ako } x > 0 \\ 0, & \text{inače} \end{cases} \quad (4.20)$$

$$= \text{ReLU}(\tanh(x))$$

Ispravljeni tangens hiperbolni je strogo pozitivan, sadrži oštru nelinearnost u 0 i na pozitivnoj domeni prelazi u zasićenje. Funkcija je uvedena radi ispitivanja utječe li magnituda izlaza na klasifikaciju s Dropout-om ili je dovoljna binarna informacija je li neuron upaljen ili ugašen. Pri inferenciji mreže izlazi prolaze kroz 0-1 funkciju te postaju binarni. Usporednom funkcije s *ReLU* na potpuno povezanom modelu pokazuje se da su performanse malo lošije. Interpretacija autora je da modeli ne ostvaruju značajan doprinos od neograničenog linearog dijela *ReLU* funkcije. [25]

4.1.18. Softsign



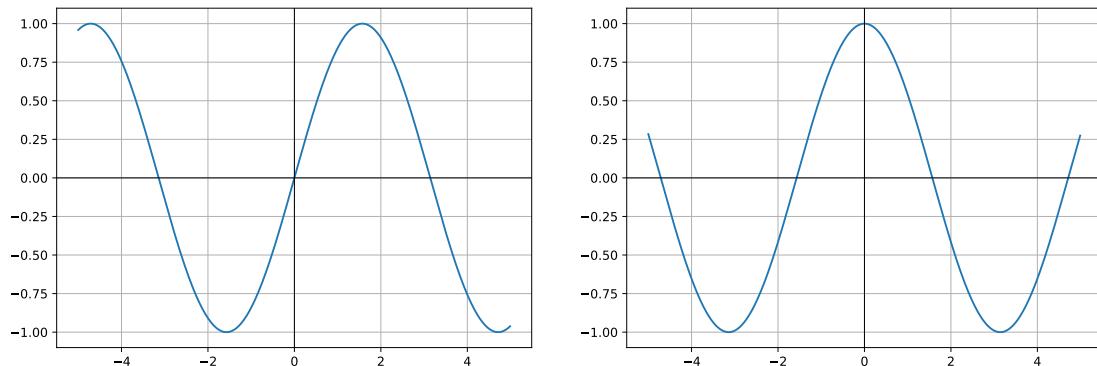
Slika 4.18: Softsign i njegova derivacija

$$f(x) = \frac{x}{1 + |x|} \quad f'(x) = \frac{1 + |x| - x \cdot \text{sign}(x)}{(1 + |x|)^2} \quad (4.21)$$

Funkcija dijeli sličnost s tangensom hiperbolnim (poglavlje 4.1.14), no za razliku od eksponencijalno konvergirajuće funkcije *tanh*, *softsign* konvergira kvadratno. Za razliku od *tanh* čiji slojevi sekvencijalno prelaze u zasićenje, svi *softsign* slojevi sporije prelaze u zasićenje i to rade zajednički. Aktivacije *tanh* gomilaju se u ekstremima (u zasićenju) i u sredini, dok kod *softsign* osim sredine većina aktivacija djeluje na zglobovima funkcije u kojima je nelinearnost najveća, a gradjenti i dalje prolaze. [12]

Funkcija je originalno predstavljena u radu Bergstra et al. [2] i testirana s nekoliko eksperimenata u području računalnog vida. No, taj rad je u vremenu pisanja ovog rada bio nedostupan autorima. Stoga je ostavljena referenca za znatiželjne čitatelje.

4.1.19. Sinus (\sin)



Slika 4.19: Sinusoida i njegova derivacija

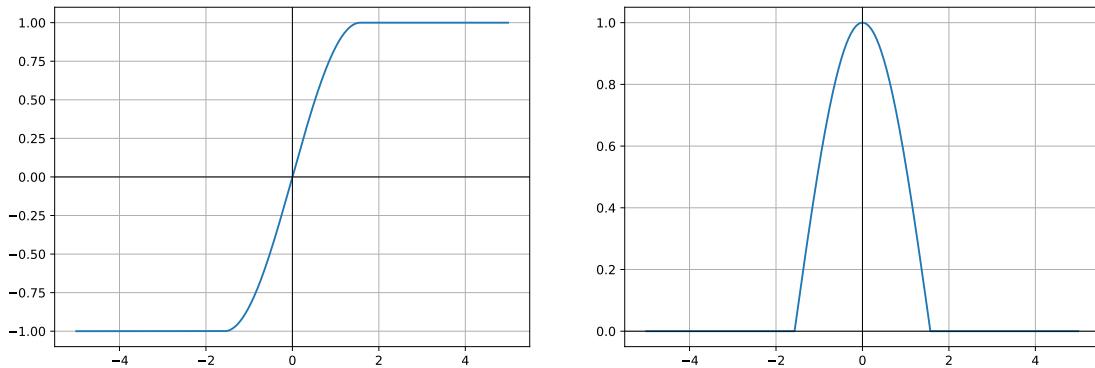
$$f(x) = \sin(x) \quad f'(x) = \cos(x) \quad (4.22)$$

Sinusoida se često pojavljuje u radovima pretrage aktivacijskih funkcija [1] pa tako i u ovom radu te pokazuje obećavajuće rezultate. U radu Parascandolo et al. [35] autori predstavljaju problematiku učenja sa sinusoidom na jednostavnom zadatku aproksimacije sinusoide. Problem stvaraju brojni lokalni optimumi na koje je izrazito osjetljiv gradijentni spust. Problem ublažava učenje stohastičkim gradijentnim spustom koji zaglađuje valovitost funkcije gubitka. Dodatno, autori pokazuju da neuronska mreža zapravo ne ovisi snažno o periodičnosti funkcije tako da su rezultate usporedili s ograničenom sinusoidom (4.23).

Po uzoru na slične funkcije kao što je sigmoida (slika 4.12) i tangens hiperbolni (slika 4.14), sinusoida najveću vrijednost gradijenta daje upravo kada je aktivacija jednaka 0 (u ograničenom intervalu).

4.1.20. Ograničeni sinus (TrSin)

(engl. *Truncated sine*)

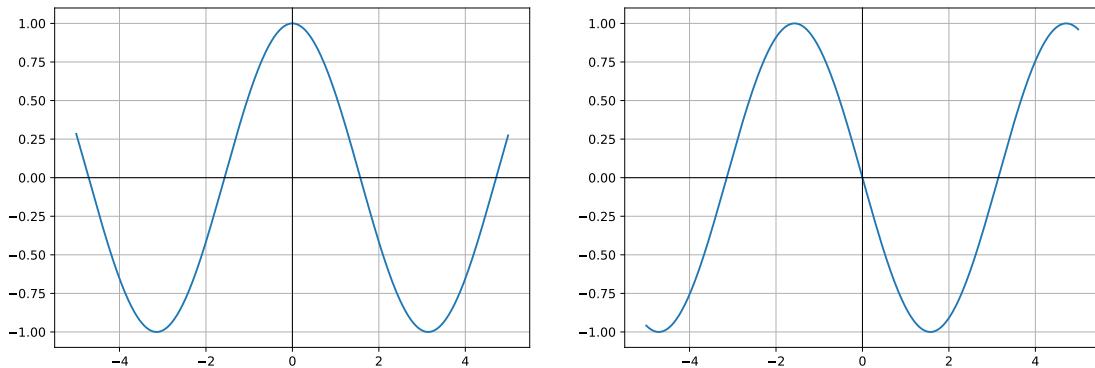


Slika 4.20: Ograničeni sinus i njegova derivacija

$$f(x) = \begin{cases} 0, & x < \frac{\pi}{2} \\ \sin(x), & |x| \leq \frac{\pi}{2} \\ 1, & \text{inače} \end{cases} \quad f'(x) = \begin{cases} \cos(x), & |x| \leq \frac{\pi}{2} \\ 0, & \text{inače} \end{cases} \quad (4.23)$$

Ograničeni sinus korišten je u radu Parascandolo et al. [35] 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.

4.1.21. Kosinus (cos)



Slika 4.21: Kosinus i njegova derivacija

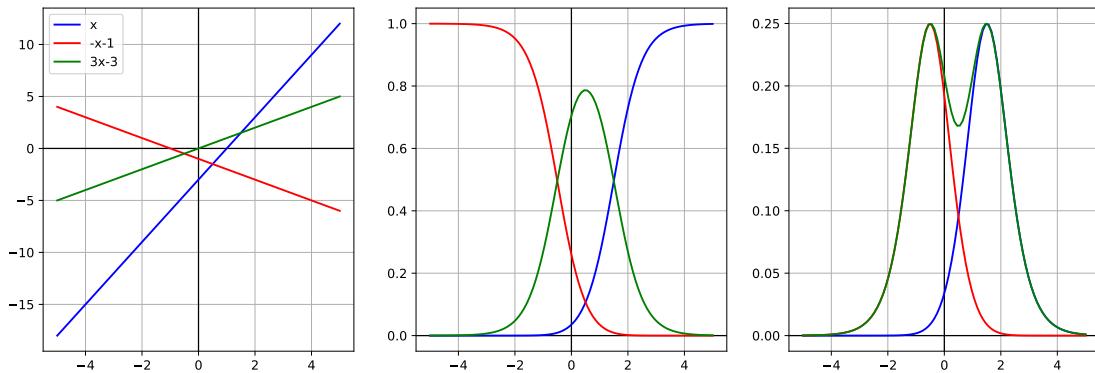
$$f(x) = \cos(x) \quad f'(x) = -\sin(x) \quad (4.24)$$

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, može se promatrati kao zasebna aktivacijska funkcija. Derivacija kosinusa se poprilično razlikuje od ostalih aktivacijskih funkcija. Unatoč tome, daje vrlo kompetitivne rezultate (poglavlje 7).

$$\cos(x) = \sin(x + \frac{\pi}{2}) \approx \sin(x + 1.571) \quad (4.25)$$

4.1.22. Softmax



Slika 4.22: Tri različita ulazna pravca te odziv funkcije *Softmax* i njene derivacije na pravce

$$f(\vec{x}) = \frac{e^{\vec{x}}}{\sum_i e^{\vec{x}_i}} \quad f'(\vec{x}) = f(\vec{x}) \cdot (1 - f(\vec{x})) \quad (4.26)$$

Softmax se tipično koristi kao izlazni sloj klasifikacijske mreže. Funkcija nije skalarna već na ulazu zahtjeva vektor. Eksponentom ulaza ističu se veće vrijednosti ulaza, a dijeljenjem sa sumom normaliziraju izlazi (suma izlaza je jednaka 1). Time se izlaz mreže može smatrati vjerojatnosnom distribucijom što je izrazito korisno za zadatke klasifikacije. Osim što se izlaza može zaključiti koja je klasa najvjerojatnija, funkcija definira i koliko je mreža sigurna u svoju predikciju. Njenim korištenjem drastično se pojednostavljuje derivacija funkcije gubitka za klasifikaciju (3.16) što ubrzava učenje mreže.

5. Optimizacija aktivacijske funkcije genetskim programiranjem

5.1. Genetsko programiranje

Genetsko programiranje je područje evolucijskog računarstva koje se bavi algoritmima pretrage hijerarhijskih struktura. Razlikuje se od susjednih područja po tome što je genotip izvršivi program i dobrota jedinke se dobiva njenim izvršavanjem. Genotip tipično poprima stablastu strukturu te u pravilu nema zadalu veličinu, no može se ograničiti. Dijelovi genotipa definirani su jezikom, skupom funkcija i listova koje definiraju prostor pretraživanja. U ovom radu se elementi jezika nazivaju čvorovima. Jezik mora biti zatvoren (izlaz svakog čvora je kompatibilan s ulazom svakog drugog čvora) i dovoljan (rješenje je moguće predstaviti isključivo koristeći elemente jezika). [10]

Evolucijski algoritmi su intrinzično stohastični te omogućuju globalno i lokalno pretraživanje prostora, za razliku od gradijentnog spusta koji lokalno pretražuje prostor. Dok se lokalnim pretraživanjem pretražuje usko susjedstvo oko točke u prostoru, globalno pretraživanje pretražuje točke po čitavoj domeni unutar dosega. To svojstvo osigurava otpornost evolucijskih algoritama na prepreke koje ograničavaju gradijentni spust (poglavlje 3.1.2). Opseg pretrage određen je jezikom te ograničenjima pri inicijalizaciji populacije i operatorima pretrage. Treba razlikovati genotipski i fenotipski prostor. Genotipski prostor definiran je jezikom i najvećom dubinom stabla, dok je fenotipski prostor definiran dekoderom koji genotip pretvara u primjenjivo rješenje. Iako ograničenja u genotipu ograničavaju fenotip, mogu imati nepredvidiv utjecaj.

5.1.1. Građa

Kao i ostali evolucijski algoritmi, algoritam genetskog programiranja sastoji se od više elemenata koji definiraju način pretrage. **Inicijalizator populacije** služi stvaranju po-

četnih točaka iz kojih kreće pretraga i poželjno je da stvara što raznolikija rješenja (globalna pretraga). **Operator odabira** iz populacije uzima po jedan par jedinki za roditelje i jednu ili dvije jedinke za zamjenu. Pri RouletteWheel odabiru vjerojatnosti odabira jedinki proporcionalna je njihovoj dobroti. Pri turnirskom odabiru nasumično se odabire n jedinki, dvije najbolje postaju roditeljima te ona najlošija biva odabrana za zamjenu. Operatori odabira naginju zadržavanju dobrih genotipa u populaciji i time osiguravaju konvergenciju algoritma (lokalna pretraga). **Operator križanja** iz genotipa odabranih roditelja izgradi jedno ili dva djeteta koja će biti dio nove populacije. Operatori križanja miješaju genotipove jedinki zadržavajući svojstva dobrih i time pomazu konvergenciji algoritma (lokalna pretraga). **Operator mutacije** nad stvorenim djetetom unosi nasumičnu izmjenu i time osigurava unos novih informacija u populaciji (globalna pretraga). Često se definira po nekoliko različitih operatora križanja i mutacije koji različito djeluju na pretragu te se time upravlja globalnošću pretrage. Pristup izgradnji nove populacije na temelju stare utječe na snagu konvergencije algoritma. **Generacijski pristup** će u svakoj iteraciji čitavu populaciju zamijeniti djecom. Pri tome često se koristi **elitizam** kojim se garantira očuvanje najbolje jedinke iz stare u novu populaciju. **Eliminacijski pristup** definira postotak populacije koji će biti zamijenjen djecom.

```

Input: funkcija  $f(x)$ 
Input: kriterij zaustavljanja  $K$ 
Input: veličina populacije  $N$ 
InicijalizirajPopulaciju(populacija, N)
while  $K$  nije zadovoljen do
     $x^* \leftarrow OdaberijNajbolju(x, populacija)$ 
    definiraj novu populaciju:  $populacija'$ 
    for  $N$  puta do
         $(p_1, p_2) \leftarrow OdaberijRoditelje(populacija)$ 
         $c \leftarrow OperatorKrijanja(p_1, p_2)$ 
         $OperatorMutacije(c)$ 
         $DodajPopulaciji(c, populacija')$ 
    end
     $populacija \leftarrow populacija'$ 
end
Result: pronađena optimalna jedinka  $x^*$ 

```

Algorithm 5: Tipičan generacijski evolucijski algoritam

Evolucijski algoritmi podržavaju nekoliko glavnih hiperparametara. **Veličina populacije** definira broj istovremenih točaka u prostoru te o njoj ovisi vjerovatnost pronalaska optima. Operatori mutacije unose izmjene u genomu i njihova primjena se regulira **vjerovatnošću primjene**. **Kriteriji zaustavljanja** određuju završetak pretrage i mogu ovisiti o kvaliteti rješenja ili raspoloživim resursima. **Broj iteracija** analogan je onome u gradijentnom spustu (poglavlje 3.1.2) i često se koristi kao gornja granica za vremenski trošak izvodenja. **Broj evaluacija** je koristan kada je poznato trajanje pojedine evaluacije želi se sukladno raspoloživom vremenu odrediti veličina populacije. **Raspoloživo vrijeme** se koristi kao i broj iteracija, ali u situacijama kada nije poznato trajanje pojedine iteracije algoritma. **Kvaliteta rješenja** je koristan kriterij zaustavljanja ako postoji definirana granica zadovoljavajuće kvalitete. Razne inačice evolucijskih algoritama mogu uvesti brojne dodatne hiperparametre.

5.1.2. Neuroevolucija genetskim programiranjem

Proizvoljnu neuronsku mrežu moguće je predstaviti stablom, gdje su čvorovi mreže ulazne i aktivacijske funkcije neurona te dodatne operacije (normalizacija grupe, dropout i ostali). Takav pristup korišten je u Wilson et al. [46] algoritmom kartezijskog

genetskog programiranja. Metoda NEAT dijeli graf na neurone i težine istovremeno pretražuje prostor arhitektura i težina [40]. Postoje i metode indirektne evolucije arhitekture neuronskih mreža čiji genotip je potrebno dekodirati. Takve metode smanjuju prostor pretraživanja i time efikasno istražuju vrlo složene arhitekture [15].

5.2. Optimizacija aktivacijskih funkcija

U literaturi postoje brojni radovi na temu optimizacija aktivacijskih funkcija, neki od kojih se služe genetskim programiranjem. U nastavku su detaljnije objašnjeni radovi koje je autor pronašao na temu optimizacije aktivacijskih funkcija.

U radu Ramachandran et al. [36] autori smanjuju prostor pretraživanja definirajući strukturu funkcija koje se pretražuju u obliku bloka. Blokove definiraju kompozicijom binarne i dviju unarnih funkcija te istražuju rekurzivna proširenja istog bloka (umjesto jedne unarne funkcije ponavlja se blok). Rekurzivna mreža predviđa funkciju na idućem mjestu u bloku temeljem prethodno odabrane funkcije. Izgrađeni blok koristi se kao aktivacijska funkcija dubokih mreža te se postignuta točnost uzima kao vrijednost nagrade za optimizaciju potpornim učenjem. Rezultat pretrage je funkcija *Swish* (poglavlje 4.1.8) koja konzistentno nadjačava *ReLU* (poglavlje 4.1.2) i ostale popularne funkcije.

U radu Basirat i Roth [1] autori koriste hibridni genetski algoritam kako bi evoluirali funkciju različito definiranu na pozitivnoj i negativnoj domeni. Dijelovi funkcije su predstavljeni stablima i križaju ih posebnim operatorima koji odvojeno mijenjaju pozitivnu i negativnu stranu. Skup čvorova su osnovne aritmetičke operacije, a listovi su popularne aktivacijske funkcije bez konstanti. Autori predstavljaju i nove aktivacijske funkcije (poglavlja 4.1.9, 4.1.10) koje su ručno izgradili s ciljem kombiniranja dobrih svojstava manjih funkcija. Na tri podatkovna skupa pokazuju da su njihove funkcije najbolje.

Rad Mayer i Schwaiger [29] predstavlja aktivacijsku funkciju definiranu kubnim splajnom. Koristi se genetski algoritam za pronašak točaka kojima se interpolira aktivacijska funkcija te pronašak arhitekture čitave mreže. Rezultati pokazuju da pronađene funkcije ostvaruju poboljšanje nad sigmoidom na zadatcima klasifikacije.

5.3. Genetsko programiranje s tabu listom

Klasičan algoritam genetskog programiranja često dovodi do stvaranja identičnih jedinki u populaciji što loše utječe na globalnost pretrage. Stoga se u ovom radu unosi primjena tabu liste u proces algoritma. Nakon stvaranja djeteta klasičnim koracima (križanje i mutacija) dodaje se dodatan korak u kome se dijete mutira dok ne postane jedinstveno. Kako algoritam ne bi zapeo definira se gornja granica dozvoljenog broja mutacija. Kada je stvoreno jedinstveno dijete, ono se dodaje u populaciju i zabilježi na kraj tabu liste.

Input: funkcija $f(x)$

Input: kriterij zaustavljanja K

Input: veličina populacije N

Input: veličina tabu liste T

InicijalizirajPopulaciju(populacija, N)

$tabu \leftarrow StvoriPraznuTabuListu(T)$

while K nije zadovoljen **do**

$x^* \leftarrow OdaberijNajbolju(x, populacija)$

definiraj novu populaciju: $populacija'$

for N puta **do**

$(p_1, p_2) \leftarrow OdaberijRoditelje(populacija)$

$c \leftarrow OperatorKrijanja(p_1, p_2)$

$OperatorMutacije(c)$

while $x \in tabu$ **do**

$| OperatorMutacije(c)$

end

$DodajNaKraj(c, tabu)$

if $|tabu| > T$ **then**

$| UkloniPrvog(tabu)$

end

$DodajPopulacijsi(c, populacija')$

end

$populacija \leftarrow populacija'$

end

Result: pronađena optimalna jedinka x^*

Algorithm 6: Genetsko programiranje s tabu listom

Veličina tabu liste definira koliko se često smije pojaviti već postojeća jedinka u populaciji. U ovom radu se istražuje utjecaj na razini broja iteracija. Na primjer, ako je populacija veličine 10 jedinki tada će veličina tabu liste za 2 iteracije biti veličine 20. Za vrijednost 0 algoritam se ponaša identično klasičnom algoritmu genetskog programiranja.

Zbog ponovljenog mutiranja jedinke treba paziti na balans između mutacija koje povećavaju ili smanjuju dubinu stabla. U suprotnom će jedinke vrlo brzo postati vrlo velike što je nepoželjno za pretraživanje aktivacijskih funkcija [36] ili će ostati suviše male što ograničava pretraživanje šireg prostora.

5.3.1. 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	relu	(4.3)	1
const	$c \in \mathbb{R}$	0	lrelu	(4.4)	1
+	$x + y$	2	threlu	(4.5)	1
-	$x - y$	2	sotplus	(4.6)	1
*	$x \cdot y$	2	elu	(4.8)	1
/	$\frac{x}{y+10^{-12}}$	2	selu	(4.9)	1
min	$\min(x, y)$	2	swish	(4.10)	1
max	$\max(x, y)$	2	sigmoid	(4.15)	1
abs	$ x $	1	hsigmoid	(4.16)	1
sin	$\sin(x)$	1	tanh	(4.17)	1
cos	$\cos(x)$	1	htanh	(4.18)	1
tan	$\tan(x)$	1	rattanh	(4.19)	1
exp	e^x	1	rectanh	(4.20)	1
log	$\log_e(x)$	1	softmax	(4.26)	1
pow2	x^2	1	softsign	(4.21)	1
pow3	x^3	1	trsin	(4.23)	1
pow	x^y	2	gauss	(??)	1

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

5.3.2. 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 odabranou. Ne obavlja se biranje djece prema dobroti zbog skupocjenosti postupka evaluacije, no preporučuje se zbog kvalitetnije pretrage.

Zamijeni 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 miješanje 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.

5.3.3. 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 (poglavlje 5.3.1).

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 dijete postaje stari korijen. Ako je odabrani čvor većeg stupnja prvo dijete 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 zadanoj 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 operatorm 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 izmjene 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 uvjet je kada se generira stablo dubine 1, što utjecaj operatora čini identičnim operatoru 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 djetetom. 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.

Zamjeni 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 dijete 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 dijeljenja, 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 [44] korištena je za učitavanje podatkovnih skupova, pretprocesiranje, izgradnju i optimizaciju neuronskih mreža te podršku izvođenja na grafičkim karticama s podrškom CUDA biblioteke. Obrada podataka i iscrtavanje grafova funkcija i rezultata implementirano je Python skriptama i Jupyter bilježnicama u programskom jeziku Python verzije 3 korištenjem raznih biblioteka (numpy, sk-learn, matplotlib i ostalim). Slike neurona i gradijentnog spusta iscrtane su alatom *draw.io*.

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 *JUnit* 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* [21] 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 operatora 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 koja se zamjenjuje 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.

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.

Evaluator

S obzirom da je postupak treniranja i validacije često vrlo zahtjevan postupak u evaluator je ugrađena memorija evaluiranih aktivacijskih funkcija. Jedinka se prvo serializira u tekstualni oblik i uspoređuje s memorijom. Ako je pronađena, dohvata se njena vrijednost i vraća umjesto ponovljenog treniranja. Postupak je valjan samo ako su eksperimenti ponovljivi za zadane parametre što u ovom projektu vrijedi.

6.2.2. Neuronska mreža

Neuronska mreža definirana je razredom *CommonModel* te služi za transparentnu izgradnju mreže iz zadanih parametara, čuvanje instance mreže i povijesti vrijednosti gubitka nakon učenja.

Zajednički razred za optimizaciju mreža

Za potrebe transparentnog učenja, validacije i testiranja neuronskih mreža te ponovljivost eksperimenata i memoriske uštede definiran je razred *TrainProcedureDL4J*. Pri inicijalizaciji razred učitava skupove za treniranje i testiranje. Ako je tako definirano parametrima, značajke skupova se normaliziraju prema skupu za učenje te se, po potrebi, iz izmiješanog skupa za učenje gradi skup za validaciju. Miješanje skupa za učenje je konzistentno u svim eksperimentima. Razred nasljeđuje sučelje s istim metodama kako bi se omogućila drugačija implementacija razreda. Jedan primjer je mogućnost pozivanja skripti koje implementiraju te postupke. Za komunikaciju s Python skriptama izgrađen je pomoći razred *PythonBridge*.

6.2.3. Pomoći mehanizmi projekta

Standardizirana pohrana rezultata

Zbog potrebe standardizirane i transparentne pohrane rezultata i parametara eksperimenata definiran je razred *StorageManager* koji definira zapisivanje podataka i putanje datoteka za različite eksperimente. Opisnik eksperimenta definiran je razredom *Context*, a sadrži naziv podatkovnog skupa i naziv eksperimenta. Metode za pohranu primaju podatke koji se zapisuju i opisnik kojim se izgradi putanja (npr. za Windows sustave: *podatkovni_skup\eksperiment*). Poddirektoriji datoteke grade se automatski, a putanje se grade razdjelnikom ovisnom o operacijskom sustavu (definiran u Javi:

`File.separator`). Standardizirana pohrana rezultata uvelike olakšava kasniju obradu podataka.

Hiperparametri

Hiperparametre programa moguće je zadati konfiguracijskom datotekom. Datoteka je formata ključ-vrijednost, odvojeni regularnim izrazom: "[: \t]+". Kroz datoteku moguće je definirati i više vrijednosti parametara koji će se pretraživati po rešetci (npr. $\{p1, p2, p3\}$). Datotekom je moguće zadati sve parametre postupaka, uključujući sjeme korišteno u brojnim procesima. Ako nije zadan važan parametar program će baciti iznimku.

Hiperparametri za učenje neuronskih mreža su definirani razredom *TrainParams* i njima se služi razred za optimizaciju mreža. Hiperparametri za evoluciju aktivacijskih funkcija definirani su razredom *EvolvingActivationParams* koji naslijeđuje *TrainParams* te kojim se služi izvršni program *EvolvingActivationProgram*.

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 uvjet čekanja *WaitCondition*. Moguće je definirati svoj uvjet (npr. čekanje dok se ne popuni spremnik) ili pričekati dok se svi poslovi ne izvrše, uvjetom koji vraća metoda *getAllFinishedCondition*.

Pri definiranju poslova često je potrebno definirati lokalnu varijablu s modifikatorom *final* koja se ažurira po početku ili završetku posla. Kako se ne bi trebalo definirati polje (što je česta praksa) definiran je pomoćan generički razred *Holder* koji nudi metode za dohvati i postavljanje unutarnjeg objekta. Za potrebe nabranja definiran je razred *Counter*, a za zajedničko pamćenje dva ili tri objekta definirani su generički razredi *Pair* i *Triple*.

Bilježenje napretka algoritma

Pri izvršavanju algoritma učenja ili pretrage predaje se objekt koji implementira sučelje *ILogger*. Objekt nudi metode za selektivno bilježenje informacija o izvođenju programa ili algoritma ovisno o njihovom tipu. Metode selektivnog bilježenja inspirirane su objektom *Log* Java razvojne biblioteke operacijskog sustava Android. Postoji nekoliko izvedbi koje preusmjeravaju ulaze ostalim mehanizmima. Tablica 6.1 prikazuje implementacije i njihovo odredište.

Klasa	Odredište
DevNullLogger	Ignorira ulaze
StdoutLogger	Standardni izlaz
FileLogger	Zapisuje u zadanu datoteku (s kreiranjem poddirektorija)
SlackLogger	Kanal servisa Slack
MultiLogger	Prosljeđuje listi zadanih implementacija

Tablica 6.1: Implementacije za bilježenje

6.2.4. Izvršni programi projekta

U projektu su napisani brojni izvršni programi čije namjene uključuju demonstraciju dodataka, izvršavanje eksperimenata, ponavljanje eksperimenta s dodatnim bilježenjem i slično. U nastavku su opisani programi korišteni za izvršavanje eksperimenata.

Pohlepna pretraga uobičajenih funkcija

Razred ovog programa je *GreedySearchProgram*. Program služi za pohlepnu pretragu opisanu u poglavlju 7.1.1 te za eksperimentiranje kombinacijama hiperparametara. Ne prima ulaze već je predodređen za aktivno mijenjanje koda. Na početku programa se nalaze zajednički hiperparametri te su definirane funkcije i arhitekture čije kombinacije želimo ispitati. Sadrži i mehanizam nastavljanja eksperimenata ako je došlo do prekida. Pod nastavljanjem se misli na nastavak pretraživanja po rešetci, samo učenje mreže nije moguće nastaviti. Na dnu programa nalaze se vrijednosti hiperparametara koje se pretražuju za svaku kombinaciju funkcije i arhitekture.

Izgradnja aktivacijskih funkcija simboličkom regresijom

Razred ovog programa je *EvolvingActivationProgram*. Program služi pokretanju eksperimenata s evolucijom aktivacijske funkcije. Osnovni ulaz u program je putanja

do datoteke s hiperparametrima algoritma. Drugi ulaz je putanja do .jar datoteke tog istog programa i nije nužan. Njime se program prebacuje u procesni način rada, pri čemu se svaki eksperiment pokreće u zasebnom procesu. Ovime se prisiljava oslobođanje memorije prethodnog eksperimenta koja može procuriti iz nepoznatih razloga. Gašenjem glavnog procesa automatski se gasi i proces djete.

Prilikom izvršavanja može doći do velikih zahtjeva za memorijom. Ako nisu postavljene gornje granice memorijskih zahtjeva, doći će do prekida programa. Za povećavanje gornje memorijске granice JVM-a, prilikom pokretanja JVM-u se treba zadati zastavica, npr. $-Xmx4g$ koja postavlja granicu na 4GB. Za ograničavanje memorije koju DL4j zauzima izvan gomile i na grafičkim karticama, treba se zadati zastavica, npr. $-Dorg.bytedeco.javacpp.maxbytes = 3g$ koja postavlja granicu na 3GB.

7. Rezultati

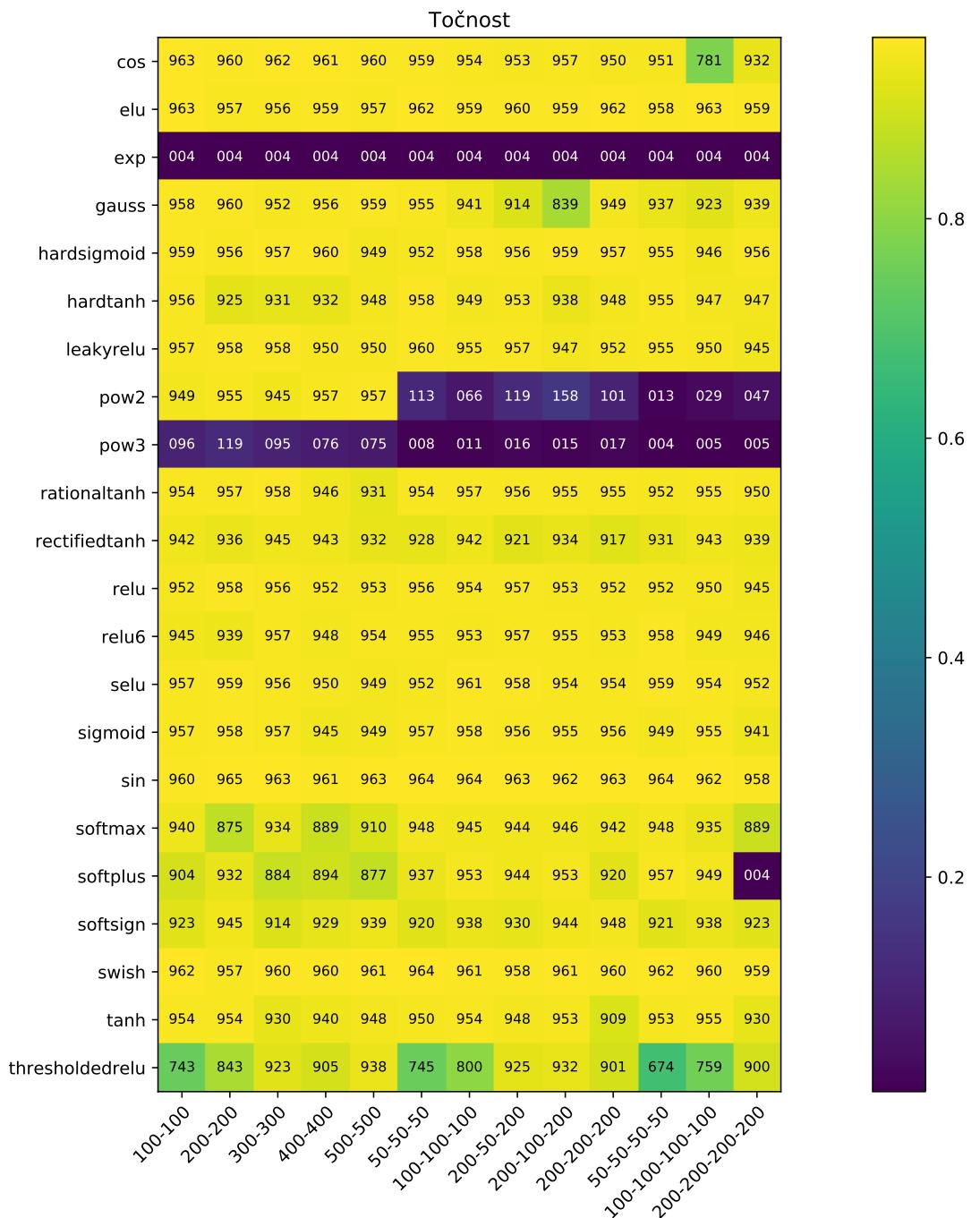
7.1. DPAv4

7.1.1. Usporedba uobičajenih aktivacijskih funkcija

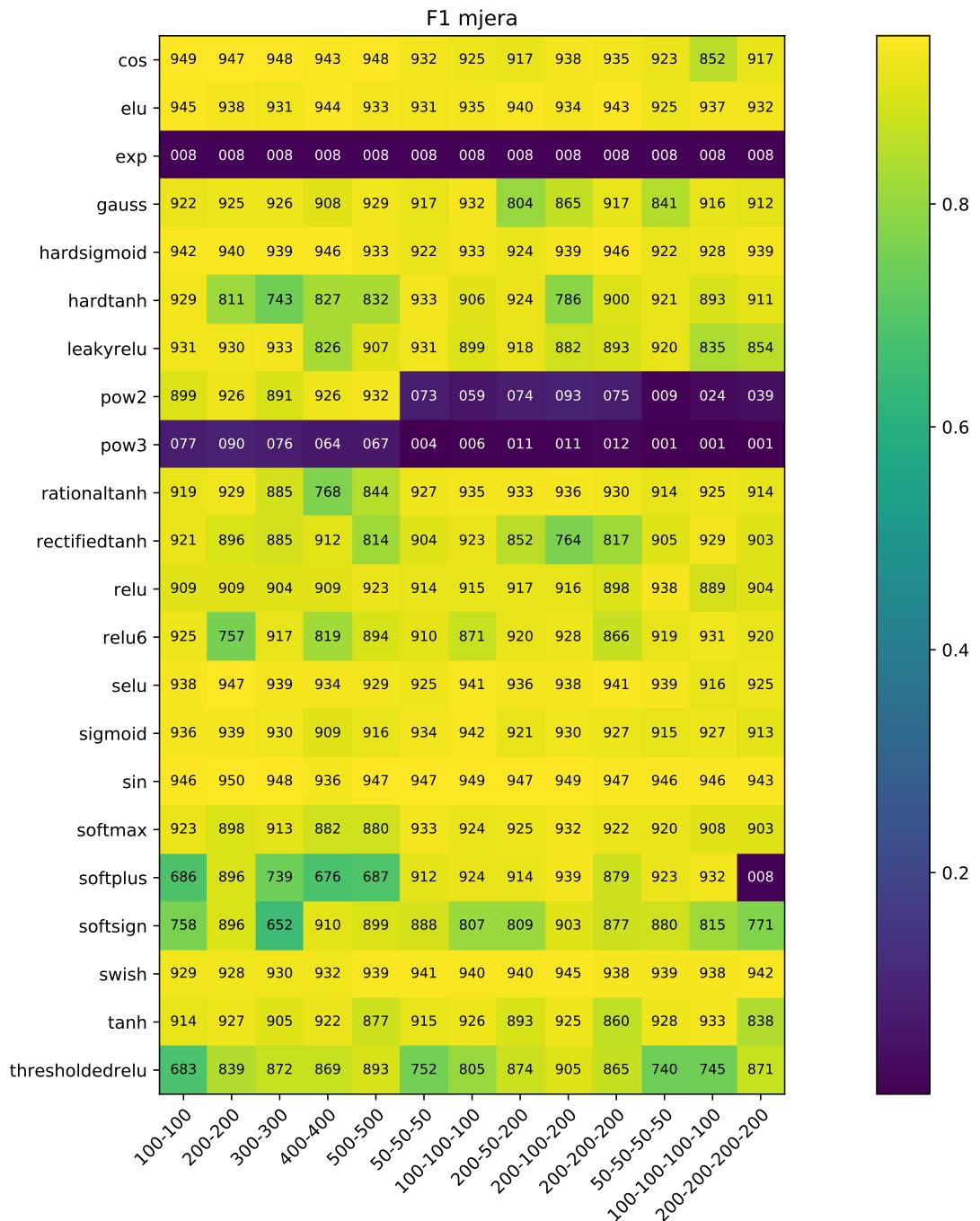
U ovom poglavlju prikazana je usporedba performansi popularnih aktivacijskih funkcija. Ispitivane su kombinacije arhitektura i aktivacijskih funkcija te su priložene postignute mjere točnosti i F1 mjere na skupu za testiranje. Za svaku kombinaciju arhitekture i aktivacijske funkcije provedeno je pretraživanje hiperparametara po rešetci i zabilježeni su najbolji postignuti rezultati. Zajednički i pretraživani hiperparametri navedeni su tablici 7.1. Najbolje kombinacije hiperparametara su navedene u dodatku A.

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	$10^{-3}, 5 \cdot 10^{-4}, 10^{-4}$

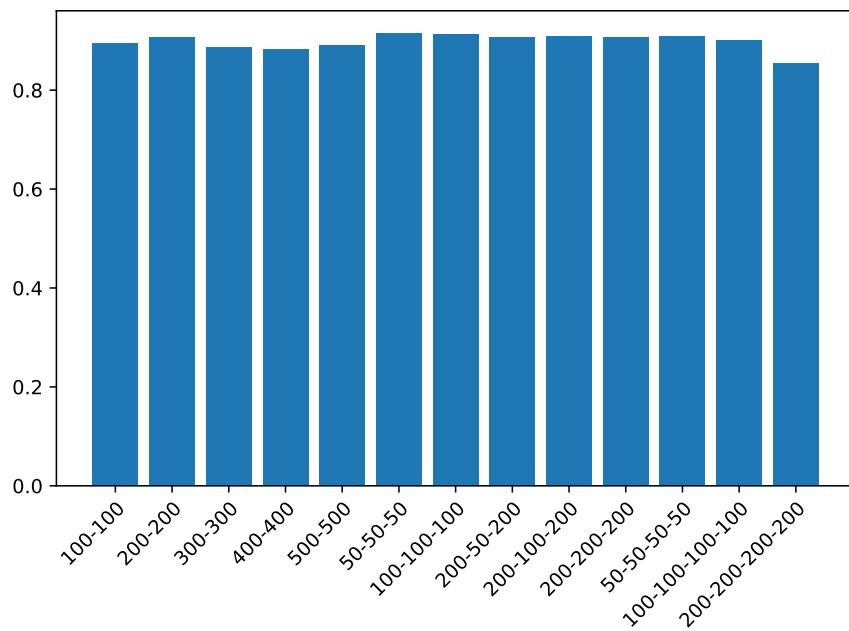
Tablica 7.1: Hiperparametri korišteni pri učenju mreža. Zarezima su odvojene vrijednosti hiperparametara koje su pretraživane po rešetci.



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



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



Slika 7.3: Prosječna F1 mjera po svim arhitekturama. Ignorirani su rezultati funkcija kojima mreža nije uspješno naučena (*exp, pow2, pow3*). Najveću sumu ostvaruje arhitektura [50-50-50] te se upravo ona koristi za dalnjim eksperimentima. Koristi se najčešća kombinacija hiperparametara za tu arhitekturu je pod **indeksom 5** tablice A.1.

7.1.2. Izgradnja aktivacijskih funkcija simboličkom regresijom

TODO: *Tablica*

TODO: *Komentar*

[IMAGE: *boxplot usporedba po tabu veličini*]

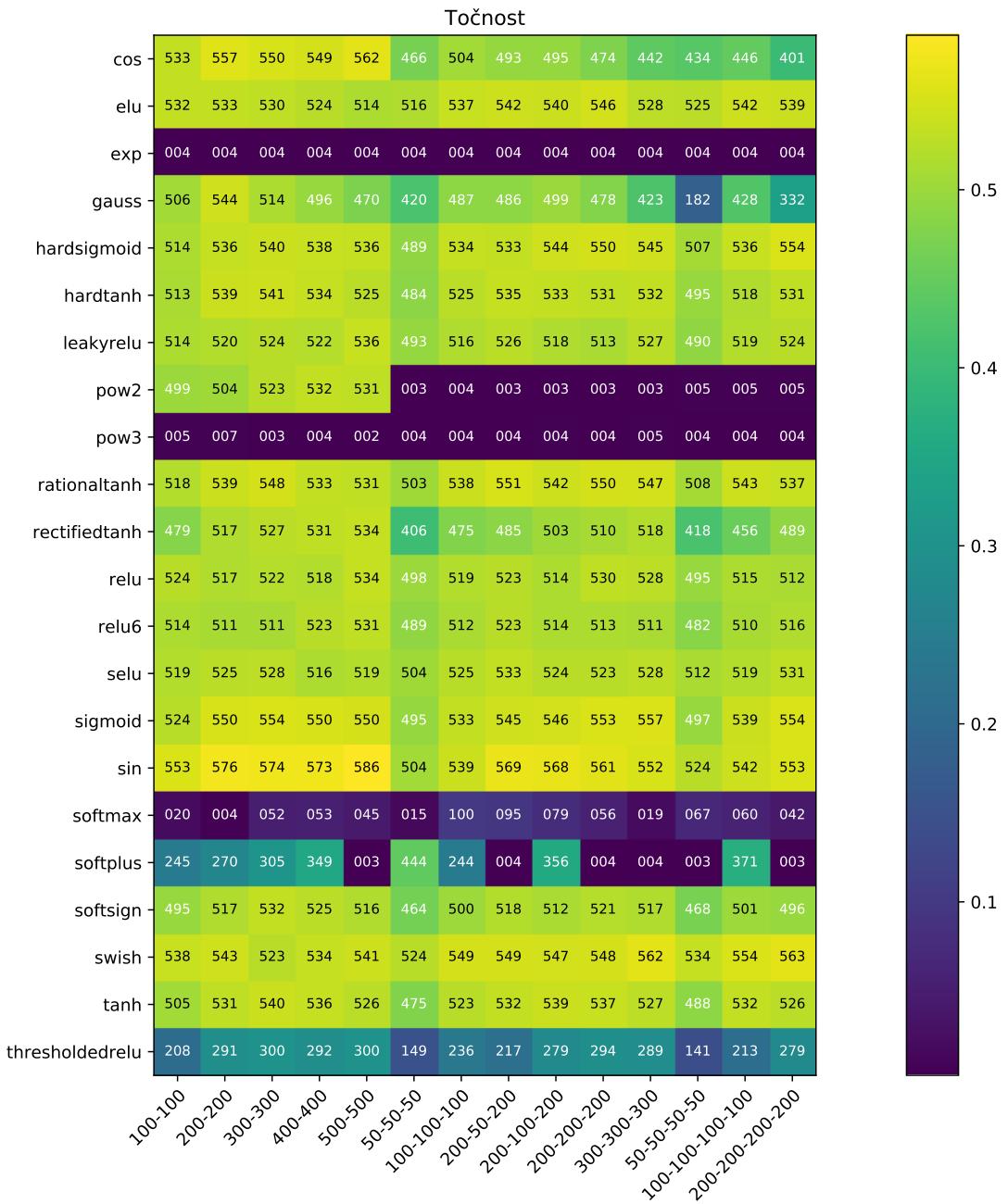
7.1.3. Izgradnja heterogenog rasporeda aktivacijskih funkcija

TODO: *Ne znam hoću li stići i ovo izvrtiti. Za najbolju prethodnu arhitekturu ču pro-naći najbolji raspored uobičajenih fja po slojevima mreže (npr. sin-relu).*

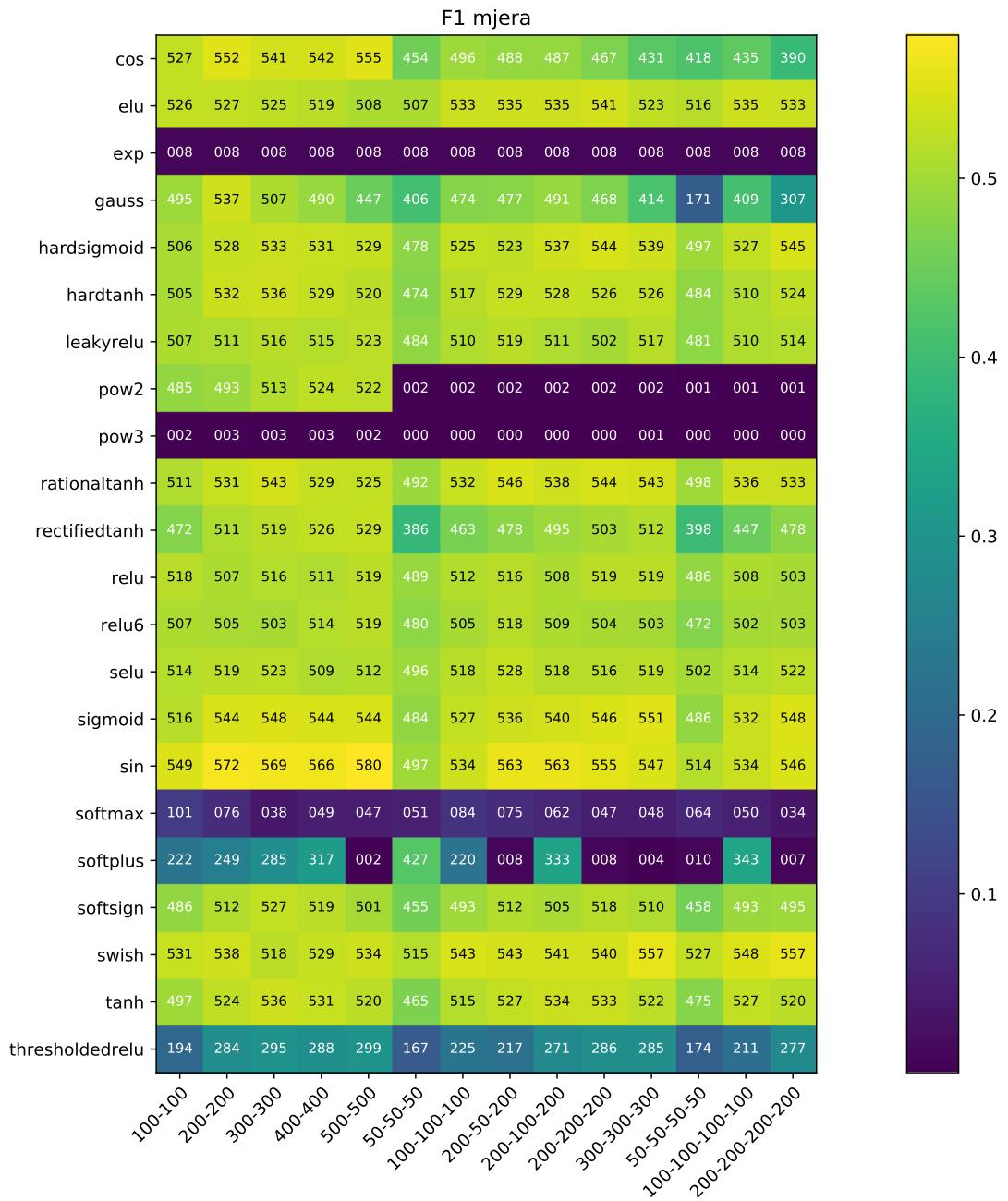
7.2. DPAv2

7.2.1. Usporedba uobičajenih aktivacijskih funkcija

U ovom poglavlju prikazana je usporedba performansi popularnih aktivacijskih funkcija. Ispitivane su kombinacije arhitektura i aktivacijskih funkcija te su priložene postignute mjere točnosti i F1 mjere na skupu za testiranje. Za svaku kombinaciju arhitekture i aktivacijske funkcije provedeno je pretraživanje hiperparametara po rešetci i zabilježeni su najbolji postignuti rezultati. Zajednički i pretraživani hiperparametri identični su eksperimentima za DPAv2 (tablica 7.1). Najbolje kombinacije hiperparametara su navedene u dodatku A.



Slika 7.4: Postignuta točnost aktivacijskih funkcija na različitim arhitekturama. Vrijednosti u čelijama su tri najznačajnije znamenke nakon decimalne točke. Na vrijednosti mjeri 0.5 tekst mijenja boju iz bijele u crnu.



Slika 7.5: Postignuta F1 mjera aktivacijskih funkcija na različitim arhitekturama. Vrijednosti u čelijama su tri najznačajnije znamenke nakon decimalne točke. Na vrijednosti mjeri 0.5 tekst mijenja boju iz bijele u crnu.

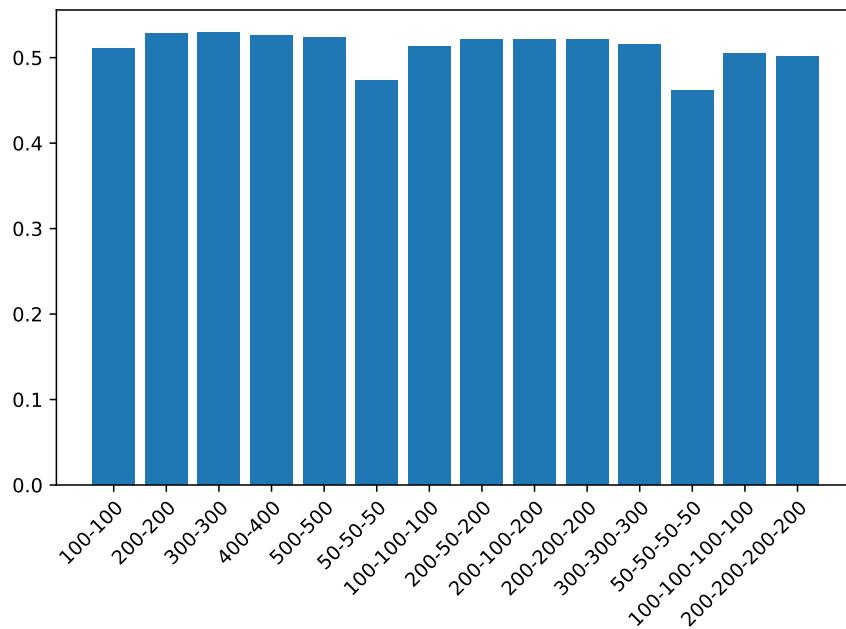
[IMAGE: mjesta aktivacija top 4 funkcije po arh]

TODO: Komentar

[IMAGE: mjesta aktivacija bottom 4 funkcije po arh]

TODO: Komentar

TODO: analiza slučaja pow2



Slika 7.6: Prosječna F1 mjera po svim arhitekturama. Ignorirani su rezultati funkcija kojima mreža nije uspješno naučena (\exp , pow2 , pow3 , softmax , softplus , thresholdedrelu). Najveću sumu ostvaruje arhitektura [300-300] te se upravo ona koristi za dalnjim eksperimentima. Koristi se najčešća kombinacija hiperparametara za tu arhitekturu je pod indeksom 5 tablice ??.

7.2.2. Izgradnja aktivacijskih funkcija simboličkom regresijom

TODO: *Tablica*

TODO: *Komentar*

[IMAGE: *boxplot usporedba po tabu veličini*]

7.2.3. Izgradnja heterogenog rasporeda aktivacijskih funkcija

TODO: *Ne znam hoću li stići i ovo izvrtiti. Za najbolju prethodnu arhitekturu će proći 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

Dropout [38] je tehniku regularizacije neuronskih mreža koja ostvaruje dobre rezultate na vrlo dubokim i širokim modelima koji su skloni pretreniranju i koadaptaciji neurona. Radi tako da za vrijeme učenja mreže proporcionalno zadanoj vjerojatnosti nasumično isključuje neurone u oba smjera. Tehnika efikasno sprječava koadaptaciju neurona širokih slojeva jer je vjerojatnost da će dva neurona koji su skloni koadaptaciji biti identično naučena vrlo mala. Isključivani neuroni će ipak učiti jer vjerojatnost da će neuron ostati isključen kroz više iteracija obrnuto proporcionalna broju iteracija.

Negativna posljedica Dropout-a je duže vrijeme učenja jer je potrebno više iteracija da svaki neuron dobije dovoljan broj efektivnih iteracija učenja. Ta činjenica nije korisna za postupke neuroevolucije kojima trošak evaluacije definira traktabilnost postupka. U sklopu projekta, provedeni su i preliminarni eksperimenti koji su dokazali spomenutu neučinkovitost.

Tensorflow Java API

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

9. Buduća istraživanja

TODO: *Primjena CNN na sirovim vremenskim uzorcima po uzoru na onaj rad*

TODO: *Ispitivanje učinkovitosti korištene optimizacije na ostalim zadatcima*

TODO: *Operatori različitog utjecaja i uzorkovanje s rastućim pragom osjetljivosti na izmijene populacije. Operatori imaju definiranu snagu, kako vrijeme ide sve manje se biraju (putujuća sigmoide) snažni operatori -> konvergencija kao sim. kaljenje*

TODO: *Paralelna evolucija arhitekture i aktivacijskih fja [41]*

10. Zaključak

TODO: *Radi/Ne radi*

TODO: *Pronađene zanimljivosti*

TODO: *Pouka za doma*

LITERATURA

- [1] Mina Basirat i Peter M. Roth. The quest for the golden activation function. *CoRR*, abs/1808.00783, 2018.
- [2] James Bergstra, Guillaume Desjardins, Pascal Lamblin, i Yoshua Bengio. Quadratic polynomials learn better image features. Technical report, 2009.
- [3] Djork-Arné Clevert, Thomas Unterthiner, i Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *CoRR*, abs/1511.07289, 2016.
- [4] R. Collobert. *Large Scale Machine Learning*. Doktorska disertacija, Université Paris VI, 2004.
- [5] Peter Dayan i L. F. Abbott. *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. The MIT Press, 2005. ISBN 0262541858.
- [6] 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.
- [7] Włodzisław Duch i Norbert Jankowski. Taxonomy of neural transfer functions. U *IJCNN*, 2000.
- [8] Włodzisław Duch i Norbert Jankowski. Transfer functions: hidden possibilities for better neural networks. U *ESANN*, 2001.
- [9] R. C. Eberhart. Standardization of neural network terminology. *IEEE Transactions on Neural Networks*, 1(2):244–245, June 1990. ISSN 1045-9227. doi: 10.1109/72.80238.

- [10] Russell C. Eberhart. *Computational Intelligence: Concepts to Implementations*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007. ISBN 1558607595, 9780080553832.
- [11] 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>.
- [12] Xavier Glorot i Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. U Yee Whye Teh i Mike Titterington, urednici, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, svezak 9 od *Proceedings of Machine Learning Research*, stranice 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL <http://proceedings.mlr.press/v9/glorot10a.html>.
- [13] 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>.
- [14] Ian Goodfellow, Yoshua Bengio, i Aaron Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [15] F.C. Gruau. *Neural Network Synthesis Using Cellular Encoding and the Genetic Algorithm*. Université de Lyon 1, 1994. URL <https://books.google.hr/books?id=PTn5rQEACAAJ>.
- [16] Çaglar Gülcöhre, Marcin Moczulski, Misha Denil, i Yoshua Bengio. Noisy activation functions. *CoRR*, abs/1603.00391, 2016. URL <http://arxiv.org/abs/1603.00391>.
- [17] Alexander Hagg, Maximilian Mensing, i Alexander Asteroth. Evolving parsimonious networks by mixing activation functions. U *GECCO*, 2017.

- [18] Hao Zheng, Zhanlei Yang, Wenju Liu, Jizhong Liang, i Yanpeng Li. Improving deep neural networks using softplus units. U *2015 International Joint Conference on Neural Networks (IJCNN)*, stranice 1–4, July 2015. doi: 10.1109/IJCNN.2015.7280459.
- [19] 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.
- [20] 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.
- [21] Domagoj Jakobović, Hrvoje Ban, Vinko Bedek, Igor Bespaljko, Iva Brajer, Luka Donđivić, 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/>.
- [22] Ilya Kalinowski i Vladimir Spitsyn. Compact convolutional neural network cascade for face detection. *CoRR*, abs/1508.01292, 2015. URL <http://arxiv.org/abs/1508.01292>.
- [23] 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>.
- [24] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, i Sepp Hochreiter. Self-normalizing neural networks. U I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, i R. Garnett, urednici, *Advances in Neural Information Processing Systems 30*, stranice 971–980. Curran Associates, Inc., 2017. URL <http://papers.nips.cc/paper/6698-self-normalizing-neural-networks.pdf>.
- [25] Kishore Reddy Konda, Xavier Bouthillier, Roland Memisevic, i Pascal Vincent. Dropout as data augmentation. *CoRR*, abs/1506.08700, 2015.

- [26] Alex Krizhevsky. Convolutional deep belief networks on cifar-10. 2010.
- [27] 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>.
- [28] Andrew L Maas, Awni Y Hannun, i Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models.
- [29] H. A. Mayer i R. Schwaiger. Differentiation of neuron types by evolving activation function templates for artificial neural networks. U *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No.02CH37290)*, svezak 2, stranice 1773–1778 vol.2, May 2002. doi: 10.1109/IJCNN.2002.1007787.
- [30] Roland Memisevic, Kishore Reddy Konda, i David Krueger. Zero-bias autoencoders and the benefits of co-adapting features. *CoRR*, abs/1402.3337, 2015.
- [31] 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>.
- [32] 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>.
- [33] Kevin P Murphy. *Machine learning: a probabilistic perspective*. Cambridge, MA, 2012.
- [34] 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>.

- [35] Giambattista Parascandolo, Heikki Huttunen, i Tuomas Virtanen. Taming the waves: sine as activation function in deep neural networks. 2017. unpublished.
- [36] Prajit Ramachandran, Barret Zoph, i Quoc Le. Searching for activation functions. 2018. URL <https://arxiv.org/pdf/1710.05941.pdf>.
- [37] 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.
- [38] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, i Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014. URL <http://jmlr.org/papers/v15/srivastava14a.html>.
- [39] Rupesh Kumar Srivastava, Klaus Greff, i Jürgen Schmidhuber. Highway networks. *CoRR*, abs/1505.00387, 2015.
- [40] Kenneth O. Stanley i Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evol. Comput.*, 10(2):99–127, Lipanj 2002. ISSN 1063-6560. doi: 10.1162/106365602320169811. URL <http://dx.doi.org/10.1162/106365602320169811>.
- [41] 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>.
- [42] 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.
- [43] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, i Rob Fergus. Intriguing properties of neural networks. U *International Conference on Learning Representations*, 2014. URL <http://arxiv.org/abs/1312.6199>.

- [44] Eclipse Deeplearning4j Development Team. Deeplearning4j: Open-source distributed deep learning for the JVM. URL <https://deeplearning4j.org/>. Apache Software Foundation License 2.0.
- [45] Joseph Turian, James Bergstra, i Yoshua Bengio. Quadratic features and deep architectures for chunking. U *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, NAACL-Short '09, stranice 245–248, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics. URL <http://dl.acm.org/citation.cfm?id=1620853.1620921>.
- [46] Dennis G Wilson, Sylvain Cussat-Blanc, Hervé Luga, i Julian F Miller. Evolving simple programs for playing atari games. U *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '18, stranice 229–236, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5618-3. doi: 10.1145/3205455.3205578. URL <http://doi.acm.org/10.1145/3205455.3205578>.
- [47] Xin Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, Sep. 1999. ISSN 0018-9219. doi: 10.1109/5.784219.
- [48] Marko Čupić, Bojana Dalbelo Bašić, i Marin Golub. *Neizrazito, evolucijsko i neuroračunarstvo*. Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu, (2013-08-12) izdanju, 2013.

Optimizirane aktivacijske funkcije klasifikatora temeljenog na umjetnim neuronским mrežama u domeni implementacijskih napada na kriptografske uređaje

Sažetak

Proučiti postojeće metode u izgradnji aktivacijskih 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 aktivacijske 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. Uslijediti učinkovitost ostvarenih postupaka s postojećim rješenjima iz literature. Radu priložiti izvorne tekstove programa, dobivene rezultate uz potrebna objašnjenja i korištenu literaturu.

Ključne riječi: Ključne riječi, odvojene zarezima.

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

Abstract

Examine existing methods in building activation functions in artificial neural networks. Give special attention to evolutionary algorithms of symbolic regression for constructing the targeted functions. Apply evolved activation 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.

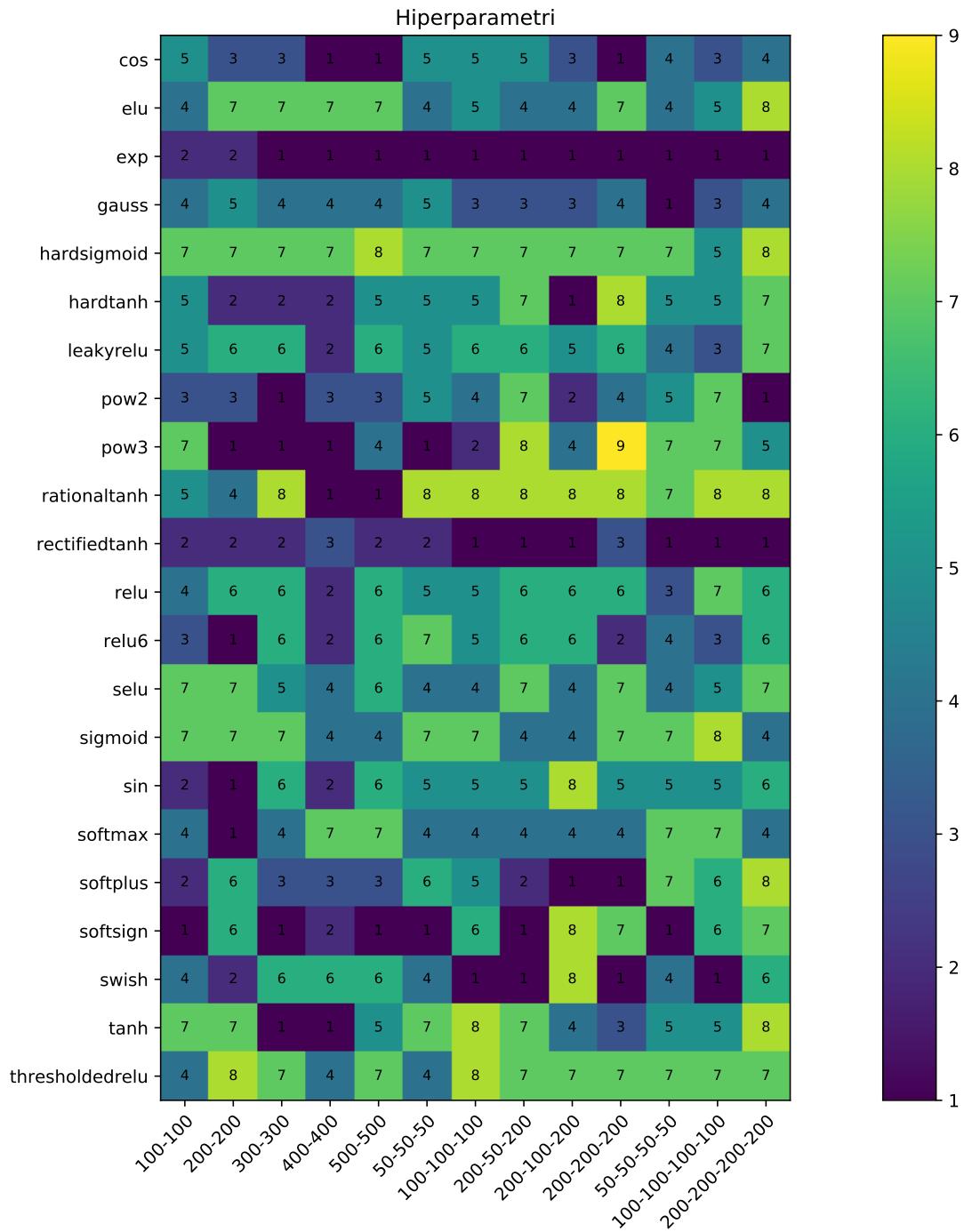
Keywords: Keywords.

Dodatak A

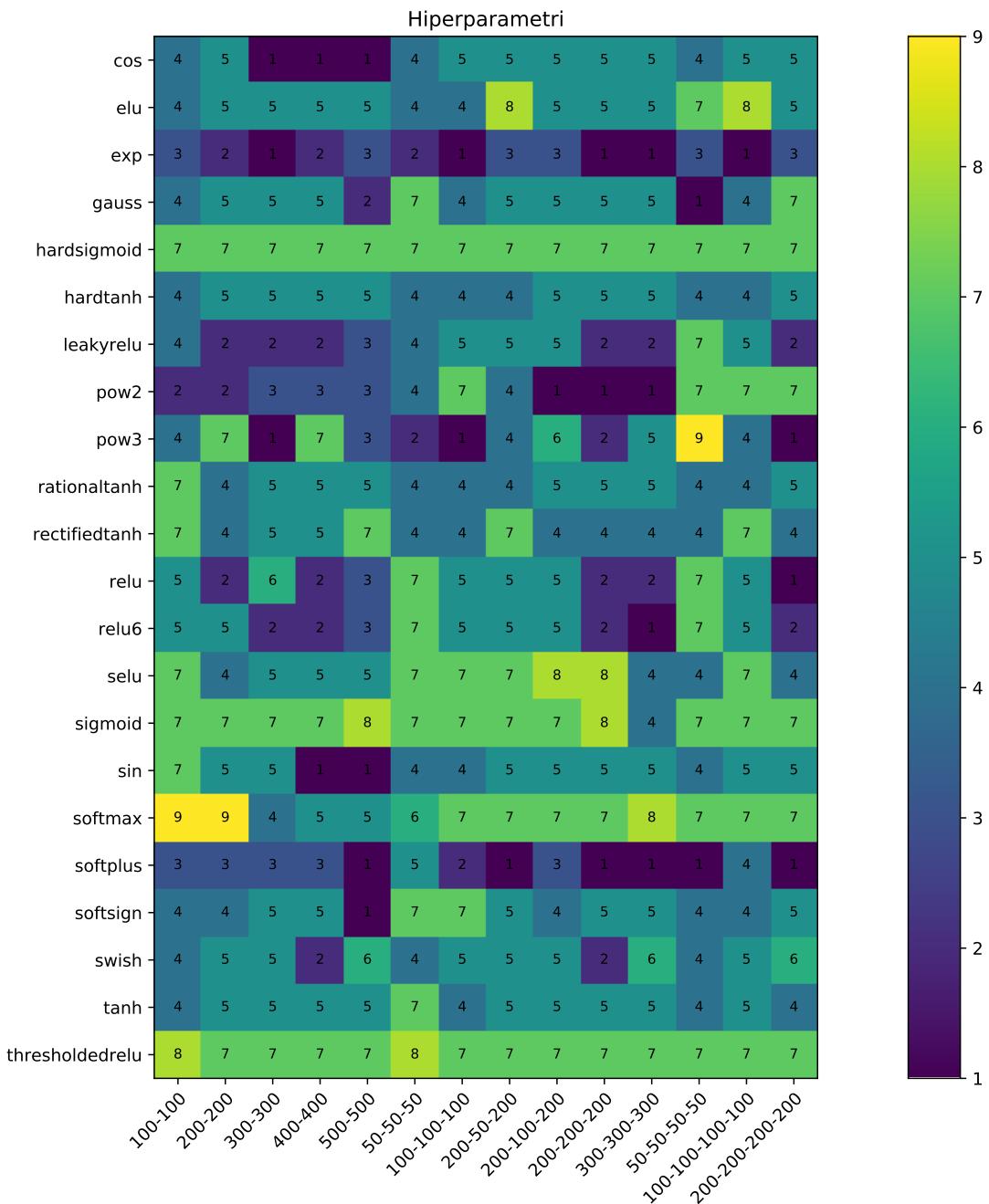
Hiperparametri pretrage po rešetci

Indeks	Koeficijent L2 reg.	Stopa učenja
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 A.1: Indeksi kombinacija hiperparametara korištenih pri pretraživanju po rešetci.



Slika A.1: Skup **DPAv4**: Optimalne kombinacije hiperparametara za aktivacijske funkcije na različitim arhitekturama. Vrijednosti u čelijama označavaju kombinacije hiperparametara navedene u tablici A.1



Slika A.2: Skup **DPAv2**: Optimalne kombinacije hiperparametara za aktivacijske funkcije na različitim arhitekturama. Vrijednosti u čelijama označavaju kombinacije hiperparametara navedene u tablici A.1