SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1485

Primjena stohastičkih algoritama optimizacije u učenju dubokih neuronskih mreža

Luka Križan

Zagreb, lipanj 2017.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA DIPLOMSKI RAD PROFILA

Zagreb, 3. ožujka 2017.

DIPLOMSKI ZADATAK br. 1485

Pristupnik: **Luka Križan (0036470261)**
Studij: Računarstvo
Profil: Računarska znanost

Zadatak: **Primjena stohastičkih algoritama optimizacije u učenju dubokih neuronskih mreža**

Opis zadatka:

Opisati problematiku oblikovanja i učenja neuronskih mreža različitih vrsta slojeva i velikog broja skrivenih slojeva. Oblikovati programski sustav za učenje neuronskih mreža koristeći dostupne programske okvire. Ispitati učinkovitost različitih stohastičkih algoritama optimizacije u učenju neuronskih mreža. Posebnu pažnju posvetiti problemima s velikim brojem primjera za učenje s primjenom u području kriptografije. Radu priložiti izvorne tekstove programa, dobivene rezultate uz potrebna objašnjenja i korištenu literaturu.

Zadatak uručen pristupniku: 10. ožujka 2017.
Rok za predaju rada: 29. lipnja 2017.

Mentor:

Izv. prof. dr. sc. Domagoj Jakobović

Predsjednik odbora za diplomski rad profila:

Prof. dr. sc. Siniša Srbljić

Djelovođa:

Doc. dr. sc. Tomislav Hrkać

SADRŽAJ

1. Uvod

Neuronske mreže kao tehnika u području umjetne inteligencije i strojnog učenja imaju vrlo bogatu povijest te su od svog nastanka kroz prve koncepte iz 1940-ih godina do danas doživjele brojne uspone i padove u popularnosti. Kroz svoju povijest, isti koncept pojavljivao se pod različitim imenima koja odražavaju različite filozofske poglede na koncept — perceptron, umjetne neuronske mreže, duboko učenje. Naziv „duboko učenje“ koristi se posljednjih desetak godina te se njime želi skrenuti pažnja s povijesnog pogleda na neuronske mreže kao modela inspiriranog biološkim mozgom na generalni koncept modela u kojem se model promatra kao kompozicija više slojeva. Iz istog razloga, u literaturi se često pojam „duboka neuronska mreža“ može pronaći pod imenom „duboki model“. Ne postoji definicija kojom bi se točno odredila dubina modela koja bi model karakterizirala kao „duboki“, stoga se dubokim modelima općenito smatraju modeli s više razina kompozicija funkcija u odnosu na tradicionalne tehnike strojnog učenja. U tradicionalnom pristup strojnom učenju koristili su se plitki klasifikatori s ručno konstruiranim značajkama — višeslojni modeli nisu bili popularni unatoč svojem velikom reprezentacijskom potencijalu zbog loše konvergencije i loših performansi. Međutim, u posljednjih desetak godina, zahvaljujući velikom napretku hardvera, matričnim operacijama optimiziranim za izvođenje na grafičkim karticama, pojavom vrlo velikih skupova za učenje te pojavom novih tehnika učenja, duboki modeli u mnogim zadacima postižu znatno bolje rezultate od plitkih modela. Duboko učenje stoga danas ima brojne primjene u računalnom vidu, obradi prirodnog jezika, raspoznavanju govora te mnogim drugim područjima. (Goodfellow et al., 2016)

S ciljem usporedbe različitih algoritama učenja dubokih modela, u sklopu ovog rada implementirana je programska podrška za izgradnju i učenje dubokih modela koristeći programske okvire ECF i TensorFlow.

U drugom poglavlju rada ukratko je objašnjena kompozitna arhitektura dubokih modela te je u trećem poglavlju dan kratak opis korištenih algoritama učenja modela. Arhitektura programske implementacije ukratko je objašnjena u četvrtom poglavlju, a rezultati učenja modela na različitim problemima prikazani su u petom poglavlju.

2. Kompozitna arhitektura modela

Tradicionalni pristupi u strojnom učenju temelje se na učenju modela na prethodno izlučenim značajkama te pri tome performanse modela izrazito ovise o izlučenim značajkama, odnosno, o odabranoj reprezentaciji skupa podataka za učenje. Unatoč tome, mnogi problemi iz područja umjetne inteligencije mogu se riješiti upravo na taj način — izlučivanjem prikladnog skupa značajki te učenjem jednostavnog modela na odabranim značajkama. Na primjer, korisna značajka u problemu identifikacije govornika na temelju zvučnog zapisa može biti procjena veličine glasnica govornika, čime se modelu daje vrlo jasna informacija je li govornik muškarac, žena ili dijete. Međutim, u brojnim zadacima vrlo je teško odrediti koje značajke izlučiti, što je posebno izraženo u problemima u području računalnog vida. Vrlo je teško izlučiti značajke sa slika zbog razlika u osvjetljenju, sjenama te različitim položajima objekata na različitim slikama. Na primjer, u problemu detekcije automobila na slikama, jedna od korisnih značajki može biti nalazi li se na slici kotač ili ne. Međutim, teško je opisati izgled kotača, koji ima jednostavan geometrijski oblik, pomoću vrijednosti intenziteta piksela. Duboko učenje rješava ovaj problem učenjem same reprezentacije. Jedna od temeljnih pretpostavki dubokih modela jest da se reprezentacija podataka može izraziti kompozicijom jednostavnijih reprezentacija, odnosno, kompozicijom značajki kroz više razina hijerarhije. Takav pristup odgovara podatcima u mnogim težim zadacima u području umjetne inteligencije. U prethodnom primjeru s detekcijom automobila, model može kompozitno naučiti reprezentaciju automobila — model iz razlike vrijednosti intenziteta piksela slike može naučiti kako prepoznati rubove, iskoristiti rubove kako bi odredio konture, te od kontura prepoznati diskriminatorne dijelove objekata, poput kotača automobila. (Goodfellow et al., 2016)

Duboki modeli mogu se opisati kao kompozicija više funkcija — slojeva, čije su funkcionalnosti i namjene opisani u nastavku.

2.1. Potpuno povezani sloj

Cilj učenja modela jest aproksimirati funkciju $\boldsymbol{y} = f^*(\boldsymbol{x})$ koja preslikava ulazne primjere $\boldsymbol{x}$ u željene izlaze $\boldsymbol{y}$ parametarskom funkcijom $\hat{\boldsymbol{y}} = f(\boldsymbol{x}, \boldsymbol{\theta})$, pri čemu model nastoji naučiti vrijednosti parametra $\boldsymbol{\theta}$ kojim će dobiti najbolju aproksimaciju funkcije f^*. Funkcija $f(\boldsymbol{x}, \boldsymbol{\theta})$ predstavlja se kompozicijom jednostavnijih funkcija, npr. $f(\boldsymbol{x}, \boldsymbol{\theta}) = f^{(n)}(f^{(n-1)}(\ldots(f^{(1)}(\boldsymbol{x}, \boldsymbol{\theta_1})\ldots), \boldsymbol{\theta_{n-1}}), \boldsymbol{\theta_n})$, pri čemu svaka funkcija $f^{(i)}$ predstavlja jedan sloj modela. Takav model može se prikazati usmjerenim acikličkim grafom koji pokazuje kako su jednostavne funkcije povezane, stoga se takav model naziva potpuno povezana mreža, unaprijedna mreža ili višeslojni perceptron (engl. *multilayer perceptron, MLP*) (Goodfellow et al., 2016). Svaki sloj u takvom modelu sastoji se od proizvoljnog broja jedinica - neurona, koji odgovara broju izlaza promatranog sloja. Svaki neuron povezan je sa svim neuronima iz prethodnog sloja te nije povezan s neuronima iz sloja kojem pripada. Svaki neuron predstavlja funkciju koja vektor izlaza prethodnog sloja preslikava u skalar produktom odziva prethodnog sloja s vektorom težina neurona uz dodavanje praga (engl. *bias*). Međutim, umjesto promatranja pojedinih neurona, znatno je jednostavnije i računski efikasnije cijeli potpuno povezani sloj promatrati kao cjelinu. Svi vektori težina pojedinih neurona mogu se predstaviti matricom $\mathbf{W}$ dimenzija $n \times d$ te svi pomaci vektorom $\boldsymbol{b}$ s n elemenata, gdje d predstavlja broj izlaza prethodnog sloja, a n broj izlaza, odnosno, broj neurona u trenutnom sloju. Tada se odzivi prethodnog sloja za N ulaznih primjera mogu prikazati matricom $\boldsymbol{X}$ dimenzija $N \times d$ čime računanje izlaza postaje vrlo jednostavno, što je prikazano izrazom 2.1 (operator $\oplus$ predstavlja operator zbrajanja uz proširenje[1] (engl. *broadcasting*).

$$\boldsymbol{O} = \boldsymbol{X}\boldsymbol{W}^\top \oplus \boldsymbol{b} \tag{2.1}$$

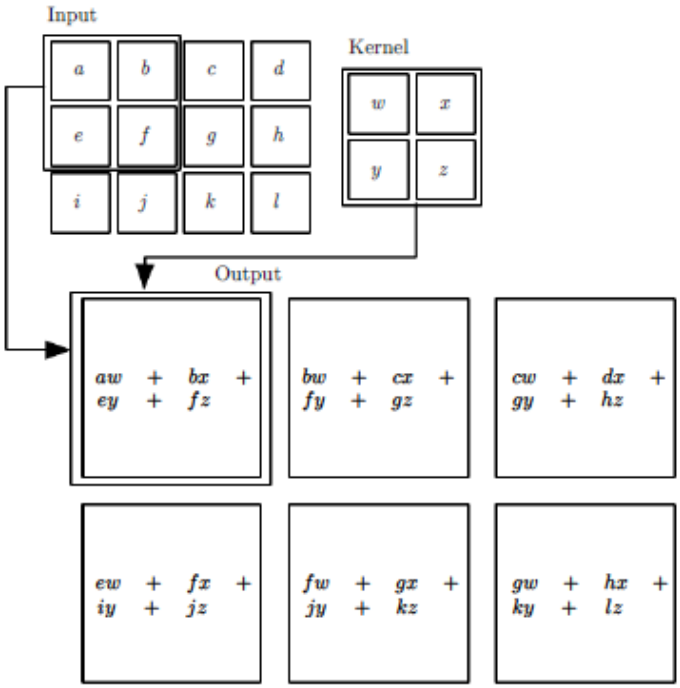
Međutim, ovako opisani slojevi obavljaju linearnu transformaciju te se njihovom kompozicijom mogu opisati isključivo linearne funkcije, stoga se slojevi kombiniraju s nelinearnim aktivacijskim funkcijama, koje su opisane u nastavku ovog poglavlja.

2.2. Konvolucijski sloj

Konvolucijski sloj, u odnosu na prethodno opisani potpuno povezani sloj, umjesto matričnog produkta ulaza i težina koristi operaciju konvolucije. Koristi se za procesiranje ulaznih podataka koji imaju topologiju rešetke, primjerice, vremenskog slijeda (1D rešetka) te slika (2D i 3D topologija rešetke). Primjer korištenja konvolucije prikazan je

[1] `https://docs.scipy.org/doc/numpy/user/basics.broadcasting.html`

na slici 2.1. Motivacija za korištenje konvolucijskih slojeva sadržana je u tri ideje koje



Slika 2.1: Primjer 2D konvolucije s jezgrom veličine 2×2 bez nadopune nulama uz vertikalni i horizontalni pomak po jednom elementu (Goodfellow et al., 2016)

mogu unaprijediti model strojnog učenja:

- lokalne interakcije
- dijeljenje parametara
- ekvivarijantnost s obzirom na translaciju.

Pristup korišten u potpuno povezanim slojevima zapravo modelira interakcije između svakog ulaza i svakog izlaza pomoću matrica težina. Posljedično, svaki neuron u bilo kojem sloju mreže u interakciji je sa svim neuronima ulaznog sloja, izravno ili preko interakcija kroz druge slojeve. Lokalne interakcije mogu se vrlo jednostavno postići korištenjem matrica težina manjih dimenzija od ulaznih primjera. Osim što ovakav pristup znatno smanjuje potreban broj parametara i potreban broj operacija za učenje modela, njime se omogućuje modelu oblikovanje manjih značajki koje se potom kompozicijom mogu kombinirati u složenije značajke. Isti pristup omogućuje dijeljenje parametara — svaki element matrice težina koristi se u više operacija (u odnosu na tradicionalni pristup u kojem se svaki element koristi na samo jednom mjestu), čime je umjesto učenja više skupova parametara potrebno naučiti samo jedan skup. Iako se time ne smanjuje vremenska složenost propagacije primjera kroz model, značajno se

(a) $\begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{pmatrix}$ (b) $\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & a & b & c & d & 0 \\ 0 & e & f & g & h & 0 \\ 0 & i & j & k & l & 0 \\ 0 & m & n & o & p & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$

Slika 2.2: Usporedba ulaznih primjera za konvoluciju bez nadopune nulama (a) i s nadopunom nulama (b). Izlaz konvolucije s filterom 3×3 i pomakom 1 bez nadopune bit će matrica dimenzija 3×3, dok će uz nadopunu izlaz imati dimenzije originalnog ulaza, 4×4.

smanjuje broj parametara modela, što je posebno izraženo u problemima računalnog vida. U problemima klasifikacije slika, svaki neuron u prvom sloju potpuno povezane mreže morao bi imati po jedan parametar za svaki piksel slike, a same slike mogu se sastojati od nekoliko tisuća ili milijuna piksela. Dijeljenje parametara u konvoluciji također uzrokuje ekvivarijantnost na pomake. Neka funkcija f predstavlja konvoluciju, a funkcija g bilo koju funkciju koja predstavlja translaciju ulaza. Tada vrijedi $f(g(x)) = g(f(x))$, što znači da ako se određeni objekt na ulazu u konvolucijski sloj pomakne, na isti će se način pomaknuti i njegova reprezentacija na izlazu iz sloja. Konvolucija, međutim, nije ekvivarijantna na ostale transformacije poput skaliranja i rotacije (Goodfellow et al., 2016).

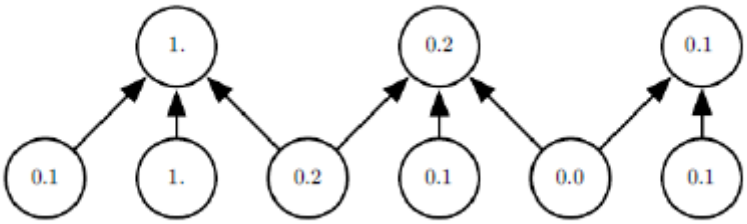
Konvolucija se često provodi uz nadopunu nulama na rubovima ulaznih primjera (engl. *padding*), čime se omogućuje neovisan odabir veličine konvolucijske jezgre i veličine izlaza konvolucije, kao što je prikazano na slici 2.2. Bez nadopune nulama, model gubi na ekspresivnosti jer se s dubinom modela postepeno smanjuje veličina reprezentacije. Međutim, posljedično, elementi na rubovima ulaznih primjera manje utječu na izlaz sloja od elemenata u sredini (Goodfellow et al., 2016).

2.3. Slojevi sažimanja

Funkcija sažimanja (engl. *pooling function*) preslikava prostorno bliske značajke u jednu značajku na izlazu. Preslikavanje se uobičajeno provodi koristeći statističku vrijednost skupa značajki, primjerice, maksimalnu vrijednost, srednju vrijednost, L_2 normu ili težinski prosjek u ovisnosti o udaljenosti od prostorne centralne vrijednosti. Budući da se prostorno bliske značajke preslikavaju u jednu značajku, mali pomak ulaznih značajki neće se u velikoj mjeri preslikati na izlaz, čime reprezentacija postaje

približno invarijantna na male translacije ulaza. Invarijantnost reprezentacije na male pomake može biti vrlo korisna u slučajevima kada je bitnije prepoznati je li određena značajka prisutna nego njenu točnu lokaciju. U kombinaciji sa konvolucijom, model može samostalno naučiti na koje transformacije značajka treba postati invarijantna (Goodfellow et al., 2016).

Sažimanje se također koristi za smanjenje dimenzionalnosti (engl. *downsampling*) ulaza, čime se može povećati računska efikasnost modela. Primjer takvog sažimanja prikazan je na slici 2.3.



Slika 2.3: Primjer sažimanja maksimumom sa susjedstvom od tri elementa i pomakom od dva elementa (Goodfellow et al., 2016)

2.4. Aktivacijske funkcije

Aktivacijske funkcije koriste se kako bi se u model uvela nelinearnost. Svi do sad spomenuti slojevi provode linearnu transformaciju te je njihova kompozicija također linearna, stoga je potrebno uvesti nelinearne slojeve kako bi i sam model bio nelinearan. U praksi se kao aktivacijske funkcije uobičajeno koriste zglobnica, sigmoida te tangens hiperbolni.

2.4.1. Zglobnica

Zglobnica (engl. *rectified linear unit, ReLU*) nad elementima x_i ulaza $\boldsymbol{x}$ provodi funkciju prikazanu izrazom 2.2.

$$g(x_i) = max(0, x_i) \tag{2.2}$$

Prednost zglobnice je da u aktivnom stanju ($x_i > 0$) propušta signal unaprijed i gradijent unatrag. Gradijent nije definiran za $x_i = 0$, međutim, učenje modela se zaustavlja prije dostizanja lokalnog minimuma u kojem je iznos gradijenta nula, te se u

implementaciji može definirati da gradijent u nuli odgovara gradijentu s lijeva (0) ili gradijentu s desna (1).

2.4.2. Sigmoida

Sigmoida nad elementima x_i ulaza $\boldsymbol{x}$ provodi funkciju prikazanu izrazom 2.3, čime ulaze preslikava u interval $(0, 1)$.

$$g(x_i) = \sigma(x_i) = \frac{1}{1 + e^{-x_i}} \tag{2.3}$$

$$\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x)) \tag{2.4}$$

Njena derivacija po ulazima (prikazana izrazom 2.4) može se iskazati preko njenih izlaza čime se ona vrlo efikasno može upotrijebiti prilikom učenja modela. Međutim, u zasićenju ($\sigma(x) \approx 0$ ili $\sigma(x) \approx 1$), prilikom propagacije gradijenta unazad, gradijent se množi sa brojem vrlo bliskim nuli čime i sam postaje vrlo blizak nuli. Problem nestajućeg gradijenta (engl. *vanishing gradient problem*) jako je izražen u modelima s većim brojem slojeva, stoga je u učenju unaprijednih modela u praksi sigmoida zamijenjena zglobnicom, dok se još uvijek koristi u učenju mreža s povratnim vezama te nekih vrsta generativnih modela.

2.4.3. Tangens hiperbolni

Tanges hiperbolni nad elementima x_i ulaza $\boldsymbol{x}$ provodi funkciju prikazanu izrazom 2.5, čime obavlja nelinarno preslikavanje ulaza u interval $(-1, 1)$.

$$g(x_i) = tanh(x_i) = \frac{e^{2x_i} - 1}{e^{2x_i} + 1} \tag{2.5}$$

U učenju se uobičajeno ponaša bolje od sigmoide zbog sličnosti identiteti u okolici $x = 0$, ali također pati od problema zasićenja.

3. Algoritmi učenja modela

Funkcije gubitka

Algoritmi nadziranog učenja optimiziraju model minimizacijom pogreške koja se računa funkcijom gubitka na temelju dobivenih izlaza modela za dane primjere za učenje te njihovih očekivanih izlaza. Učenje se može provesti grupno, koristeći sve podatke iz skupa za učenje, ili nad mini-grupama (engl. *minibatch*) koristeći u svakoj iteraciji samo manji podskup primjera za učenje, čime se proces učenja značajno ubrzava zbog potrebe za manjim brojem unaprijednih prolaza kroz model. Male veličine grupa također u proces učenja unose regularizacijski efekt — pogreška i gradijent nisu u potpunosti precizni, čime se može poboljšati generalizacija. Funkcije gubitka tradicionalno se kombiniraju s regularizacijskim tehnikama, kažnjavajući složenost modela na temelju L_2 ili L_1 norme težina modela (pomnoženih s regularizacijskim faktorom λ). Funkcija gubitka s L_2 regularizacijom prikazana je izrazom 3.1. Dvije funkcije gubitka korištene u ovom radu, srednji kvadratni gubitak te gubitak unakrsne entropije, opisane su u nastavku.

$$E(\boldsymbol{w}|\boldsymbol{y}, \hat{\boldsymbol{y}}) = L(\boldsymbol{y}, \hat{\boldsymbol{y}}) + \frac{\lambda}{2}\boldsymbol{w}^\top \boldsymbol{w} \tag{3.1}$$

Srednji kvadratni gubitak

Srednji kvadratni gubitak (engl. *mean squared error*) za pojedini primjer za učenje računa se prema izrazu:

$$L(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})^2$$

gdje n predstavlja broj izlaza modela, $\boldsymbol{y}$ očekivani izlaz, a $\hat{\boldsymbol{y}}$ dobiveni izlaz. Ukupni gubitak za cijelu mini-grupu računa se kao srednja vrijednost gubitaka za primjere iz mini-grupe.

Gubitak unakrsne entropije

U problemima klasifikacije, izlazi posljednjeg sloja modela mogu se interpretirati kao nenormalizirane log-vjerojatnosti (engl. *logits*) za svaki razred. Nad izlazom posljednjeg sloja $\boldsymbol{z}$ provodi se *softmax* funkcija, kojom se dobivaju vjerojatnosti klase, kao što je prikazano izrazom 3.2.

$$P(\hat{y} = i | f(\boldsymbol{x}, \boldsymbol{\theta})) = softmax(\boldsymbol{z})_i = \frac{exp(z_i)}{\sum_j exp(z_j)} \tag{3.2}$$

Gubitak unakrsne entropije (engl. *cross entropy loss*) računa se izrazom 3.3.

$$L(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_{i=1}^{C} y_i log(\hat{y}_i) \tag{3.3}$$

Navedeni izraz uz uvrštavanje izraza 3.2 se za stvarnu klasu primjera ($y_i = 1$) te ostale klase ($y_i = 0$) svodi na negativnu log-izglednost stvarne klase.

3.1. *Backpropagation* i gradijentni spust

Propagirajući ulaze $\boldsymbol{x}$ kroz model kako bi se dobili izlazi $\hat{\boldsymbol{y}}$, informacije kroz model putuju unaprijed kroz skrivene slojeve prema izlaznom sloju modela. Prilikom učenja modela, dobiveni izlazi $\hat{\boldsymbol{y}}$ koriste se kako bi se izračunao skalarni gubitak za dane primjere za učenje $\boldsymbol{x}$. Algoritam *Backpropagation* (Rumelhart et al., 1986) omogućuje propagaciju dobivenog gubitka unazad kroz izlazni sloj i skrivene slojeve s ciljem izračuna gradijenata gubitka po parametrima modela. Propagirajući gubitak unazad, za određivanje pojedinog gradijenta gubitka po parametrima određenog sloja, koristi se pravilo derivacije kompozicija funkcija, poznato kao i pravilo ulančavanja (engl. *chain rule*). Neka su $\boldsymbol{w}$ parametri promatranog i-tog sloja modela, L gubitak modela za dane ulaze $\boldsymbol{x}$, $\boldsymbol{o}$ te $\boldsymbol{i}$ izlazi i ulazi iz slojeva, te neka se model sastoji od n slojeva. Koristeći pravilo ulančavanja, gradijent gubitka po parametrima $\boldsymbol{w}$ može se iskazati pomoću gradijenata po izlazima i ulazima slojeva koji se nalaze nakon promatranog i-tog sloja, kao što je prikazano izrazom 3.4.

$$\frac{\partial L}{\partial \boldsymbol{w}} = \frac{\partial L}{\partial \boldsymbol{o}_n} \frac{\partial \boldsymbol{o_n}}{\partial \boldsymbol{i_n}} \cdots \frac{\partial \boldsymbol{o}_{i+1}}{\partial \boldsymbol{i}_{i+1}} \frac{\partial \boldsymbol{i}_{i+1}}{\partial \boldsymbol{w}} \tag{3.4}$$

Korekcija parametara i-tog sloja provodi se koristeći algoritam gradijentnog spusta (engl. *gradient descent*), kao što je prikazano izrazom 3.5, pri čemu η predstavlja stopu učenja, $\eta \in (0, 1)$.

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \frac{\partial L}{\partial \boldsymbol{w}} \tag{3.5}$$

Implementacijski promatrano, uz unaprijedni prolaz, potrebno je implementirati računanje gradijenta izlaza iz sloja po ulazu u sloj, te po parametrima sloja, ukoliko postoje. Algoritam *Backpropagation* i danas se dominantno koristi za učenje dubokih modela zahvaljujući jednostavnosti implementacije i računskoj efikasnosti.

Algoritam Adam

Odabir stope učenja ima velik utjecaj na proces učenja modela — premala stopa učenja uzrokovat će vrlo sporu konvergenciju te moguće rano zapinjanje u lošem lokalnom minimumu, dok će prevelika stopa učenja u mnogim slučajevima uzrokovati oscilacije, nestabilnosti ili čak i divergenciju. Prilikom procesa učenja, stopa učenja ne bi trebala biti konstantna — veća stopa učenja u početku može znatno ubrzati konvergenciju, međutim, kako proces učenja odmiče, stopa učenja trebala bi se smanjiti s ciljem izbjegavanja nestabilnosti. Promjenjivost stope učenja može se jednostavno postići postepenim smanjivanjem njene vrijednosti kroz iteracije do određene konačne vrijednosti, međutim, takvim se postupkom u problem uvodi dodatan hiperparametar. Umjesto uvođenja dodatnih hiperparametara, bolje je kao zamjenu za stohastički gradijentni spust iskoristiti već postojeće algoritme s adaptivnim stopama učenja, primjerice, algoritam Adam.

Algoritam Adam (engl. ***Ada**ptive **M**oments*) (Kingma i Ba, 2014) koristi momente prilikom korekcije parametara modela — akumulira eksponencijalno umanjujući prosjek gradijenta i njegovog kvadrata te korekciju parametara provodi na temelju uprosječenih gradijenata umjesto na izravno izračunatom gradijentu. Pseudokôd algoritma prikazan je na slici 3.1. U praksi je pokazao dobre rezultate i veliku robusnost na vrijednost stope učenja i ostalih vlastitih parametara.

3.2. Genetsko kaljenje

Genetsko kaljenje (engl. *Genetic Annealing, GA*) (Yao, 1991) je populacijska varijanta optimizacijskog algoritma simulirano kaljenje (engl. *Simulated Annealing, SA*). Pretraživanje prostora rješenja provodi se genetskim operatorom mutacije nad populacijom rješenja. Korištena implementacija algoritma temelji se na još jednoj varijaciji *SA* algoritma, *Annealed demon* algoritmu (Talbi, 2009). Originalni *demon* algoritam (Creutz, 1983) zasnovan je na ideji da je energija sustava konstantna, pri čemu potencijalna energija sustava odgovara vrijednosti funkcije cilja za dano rješenje. Poboljšanje rješenja uzrokuje smanjenje potencijalne energije, čija se razlika pribraja *demonu*,

Require: Step size ϵ (Suggested default: 0.001)

Require: Exponential decay rates for moment estimates ρ_1 and ρ_2 in $[0, 1)$. (Suggested defaults: 0.9 and 0.999 respectively)

Require: Small constant δ used for numerical stabilization. (Suggested default: 10^{-8})

Require: Initial parameters $\boldsymbol{\theta}$

Initialize 1st and 2nd moment variables $\boldsymbol{s} = \mathbf{0}, \boldsymbol{r} = \mathbf{0}$

Initialize time step $t = 0$

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$

Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

$t \leftarrow t + 1$

Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g}$

Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g}$

Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \rho_1^t}$

Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \rho_2^t}$

Compute update: $\Delta\boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$ (operations applied element-wise)

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

end while

Slika 3.1: Pseudokôd algoritma Adam (Goodfellow et al., 2016). Operator $\odot$ predstavlja množenje po elementima (engl. *element-wise*).

dodatnom stupnju slobode u sustavu čija je vrijednost strogo pozitivna. Nepoboljšavajuće rješenje povećava potencijalnu energiju sustava pri čemu se razlika oduzima od vrijednosti *demona*. Početna vrijednost *demona* te iznos funkcije cilja početnog rješenja definiraju ukupnu energiju sustava, a vrijednost *demona* definira prihvaćanje nepoboljšavajućih rješenja. Za razliku od standardnog algoritma *SA*, prihvaćanje nepoboljšavajućih rješenja je deterministično, ovisno je o razlici iznosa funkcije cilja novog i prethodnog rješenja — rješenje će biti prihvaćeno ukoliko je razlika iznosa manja od vrijednosti *demona* zbog strogog ograničenja na pozitivnost njegove vrijednosti. Zahvaljujući determinističkoj odluci o prihvaćanju rješenja, algoritam pokazuje bolje vremenske performanse od standardnog *SA* algoritma jer nije potrebno generirati slučajne brojeve. *Annealed demon* varijacija algoritma uklanja ograničenje na konstantnost energije u sustavu te se vrijednost *demona* eksponencijalno smanjuje kroz iteracije algoritma, kako bi se smanjilo prihvaćanje nepoboljšavajućih rješenja u kasnijim iteracijama algoritma. U populacijskoj implementaciji algoritma korištenoj u ovom radu, vrijednost *demona* dijele sve jedinke u populaciji. Pseudokôd korištenog algoritma prikazan je na slici 3.2.

3.3. Evolucijska strategija

Evolucijska strategija, kao i genetski algoritmi, spada u skupinu evolucijskih algoritama, koji koriste evolucijske operatore (selekcija, križanje, mutacija) nad populacijom rješenja kako bi pronašli optimalno rješenje za dani problem. Operator križanja koristi se kako bi se kombinacijom postojećih rješenja u populaciji dobilo novo rješenje, a operator mutacije kako bi se uvele slučajne promjene u rješenja, čime se može proširiti prostor pretraživanja. U evolucijskoj strategiji koriste se dvije populacije — populacija roditelja (veličine μ) te populacija potomaka (veličine λ, $\lambda \geq \mu$) koja se stvara od jedinki iz populacije roditelja koristeći evolucijske operatore. Po završetku generacije algoritma, nova populacija roditelja stvara se koristeći jedinke iz obje populacije (Talbi, 2009). Pseudokôd algoritma prikazan je na slici 3.3. U implementaciji algoritma korištenoj u ovom radu, nova populacija roditelja stvara se koristeći najbolje jedinke iz obje populacije (ili samo od jedinki iz populacije potomaka), a roditelji za stvaranje populacije potomaka odabiru se nasumično.

Randomly select an initial population of `N` configurations.
for `i=1` to `N` **do**
    Initialize the i*th* threshold `Th[i]` with the energy of the i*th* configuration.
end for
Empty the energy bank: `DE = 0`
repeat
    Begin cooling loop:
    for `i=1` to `N` **do**
        Splice or randomly mutate the i*th* configuration.
        Compute the energy, `E`, of the resulting mutant.
        if `E > Th[i]` **then**
            Restore the old configuration.
        end if
        if `E <= Th[i]` **then**
            Add energy difference to `DE`: `DE=DE+Th[i]-E`
            Reset threshold: `Th[i]=E`
            Replace the old configuration with successful mutant.
        end if
    end for
    Compute reheating increment, `dE`: `dE=DE*C/N, 0<C<1`
    Begin reheating loop:
    for `i=1` to `N` **do**
        Add `dE` to each threshold: `Th[i] = Th[i]+dE`
    end for
until stopping criteria

Slika 3.2: Pseudokôd genetskog kaljenja (Price, 1994). Inicijalna vrijednost praga `Th[i]` postavlja se na vrijednost funkcije cilja jedinke uz pribrojenu inicijalnu vrijednost hiperparametra *energy bank* (*demon*) podijeljenu s brojem jedinki u populaciji. Drugi hiperparametar algoritma je faktor hlađenja (`C`), kojim se regulira koliko će se brzo smanjivati ukupna energija sustava, odnosno, smanjenje prihvaćanja nepoboljšavajućih rješenja.

Initialize a population of μ individuals.
Evaluate the μ individuals.
repeat
    Generate λ offsprings from μ parents.
    Evaluate the λ offsprings.
    Replace the population with μ individuals from parents and offsprings.
until stopping criteria

Slika 3.3: Pseudokôd evolucijske strategije (Talbi, 2009)

3.4. *Steady State Tournament*

Algoritam *Steady State Tournament* je genetski algoritam s turnirskom eliminacijskom selekcijom (Talbi, 2009). U svakoj iteraciji algoritma, nasumično se odabere k jedinki te se najlošija od njih zamjeni produktom križanja dvije jedinke (uzetih iz skupa prethodno odabranih k jedinki, uz zanemarivanje najlošije jedinke), te se nad dobivenom jedinkom provede mutacija. Veličina turnira (k) je hiperparametar algoritma s minimalnom vrijednosti 3. Algoritam ima implicitno svojstvo elitizma — najbolja jedinka ne može biti odabrana za eliminaciju iz populacije. Implementacijski promatrano, jedna generacija algoritma obično uključuje N iteracija (N je veličina populacije), kako bi broj evaluacija funkcije cilja po jednoj generaciji odgovarao broju evaluacija ostalih populacijskih algoritama. Pseudokôd algoritma prikazan je na slici 3.4.

3.5. Algoritam Hooke-Jeeves

Algoritam Hooke-Jeeves koristi se za pronalaženje optimuma unimodalnih funkcija iterativnim pomacima po svakoj dimenziji trenutnog rješenja koristeći tri vektora: bazni vektor $\boldsymbol{x}_B$, početni vektor pretraživanja $\boldsymbol{x}_P$ te vektor $\boldsymbol{x}_N$ dobiven pretraživanjem uz pomak Δx. Početno, vektori $\boldsymbol{x}_B$ i $\boldsymbol{x}_P$ postavljaju se na iste vrijednosti koje se mogu stvoriti nasumično. Novo rješenje $\boldsymbol{x}_N$ stvara se iteriranjem po svim dimenzijama vektora $\boldsymbol{x}_P$ uz dodavanje ili oduzimanje vrijednosti Δx. Ukoliko pomak poboljšava vrijednost funkcije cilja, on se prihvaća te se pretraživanje nastavlja kroz ostale dimenzije. Ukoliko je vrijednost funkcije cilja novo dobivenog rješenja $\boldsymbol{x}_N$ nakon prolaska kroz sve dimenzije bolja od vrijednosti funkcije cilja vektora $\boldsymbol{x}_B$, pretraživanje se nastavlja s novim početnim vektorom $\boldsymbol{x}_P = 2\boldsymbol{x}_N - \boldsymbol{x}_B$, a $\boldsymbol{x}_N$ postaje novi bazni vektor.

Require: Tournament size `k`.
  Initialize a population of `N` individuals.
  repeat
    for `i=0` to `N` **do**
      Randomly add `k` individuals to the tournament.
      Select the worst individual from the tournament.
      Randomly select two parents from remaining individuals in the tournament.
      Replace the worst individual with crossover child.
      Perform mutation on child.
    end for
  until stopping criteria

Slika 3.4: Pseudokôd algoritma *Steady State Tournament*

Ako novo dobiveno rješenje $\boldsymbol{x}_N$ ne predstavlja bolje rješenje od baznog vektora $\boldsymbol{x}_B$, vrijednost pomaka se smanjuje te se pretraživanje nastavlja iz baznog vektora. Uvjet zaustavljanja algoritma temelji se na vrijednosti pomaka — pretraživanje se provodi sve dok vrijednost pomaka ne postane manja od zadane preciznosti ϵ.

Budući da funkcija gubitka dubokog modela koja se učenjem nastoji minimizirati nije unimodalna, algoritam Hooke-Jeeves je u ovom radu korišten u genetskoj varijanti u kojoj se pretraživanje provodi nad populacijom rješenja prethodno opisanim postupkom. Dodatno, nakon što vrijednost pomaka za određeno rješenje u populaciji postane manja od zadane preciznosti, ono se zamjenjuje novim rješenjem. Novo rješenje stvara se križanjem roditelja odabranih po distribuciji vjerojatnosti temeljenoj na vrijednostima funkcije cilja cijele populacije (proporcionalna selekcija) te se nad dobivenim rješenjem provodi mutacija. Pseudokôd korištenog algoritma prikazan je na slici 3.5.

Budući da je tijekom jedne iteracije algoritma nad svakim rješenjem u populaciji potrebno provesti dvije evaluacije funkcije cilja (provjera je li novo rješenje bolje za pozitivan ili negativan pomak) po svakoj dimenziji rješenja, uporaba navedenog algoritma ograničena je isključivo na vrlo jednostavne modele zbog prevelike vremenske složenosti.

Require: Initial step size Δx

Require: Precision ϵ

Initialize population of N individuals with starting vectors $\boldsymbol{x}^{(i)}$ and base vectors $\boldsymbol{x}_B^{(i)}$.

Initialize steps $\Delta x^{(i)}$.

repeat

for every starting vector $\boldsymbol{x}^{(i)}$ in population **do**

if $\Delta x^{(i)} > \epsilon$ **then**

for every value $x_j^{(i)}$ in $\boldsymbol{x}^{(i)}$ **do**

$x_j^{(i)} \leftarrow x_j^{(i)} + \Delta x^{(i)}$

if change improves fitness **then**

continue

end if

$x_j^{(i)} \leftarrow x_j^{(i)} - 2\Delta x^{(i)}$

if change does not improve fitness **then**

Return to original value: $x_j^{(i)} \leftarrow x_j^{(i)} + \Delta x^{(i)}$.

end if

end for

if changes improved fitness compared to $\boldsymbol{x}_B^{(i)}$ **then**

Set new value for starting vector: $\hat{\boldsymbol{x}}^{(i)} \leftarrow 2\boldsymbol{x}^{(i)} - \boldsymbol{x}_B^{(i)}$.

Update base vector: $\boldsymbol{x}_B^{(i)} \leftarrow \boldsymbol{x}^{(i)}$.

Update starting vector: $\boldsymbol{x}^{(i)} \leftarrow \hat{\boldsymbol{x}}^{(i)}$.

else

Reduce the step size: $\Delta x^{(i)} \leftarrow \Delta x^{(i)} * 0.5$.

Set base vector as a new starting vector: $\boldsymbol{x}^{(i)} \leftarrow \boldsymbol{x}_B^{(i)}$.

end if

else if this is not the best individual in population **then**

Select two parents from population for crossover.

Create crossover child and set $\boldsymbol{x}^{(i)}$ to its value.

Mutate $\boldsymbol{x}^{(i)}$ and reinitialize $\Delta x^{(i)}$.

end if

end for

until stopping criteria

Slika 3.5: Pseudokôd genetske varijante algoritma Hooke-Jeeves

Input: Initial population P_0.
$P = P_0$ /* Generation of the initial population of random antibodies */
repeat
    Evaluate all existing antibodies and compute their affinities.
    Select N antibodies with highest affinities.
    Clone the selected antibodies.
    Maturate the cloned antibodies.
    Evaluate all cloned antibodies.
    Add R best cloned antibodies to the pool of antibodies.
    Remove worst members of the antibodies pool.
    Add new random antibodies into the population.
until stopping criteria satisfied

Slika 3.6: Pseudokôd algoritma *Clonal Selection* (Talbi, 2009)

3.6. Algoritam *Clonal Selection*

Algoritam *Clonal Selection* (CLONALG) je evolucijski algoritam inspiriran imunološkim sustavom koji pretražuje prostor rješenja populacijom antitijela koja predstavljaju potencijalna rješenja. Iteracija algoritma sastoji se od nekoliko faza, pri čemu prvu fazu čini postupak kloniranja prilikom kojeg se stvara određeni broj klonova N najboljih antitijela u populaciji. Nakon kloniranja, novo stvorena antitijela ulaze u fazu sazrijevanja u kojoj se nad njima provodi hipermutacija — mutacija koja se ponavlja više puta. Broj ponavljanja obrnuto je proporcionalan kvaliteti antitijela — što je antitijelo bolje u usporedbi s ostalim antitijelima u populaciji, nad njim će se provesti manji broj mutacija. U sljedećoj fazi stvara se nova populacija odabirom R najboljih klonova te se preostala mjesta u populaciji popune novim nasumično stvorenim antitijelima kako bi se omogućilo proširenje prostora pretraživanja (Talbi, 2009). Pseudokôd algoritma prikazan je na slici 3.6.

4. Programska implementacija

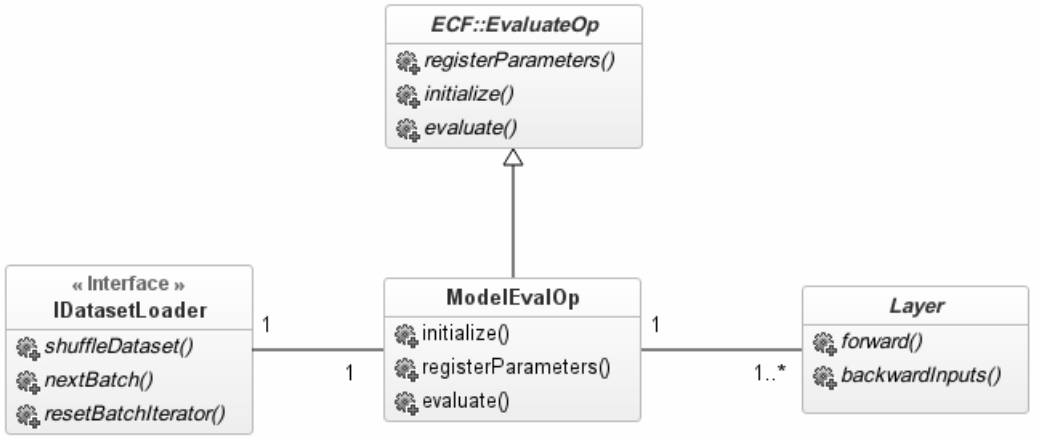
Programska implementacija za učenje dubokih modela izvedena u sklopu ovog rada napisana je u jeziku C++ koristeći programske okvire ECF[1] i TensorFlow[2]. ECF je programski okvir za evolucijsko računanje, sadrži implementacije brojnih evolucijskih algoritama (uključujući *Genetic Annealing*, *Steady State Tournament*, evolucijsku strategiju, CLONALG i genetsku varijantu algoritma Hooke-Jeeves, koji su korišteni u sklopu ovog rada) te potrebne funkcionalnosti za njihovo korištenje — različite genotipove (strukture podataka za opis jedinki), evolucijske operatore te mehanizme za konfiguriranje njihovih parametara.

TensorFlow je programski okvir otvorenog kôda za obavljanje numeričkih operacija putem računskih grafova. Čvorovi grafa predstavljaju numeričke operacije, a bridovi grafa predstavljaju višedimenzionalne skupove podataka (tenzore) koji predstavljaju rezultate numeričkih operacija te se koriste u komunikaciji između čvorova. Osim implementacije struktura podataka i osnovnih operacija potrebnih za učenje modela (matematičke operacije nad matricama i tenzorima) s podrškom za paralelizaciju na CPU i GPU, TensorFlow također sadrži i implementacije složenijih operacija, poput konvolucije za korištenje u konvolucijskim slojevima, implementacije raznih slojeva sažimanja, automatsko diferenciranje te mnoge druge funkcionalnosti. Sadrži vrlo bogato klijentsko sučelje prema programskom jeziku Python, međutim, u trenutku pisanja ovog rada, sučelje prema jeziku C++ još je uvijek vrlo slabo dokumentirano te ne sadržava brojne funkcionalnosti koje su sadržane u sučelju prema Pythonu (primjerice, automatsko diferenciranje).

Pomoću programskog paketa implementiranog u sklopu ovog rada moguće je definirati proizvoljan duboki model koristeći dostupne slojeve te model naučiti bilo kojim algoritmom sadržanim u ECF-u (koji podržava *FloatingPoint* genotip) bez potrebe za ponovnim prevođenjem izvornog kôda. Naučeni model moguće je spremiti te ga koristiti i u drugim aplikacijama baziranim na TensorFlowu. Cijeli izvorni kôd programske

[1] `http://ecf.zemris.fer.hr/`

[2] `https://www.tensorflow.org/`



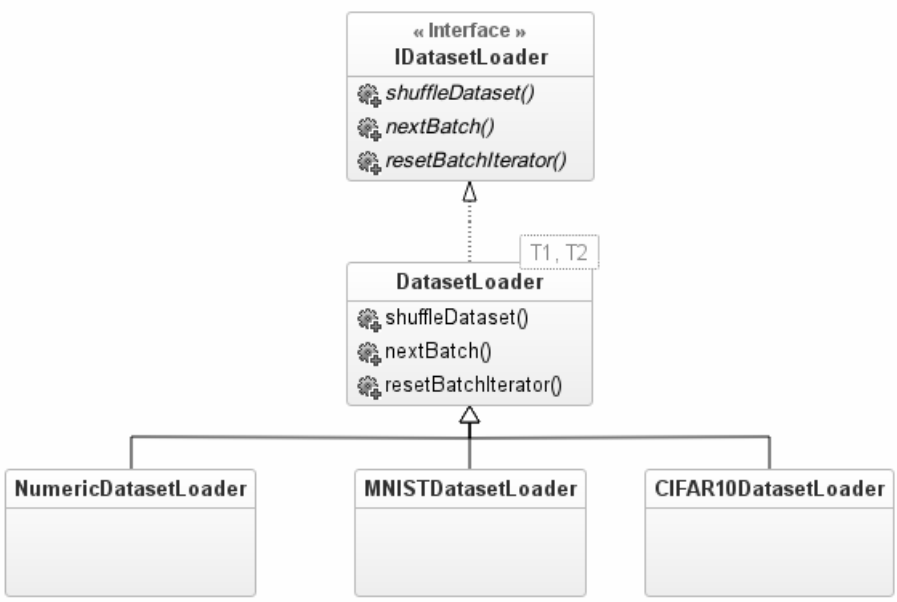
Slika 4.1: Pojednostavljeni dijagram razreda s prikazom odnosa za *ModelEvalOp* razred (prikazane su samo osnovne javne metode)

implementacije, zajedno s uputama za instalaciju i izgradnju TensorFlowa te uputama za korištenje nalazi se na javno dostupnom repozitoriju[3], a u nastavku je dan kratak opis arhitekture implementacije.

Osnovni dio programskog paketa čini razred *ModelEvalOp* koji uz provođenje evaluacije modela nad skupom podataka za učenje, provodi i usklađivanje ostalih komponenata sustava. Prilikom inicijalizacije evaluacijskog operatora, na temelju parametrizacije provodi se čitanje skupa podataka za učenje (preko sučelja *IDatasetLoader*) i izgradnja modela stvaranjem slojeva. Zbog redoslijeda inicijalizacije komponenata ECF-a, u trenutku inicijalizacije populacije nije poznat stvaran broj parametara modela, stoga evaluacijski operator provodi reinicijalizaciju populacije rješenja na temelju stvarnog broja parametara modela stvaranjem slučajnih brojeva prema distribuciji zadanoj konfiguracijskom datotekom. Pojednostavljeni model sustava s odnosima modela razreda *ModelEvalOp* prikazan je na slici 4.1.

Upravljanje skupom podataka za učenje obavlja se putem sučelja *IDatasetLoader*. Sučelje je realizirano preko razreda *DatasetLoader* čije su osnovne funkcionalnosti provođenje slučajne permutacije skupa podataka za učenje te podjela skupa na minigrupe. Samo čitanje skupova podataka implementirano je u konstruktorima razredima koji nasljeđuju razred *DatasetLoader*. Budući da skupovi podataka nemaju standardizirani format, za većinu skupova podataka potrebno je implementirati specijalizirani razred. Programska implementacija podržava čitanje skupova MNIST i CIFAR-10 te numeričkih skupova podataka pohranjenih u obliku tekstualnih datoteka. Pojednostavljena arhitektura modela za upravljanje podacima za učenje prikazana je na dijagramu

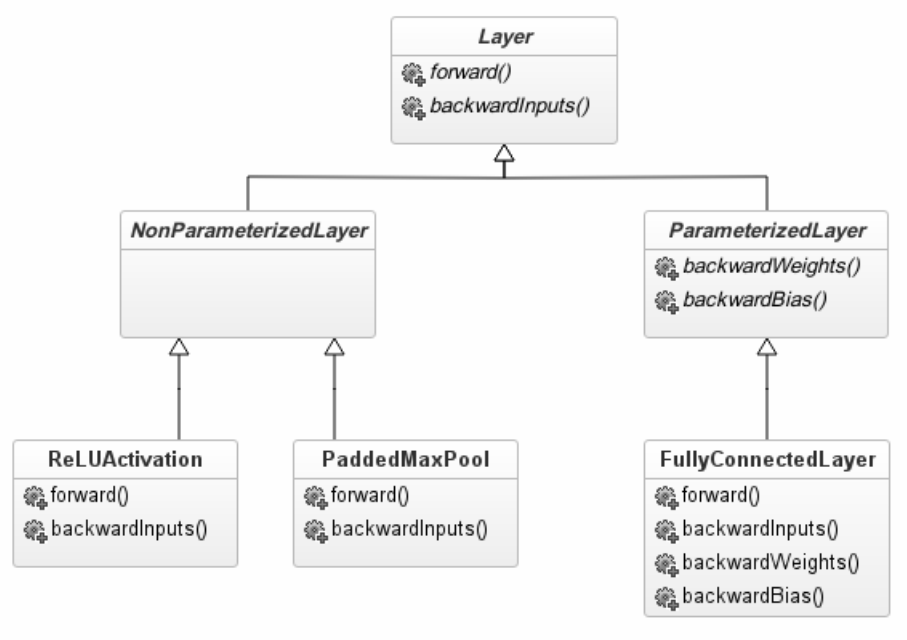
[3] `https://bitbucket.org/lkrizan/ecfdeepmodeltraining`



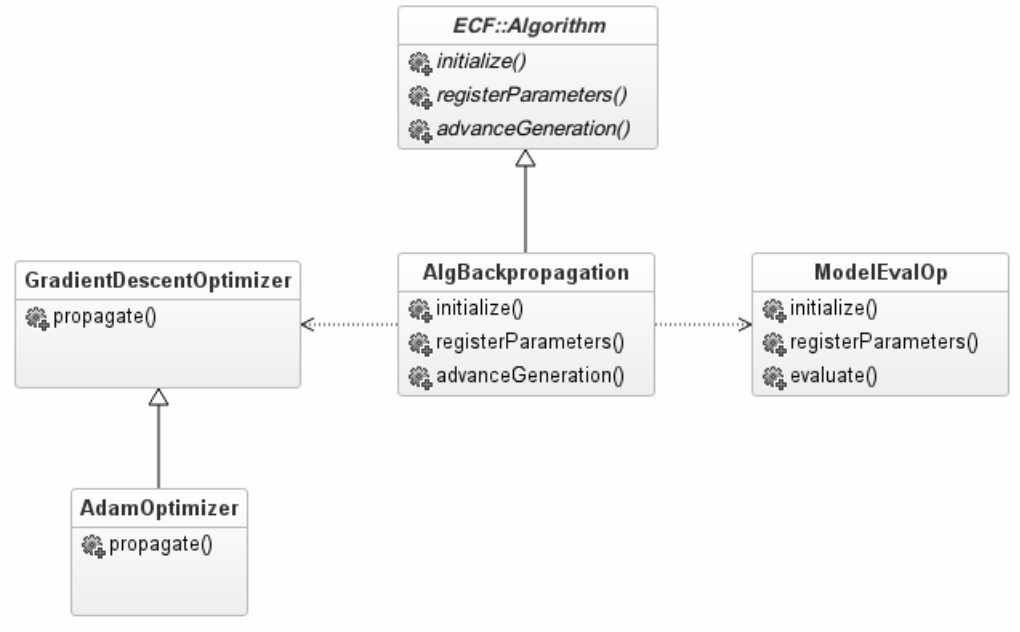
Slika 4.2: Pojednostavljeni dijagram razreda s prikazom odnosa za *DatasetLoader* razred

razreda na slici 4.2.

Slojevi modela opisani su pomoću apstraktnih razreda *Layer*, *NonParameterizedLayer* (aktivacijske funkcije, slojevi sažimanja) te *ParameterizedLayer* (potpuno povezani sloj, konvolucijski sloj). Pomoću konstruktora svakog naslijeđenog razreda stvaraju se čvorovi u računskom grafu koji opisuju operaciju koju obavlja sloj. Izlazni čvor sloja moguće je dohvatiti metodom *forward*. Svi slojevi moraju implementirati metodu *backwardInputs* kojom se računa gradijent izlaza sloja po ulazima u sloj te razredi naslijeđeni iz razreda *ParameterizedLayer* moraju implementirati metode *backwardWeights* i *backwardBias* za izračun gradijenta po parametrima sloja kako bi se omogućio rad algoritma *Backpropagation*. Računski graf za izračun gradijenta unazad te izračun korekcija parametara, zbog redoslijeda inicijalizacije komponenata, stvara se prilikom prvog poziva metode *advanceGeneration* nad objektom razreda *AlgBackpropagation* prolaskom unazad kroz slojeve mreže pozivajući prethodno spomenute metode. Pojednostavljeni model opisa slojeva i model algoritma *Backpropagation* prikazani su na dijagramima razreda na slikama 4.3 i 4.4.



Slika 4.3: Pojednostavljeni dijagram razreda s prikazom odnosa za *Layer* razred



Slika 4.4: Pojednostavljeni dijagram razreda s prikazom odnosa za *AlgBackpropagation* razred

5. Usporedba rezultata

Usporedba uspješnosti učenja modela provedena je na različitim problemima koristeći modele različite složenosti. Uz algoritme opisane u poglavlju 3, u problemima klasifikacije slika korišten je i hibridni algoritam koji nakon korekcija parametara *Backpropagation*/Adam algoritmom provodi dodatne korekcije parametara izvođenjem nekoliko generacija jednog od opisanih populacijskih algoritama, čime bi se mogla ostvariti brža konvergencija i bolji konačni rezultat učenja. U populacijskim algoritmima, korištena su dva operatora mutacije s jednakim vjerojatnostima izvođenja — zamjena određenog parametra brojem generiranim uniformnom razdiobom te korekcija određenog parametra brojem generiranim normalnom razdiobom. Očekivano vrijeme učenja za populacijske algoritme znatno je dulje u odnosu na vrijeme učenja *Backpropagation* algoritmom zbog većeg potrebnog broja evaluacija. Početna pretpostavka prije učenja modela bila je da će uspješnost učenja populacijskim algoritmima u usporedbi s *Backpropagation* algoritmom padati s porastom složenosti modela zbog povećanja broja parametara čime se znatno povećava prostor mogućih rješenja. Prilikom korištenja većih skupova podataka za učenje, učenje više nije praktično provoditi nad cijelim skupom podataka zbog ograničenja memorije grafičke kartice te je uobičajena praksa učenje provoditi koristeći mini-grupe. Takav pristup može poboljšati rezultate učenja gradijentnim postupcima zbog regularizacijskog efekta koji nastaje nepreciznom procjenom gradijenta, međutim, pretpostavka je da će učenje korištenjem mini-grupa smanjiti uspješnost učenja populacijskim algoritmima zbog neprecizne procjene pogreške, na kojoj se temelji iznos funkcije cilja.

Učenje modela provedeno je na računalu s procesorom Intel i5-4460 @ 3.20 GHz, 8 GiB radne memorije te grafičkom karticom NVIDIA GeForce GTX 960 s 1024 *cuda* jezgara i 2 GiB memorije. Svi rezultati prikazani u tablicama i grafovima u nastavku dobiveni su kao srednja vrijednost 20 mjerenja za dane parametre.

5.1. Jednostavna regresija

Kao prvi problem za učenje modela korištena je regresija dvije funkcije s dva ulazna parametra, koje su prikazane u tablici 5.1. Navedene funkcije su odabrane jer se mogu aproksimirati jednostavnim modelom s vrlo malim brojem parametara čime bi svi korišteni algoritmi trebali ostvariti dobre rezultate. Korištena je vrlo jednostavna arhi-

Tablica 5.1: Funkcije korištene u problemu regresije

Oznaka	Izraz
f1	$f(x, y) = (x-1)^2 + (y-2)^2$
f2	$f(x, y) = \frac{8}{2 + x^2 + y^2}$

tektura modela — potpuno povezani sloj s 10 neurona sa sigmoidalnom aktivacijskom funkcijom te izlaznim potpuno povezanim slojem s jednim neuronom, koji se ukupno sastoje od 41 parametra koje je potrebno naučiti. Kao funkcija gubitka, korištena je srednja kvadratna pogreška sa L_2 regularizacijom.

Za regresiju funkcije f1 iz tablice 5.1 korišteni primjeri za učenje i evaluaciju generirani su koristeći ulazne parametre iz intervala $[-3, 3]$ te se skup za učenje sastojao od 4000 primjera, a skup za evaluaciju od 1625 primjera. Učenje *Backpropagation* algoritmom provedeno je uz korekciju parametara modela Adam algoritmom uz stopu učenja 10^{-2}. *Steady State Tournament* (SST) korišten je sa veličinom turnira 3 i populacijom veličine 30, *Genetic Annealing* (GAn) s elitizmom uz početnu vrijednost *demona* 25, faktorom hlađenja 0.8 te populacijom od 30 jedinki. Evolucijska strategija (ES) korištena je s dvije podpopulacije sa 15 roditelja od kojih se stvara 15 potomaka mutacijom. Algoritam *Clonal Selection* (CLONALG) korišten je uz kloniranje 20 antitijela po iteraciji, bez dodavanja nasumično stvorenih antitijela u populaciju na kraju iteracije te veličinu populacije 20. Genetska varijanta algoritma Hooke-Jeeves (GHJ) korištena je uz početnu vrijednost pomaka 1, preciznost 10^{-6} te veličinu populacije 5. Regularizacijski faktor korišten prilikom učenja za sve algoritme iznosio je 10^{-4} te je vjerojatnost mutacije u svim populacijskim algoritmima iznosila 0.5. Učenje svim korištenim algoritmima provedeno je u 7500 iteracija (osim za algoritam Hooke-Jeeves) koristeći cijeli skup za učenje prilikom evaluacije i izračuna gradijenata. Učenje algoritmom Hooke-Jeeves provedeno je u samo 1000 iteracija zbog dugog vremena izvođenja i vrlo brze konvergencije. Početne vrijednosti parametara generirane su po uniformnoj razdiobi u intervalu $[-2, 2]$. Prvi operator mutacije generirao je nove vri-

jednosti težina po uniformnoj razdiobi u intervalu $[-10, 10]$, a drugi operator mutacije je vrijednost korekcije težina generirao po normalnoj razdiobi sa srednjom vrijednosti 0 i standardnom devijacijom 3. Prilikom učenja, svakih 10 iteracija spremane su trenutne vrijednosti parametara modela za najbolju jedinku u populaciji kako bi se mogla prikazati promjena pogreške na evaluacijskom skupu podataka.

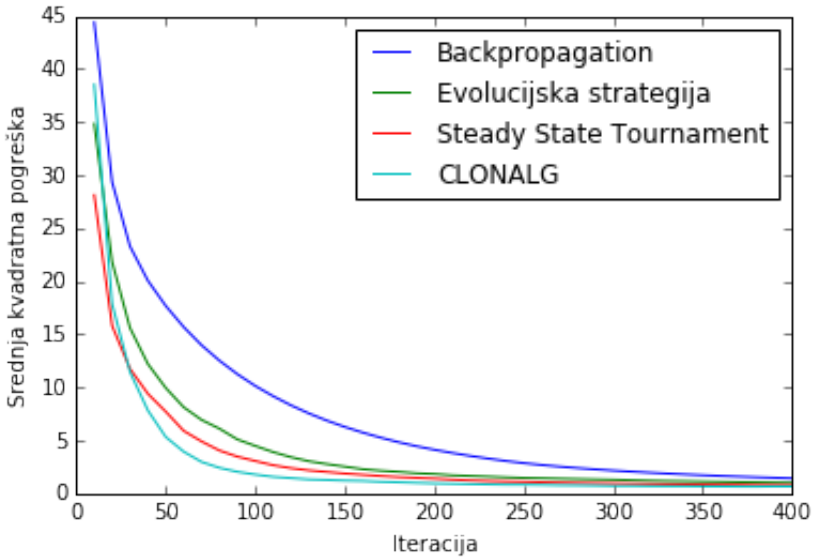
U tablici 5.2 te na slikama 5.1 i 5.2 prikazano je kretanje srednje kvadratne pogreške na evaluacijskom skupu po iteracijama učenja, a u tablici 5.3 prikazano je vrijeme izvođenja za korištene algoritme. Iz dobivenih rezultata vidi se da algoritmi *Steady State Tournament*, evolucijska strategija i CLONALG u prvih približno 1000 iteracija brže konvergiraju od algoritma *Backpropagation*, međutim, znatno brže ulaze u zasićenje te zbog toga u konačnici ostvaruju višu pogrešku na evaluacijskom skupu podataka od algoritma *Backpropagation*. Genetska varijanta algoritma Hooke-Jeeves pokazala je znatno bolje rezultate od ostalih algoritama, međutim, budući da je vrijeme evaluacije rješenja dominantno u ukupnom vremenu izvođenju algoritma te da broj evaluacija po iteraciji algoritma GHJ ne ovisi samo o broju jedinki u populaciji, nego i o broju parametara modela, moguće ga je u razumnom vremenu koristiti isključivo nad modelima s vrlo jednostavnom arhitekturom. Algoritmi *Genetic Annealing* i CLONALG pokazali su najlošije rezultate od korištenih algoritama, međutim, kada se uzme u obzir da se izlaz korištene funkcije za ulaze $x, y \in [-3, 3]$ nalazi u intervalu $[0, 41]$, te da srednja vrijednost izlaza za korištene primjere iznosi 11.61, dobivena srednja kvadratna pogreška na evaluacijskom skupu za sve algoritme je zadovoljavajuća.

Tablica 5.2: Srednja kvadratna pogreška na evaluacijskom skupu podataka po iteracijama za korištene algoritme učenja u problemu regresije funkcije f1

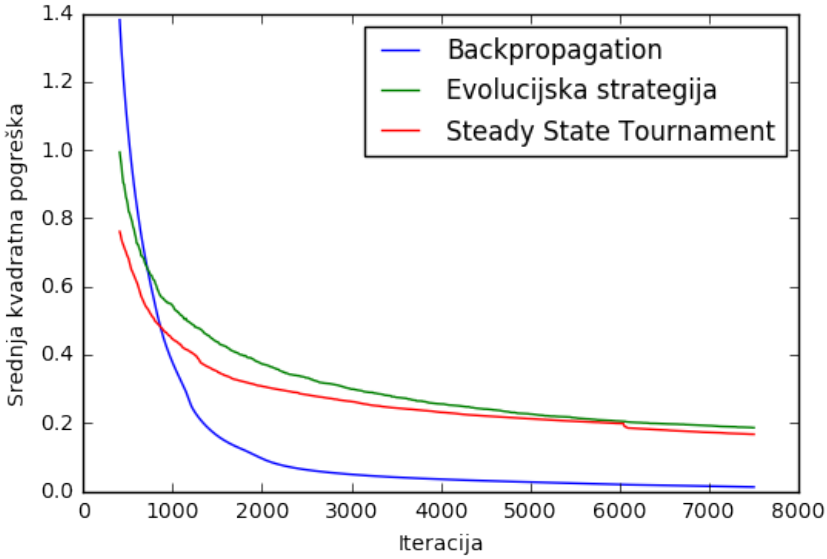
Iteracija	*Backprop*	ES	GAn	SST	CLONALG	GHJ
10	44.3810	34.8289	80.3836	28.1177	38.5238	**1.61149**
50	17.6691	9.94093	37.3064	7.70916	5.31882	**0.079367**
100	10.1363	4.48535	23.2708	3.06478	1.78714	**0.024308**
500	1.06487	0.840271	6.82383	0.687032	0.547925	**0.003516**
1000	0.377694	0.543626	3.132717	0.446689	0.444438	**0.003137**
2500	0.062537	0.332387	1.461709	0.282539	0.346213	-
5000	0.026526	0.226032	0.924315	0.212419	0.319583	-
7500	0.012173	0.185952	0.734247	0.166825	0.343024	-

Tablica 5.3: Usporedba vremena učenja za korištene algoritme u problemu regresije funkcije f1. Učenje je provedeno kroz 1000 iteracija za algoritam GHJ te kroz 7500 iteracija za ostale algoritme.

Algoritam	$\bar{T}$ (s)	σ_T
Backpropagation	55.401	5.188
Evolucijska strategija	169.810	1.644
Genetic Annealing	165.179	1.058
Steady State Tournament	177.724	10.653
CLONALG	466.222	14.810
GHJ	254.817	6.452



Slika 5.1: Prikaz kretanja srednje kvadratne pogreške na evaluacijskom skupu primjera do 400. iteracije u problemu regresije funkcije f1 za korištene algoritme (bez algoritma *Genetic Annealing* i genetske varijante algoritma Hooke-Jeeves)



Slika 5.2: Prikaz kretanja srednje kvadratne pogreške na evaluacijskom skupu primjera od 400. iteracije do konačne iteracije u problemu regresije funkcije f1 za korištene algoritme (bez algoritama *Genetic Annealing*, CLONALG i genetske varijante algoritma Hooke-Jeeves)

Za regresiju funkcije f2 iz tablice 5.1 korišteni primjeri za učenje i evaluaciju generirani su koristeći ulazne parametre iz intervala $[-10, 10]$. Učenje je provedeno na istoj arhitekturi modela kao i u prethodnom primjeru uz isti skup algoritama sa istim parametrima uz jednake veličine skupova podataka za učenje i evaluaciju. Zbog razlike u kodomeni funkcije, početne vrijednosti parametara generirane su po normalnoj razdiobi sa srednjom vrijednosti 0 i standardnim odstupanjem 0.1, uz ponovno generiranje vrijednosti ukoliko ona odstupa od srednje vrijednosti za više od dvije vrijednosti standardnog odstupanja. Prvi operator mutacije generirao je nove vrijednosti težina po uniformnoj razdiobi u intervalu $[-1, 1]$, a drugi operator mutacije je vrijednost korekcije težina generirao po normalnoj razdiobi sa srednjom vrijednosti 0 i standardnom devijacijom 0.5.

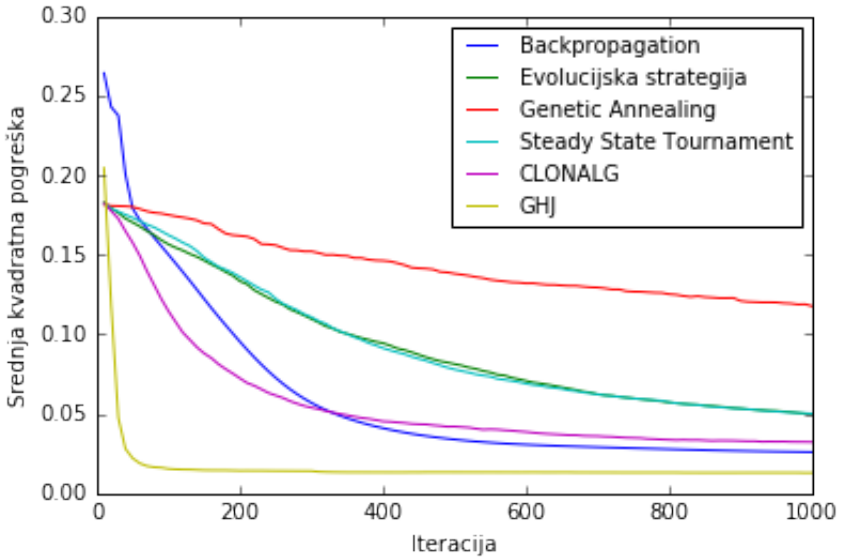
Dobiveni rezultati kretanja srednje kvadratne pogreške na skupu primjera za evaluaciju prikazani su u tablici 5.4 te na slici 5.3, a vrijeme učenja u tablici 5.5. Dobiveni rezultati u velikoj mjeri odgovaraju rezultatima dobivenim u prethodnom primjeru, osim što je CLONALG pokazao bolje rezultate u usporedbi s drugim algoritmima u odnosu na prethodni primjer. Konačna vrijednost pogreške za genetsku varijantu algoritma Hooke-Jeeves (GHJ) malo je veća u odnosu na prethodni primjer zbog veće složenosti funkcije, a konačne vrijednosti pogreške za ostale algoritme su manje jer se izlaz funkcije za korištene primjere za učenje nalazi u intervalu $[0, 4]$, koji je znatno uži od raspona vrijednosti izlaza u prethodnom primjeru.

Tablica 5.4: Srednja kvadratna pogreška na evaluacijskom skupu podataka po iteracijama za korištene algoritme učenja u problemu regresije funkcije f2

Iteracija	*Backprop*	ES	GAn	SST	CLONALG	GHJ
10	0.292788	0.179095	0.180684	0.179017	0.177534	**0.118894**
50	0.171790	0.167780	0.178733	0.171053	0.149383	**0.018977**
100	0.144692	0.154417	0.174115	0.160124	0.107876	**0.015238**
500	0.033492	0.080393	0.137486	0.076956	0.041709	**0.013334**
1000	0.025974	0.049475	0.117348	0.050052	0.032141	**0.012983**
2500	0.018634	0.031624	0.080961	0.035170	0.030541	-
5000	0.017456	0.024647	0.058296	0.029125	0.030036	-
7500	0.017427	0.021896	0.048789	0.026909	0.029660	-

Tablica 5.5: Usporedba vremena učenja za korištene algoritme u problemu regresije funkcije f2. Učenje je provedeno kroz 1000 iteracija za algoritam GHJ te kroz 7500 iteracija za ostale algoritme.

Algoritam	$\bar{T}$ (s)	σ_T
Backpropagation	53.565	1.308
Evolucijska strategija	165.004	5.116
Genetic Annealing	181.811	56.875
Steady State Tournament	178.663	52.151
CLONALG	464.310	51.989
GHJ	260.508	7.587

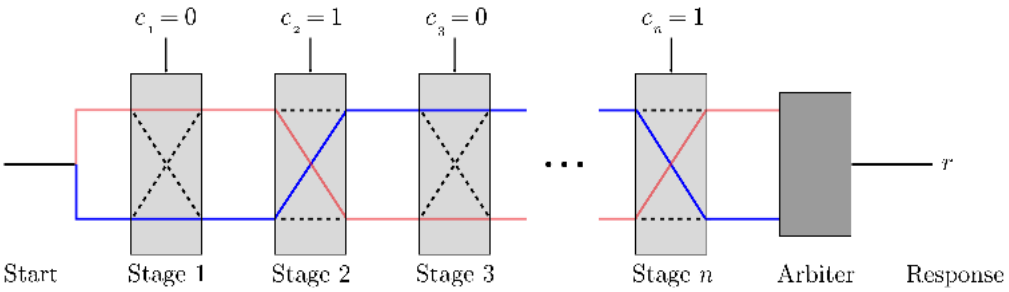


Slika 5.3: Prikaz kretanja srednje kvadratne pogreške na evaluacijskom skupu primjera do 1000. iteracije u problemu regresije funkcije f2 za korištene algoritme

5.2. Predikcija odziva PUF-a

5.2.1. *Arbiter* PUF

PUF (engl. *Physical Unclonable Function*) je fizički ostvarena funkcija koja iskorištava razlike koje nastaju u proizvodnji fizičkih objekata od kojih je sastavljena, čime je svako fizičko ostvarenje funkcije jedinstveno. Zbog jedinstvenih karakteristika svakog ostvarenja, odnosno, zbog nemogućnosti izrade dvije u potpunosti jednake instance PUF-a, PUF-ovi imaju veliku mogućnost primjene u kriptografiji. *Arbiter* PUF ostvaruje se slanjem dva signala kroz n razina kašnjenja (engl. *delay stage*), prema modelu prikazanom na slici 5.4. Svaka razina kašnjenja ostvarena je pomoću dva dvobitna multipleksora identične strukture pri čemu oba multipleksora koriste oba signala kao ulaze. Zahvaljujući inherentnim razlikama u kašnjenju svakog multipleksora, kašnjenje signala prolaskom kroz multipleksore bit će različito. Svaka razina kašnjenja na upravljačke ulaze multipleksora prima *challenge* bit c_i koji, ukoliko je postavljen, mijenja redoslijed gornjeg i donjeg signala, čime signali za n razina kašnjenja ukupno mogu proći kroz 2^n različitih puteva. U zadnjem sloju *Arbiter* PUF-a nalazi se *arbiter* (sudac) koji određuje koji je signal prije stigao do izlaza te na temelju toga postavlja izlaz PUF-a. Ukoliko je gornji signal bio brži, izlaz funkcije (engl. *response*) bit će „1“, te ukoliko je donji signal bio brži, izlaz će biti postavljen na vrijednost „0“. (Becker, 2015)



Slika 5.4: Model *Arbiter* PUF-a (Becker, 2015)

Napad na *Arbiter* PUF metodama strojnog učenja provodi se nadziranim učenjem odziva PUF-a za dane *challenge* bitove te je s ciljem usporedbe rezultata učenja proveden na generiranim primjerima za *Arbiter* PUF sa 64 razine kašnjenja, pri čemu su vrijednosti bitova *true* i *false* predstavljene vrijednostima „1“ i „-1“. Učenje je provedeno korištenjem istih algoritama kao i u problemu regresije nad 128, 512, 2048 te 8192 *challenge-response* parova s ciljem otkrivanja dovoljnog broja parova za uspješnu predikciju odziva PUF-a. Korišteni model sastojao se od tri potpuno povezana sloja sastavljenih redom od 15, 10 te jednog neurona sa aktivacijskom funkcijom *ReLU* između prvog i drugog te drugog i trećeg sloja, te aktivacijskom funkcijom tangens hiperbolni na izlazu iz modela. Kao funkcija gubitka korišten je srednji kvadratni gubitak između stvarne vrijednosti odziva i dobivenog odziva uz L_2 regularizaciju te se model sastojao od ukupno 1146 parametara. Evaluacija naučenog modela provedena je na skupu podataka koji se sastojao od 100000 *challenge-response* parova te je prilikom evaluacije pozitivna vrijednost izlaza modela predstavljala odziv „1“, a negativna vrijednost izlaza (uključivo s nulom) odziv „-1“. Učenje *Backpropagation* algoritmom provedeno je uz korekciju parametara modela Adam algoritmom uz stopu učenja 0.005. *Steady State Tournament* (SST) korišten je sa veličinom turnira 3 i populacijom veličine 30, *Genetic Annealing* (GAn) s elitizmom uz početnu vrijednost *demona* 25, faktorom hlađenja 0.8 te populacijom od 30 jedinki. Evolucijska strategija (ES) korištena je s dvije podpopulacije sa 15 roditelja od kojih se stvara 15 potomaka mutacijom. Učenje svim korištenim algoritmima provedeno u je 5000 iteracija bez podjele skupova podataka na mini-grupe. Početne vrijednosti parametara generirane su po uniformnoj razdiobi u intervalu $[-0.5, 0.5]$. Vjerojatnost mutacije u svim je populacijskim algoritmima iznosila 0.5 te je prvi operator mutacije generirao nove vrijednosti težina po uniformnoj razdiobi u intervalu $[-1, 1]$, a drugi operator mutacije je vrijednost korekcije težina generirao po normalnoj razdiobi sa srednjom vrijednosti 0 i standardnom devijacijom

0.5. Zbog korištenja vrlo malog broja primjera za učenje, korištena je viša stopa regularizacije, 0.01. Prilikom učenja, svakih 10 iteracija spremane su trenutne vrijednosti parametara modela za najbolju jedinku u populaciji kako bi se mogla prikazati promjena točnosti predikcija po iteracijama na evaluacijskom skupu podataka.

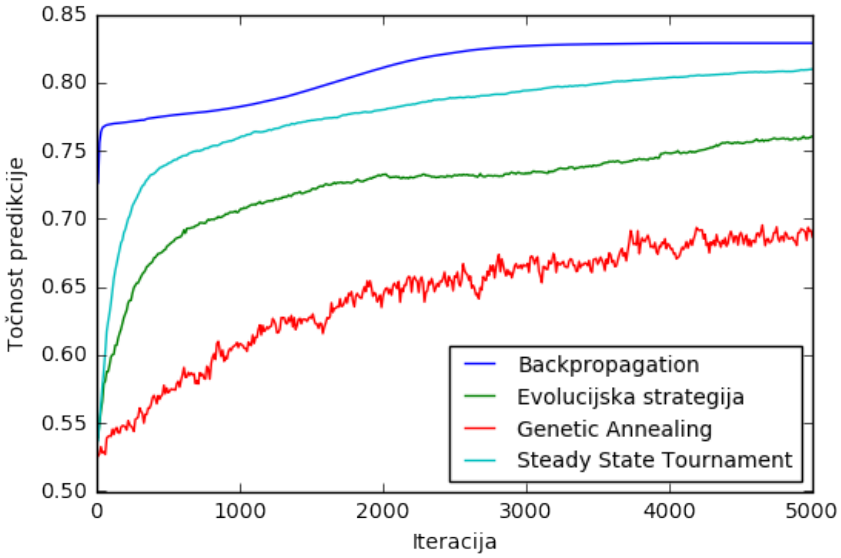
U tablicama 5.6, 5.8, 5.10 i 5.12 te na slikama 5.5, 5.6, 5.7 i 5.8 prikazano je kretanje pogreške na evaluacijskom skupu podataka po iteracijama učenja koristeći redom 128, 512, 2048 i 8192 primjera za učenje. Iz dobivenih podataka vidi se da učenje *Backpropagation* algoritmom znatno brže konvergira od ostalih algoritama — u vrlo malom broju iteracija postiže visoku točnost te ulazi u zasićenje. Ostali algoritmi (s izuzetkom *Genetic Annealing* algoritma) također su pokazali dobre rezultate te je njima moguće postići gotovo jednaku točnost kao i algoritmom *Backpropagation*, ali uz znatno sporiju konvergenciju i dulje vrijeme izvođenja. Vrijeme izvođenja 5000 iteracija učenja modela na 128, 512, 2048 i 8192 primjera za učenje redom je prikazano u tablicama 5.7, 5.8, 5.10 i 5.12. Očekivano, s porastom broja primjera korištenih prilikom učenja uz grupno učenje, poraslo je i računsko vrijeme učenja. Budući da je uz 128 primjera za učenje postignuta točnost veća od 80% u predikciji te da je uz 512 primjera postignuta točnost veća od 90%, iz konačnih rezultata može se zaključiti da su *Arbiter* PUF-ovi u potpunosti nesigurni na napade metodama strojnog učenja jer je potreban vrlo mali broj *challenge-response* parova kako bi se mogla dobiti dobra predikcija odziva.

Tablica 5.6: Točnost predikcije odziva PUF-a na evaluacijskom skupu podataka za model naučen na 128 primjera

Iteracija	*Backpropagation*	ES	GAn	SST
10	**0.726523**	0.539593	0.525833	0.537201
50	**0.767616**	0.579080	0.528793	0.588755
100	**0.769511**	0.598707	0.540293	0.638408
500	**0.775593**	0.680368	0.572383	0.740687
1000	**0.782211**	0.706552	0.608417	0.759859
2000	**0.810835**	0.732575	0.642911	0.780075
3000	**0.826822**	0.733731	0.665540	0.793986
4000	**0.828764**	0.748256	0.680864	0.803599
5000	**0.828934**	0.760551	0.686842	0.809805

Tablica 5.7: Vrijeme izvođenja 5000 iteracija učenja predikcije odziva PUF-a na 128 primjera

Algoritam	$\bar{T}$ (s)	σ_T
Backpropagation	36.044	0.952
Evolucijska strategija	138.819	0.744
Genetic Annealing	134.739	1.488
Steady State Tournament	145.052	1.538



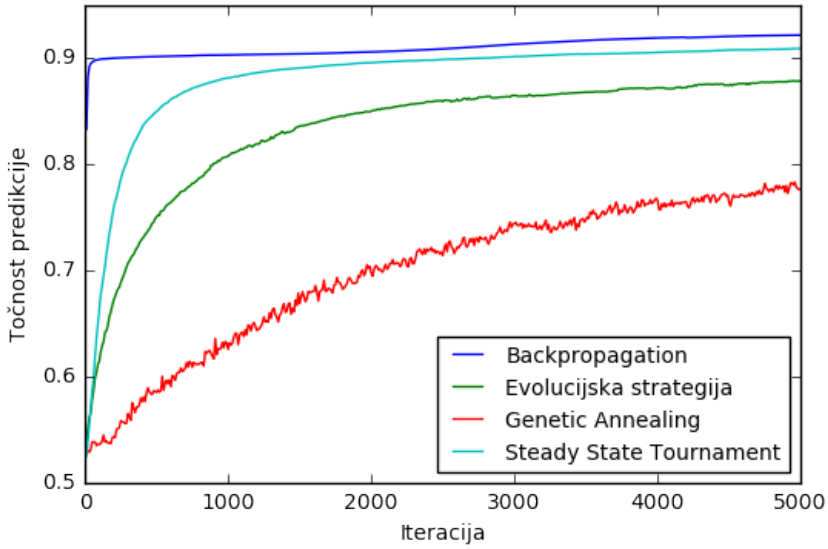
Slika 5.5: Prikaz kretanja točnosti predikcije odziva PUF-a na evaluacijskom skupu primjera po iteracijama učenja za modele naučene na 128 primjera

Tablica 5.8: Točnost predikcije odziva PUF-a na evaluacijskom skupu podataka za model naučen na 512 primjera

Iteracija	*Backpropagation*	ES	GAn	SST
10	**0.832632**	0.539510	0.529750	0.524412
50	**0.895782**	0.577462	0.537857	0.586293
100	**0.898244**	0.614098	0.535697	0.661988
500	**0.901048**	0.749989	0.584624	0.849256
1000	**0.902599**	0.807319	0.625254	0.881063
2000	**0.905433**	0.849551	0.697048	0.895248
3000	**0.912656**	0.864365	0.743613	0.901084
4000	**0.918679**	0.871696	0.765929	0.905128
5000	**0.921279**	0.877860	0.775462	0.908660

Tablica 5.9: Vrijeme izvođenja 5000 iteracija učenja predikcije odziva PUF-a na 512 primjera

Algoritam	$\bar{T}$ (s)	σ_T
Backpropagation	37.409	1.278
Evolucijska strategija	152.734	6.816
Genetic Annealing	140.375	1.917
Steady State Tournament	150.990	1.475



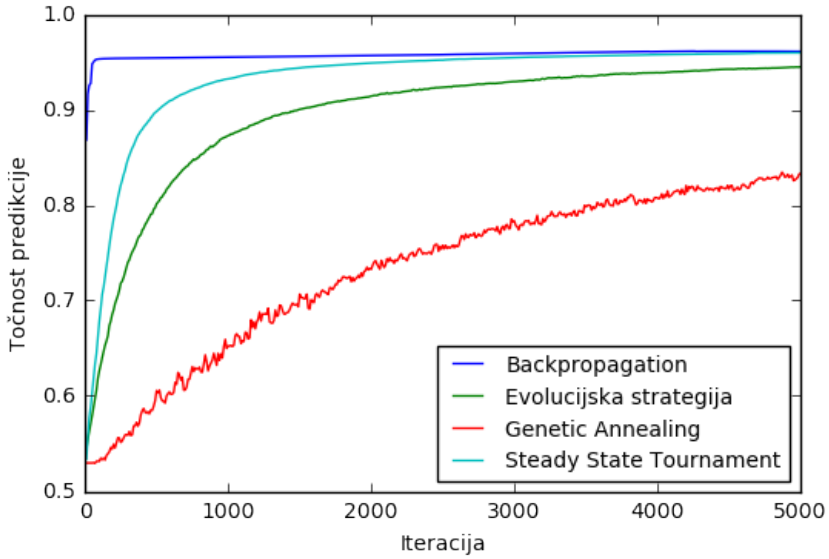
Slika 5.6: Prikaz kretanja točnosti predikcije odziva PUF-a na evaluacijskom skupu primjera po iteracijama učenja za modele naučene na 512 primjera

Tablica 5.10: Točnost predikcije odziva PUF-a na evaluacijskom skupu podataka za model naučen na 2048 primjera

Iteracija	*Backpropagation*	ES	GAn	SST
10	**0.868522**	0.543925	0.529748	0.533919
50	**0.947987**	0.579342	0.529748	0.603451
100	**0.953264**	0.626257	0.531488	0.678242
500	**0.954393**	0.801304	0.605472	0.898496
1000	**0.955222**	0.872895	0.652278	0.932308
2000	**0.956816**	0.914400	0.732753	0.949044
3000	**0.958986**	0.929534	0.783305	0.954781
4000	**0.961028**	0.938795	0.807453	0.957841
5000	**0.961181**	0.944929	0.833335	0.959698

Tablica 5.11: Vrijeme izvođenja 5000 iteracija učenja predikcije odziva PUF-a na 2048 primjera

Algoritam	$\bar{T}$ (s)	σ_T
Backpropagation	42.200	4.217
Evolucijska strategija	160.362	1.385
Genetic Annealing	156.317	0.991
Steady State Tournament	163.223	2.210



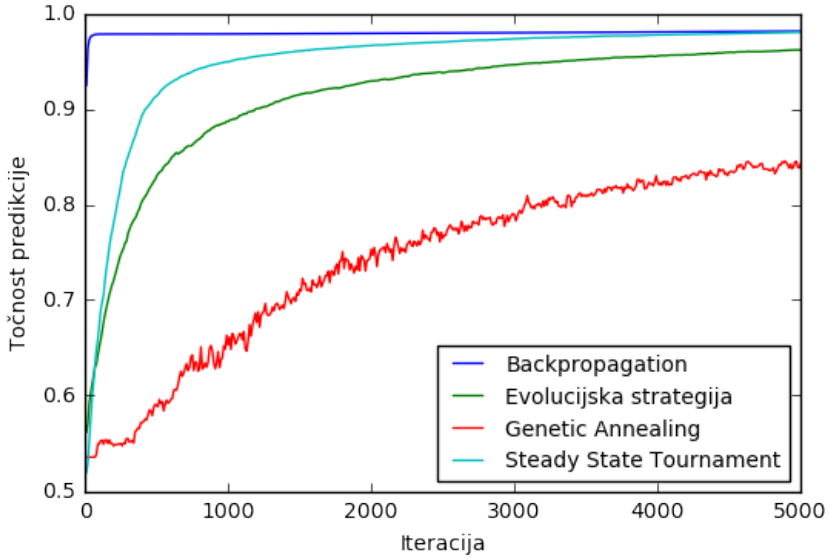
Slika 5.7: Prikaz kretanja točnosti predikcije odziva PUF-a na evaluacijskom skupu primjera po iteracijama učenja za modele naučene na 2048 primjera

Tablica 5.12: Točnost predikcije odziva PUF-a na evaluacijskom skupu podataka za model naučen na 8192 primjera

Iteracija	*Backpropagation*	ES	GAn	SST
10	**0.924954**	0.561778	0.535644	0.530104
50	**0.976800**	0.611698	0.535644	0.613974
100	**0.978290**	0.651431	0.549391	0.689304
500	**0.978371**	0.829661	0.589856	0.914966
1000	**0.978517**	0.886992	0.653095	0.949956
2000	**0.979061**	0.929242	0.753085	0.966619
3000	**0.979691**	0.946465	0.789302	0.973444
4000	**0.980558**	0.955426	0.823890	0.977448
5000	**0.981337**	0.961932	0.842695	0.980139

Tablica 5.13: Vrijeme izvođenja 5000 iteracija učenja predikcije odziva PUF-a na 8192 primjera

Algoritam	$\bar{T}$ (s)	σ_T
Backpropagation	60.036	1.631
Evolucijska strategija	242.557	2.200
Genetic Annealing	235.744	2.674
Steady State Tournament	245.220	1.954



Slika 5.8: Prikaz kretanja točnosti predikcije odziva PUF-a na evaluacijskom skupu primjera po iteracijama učenja za modele naučene na 8192 primjera

Utjecaj veličine mini-grupa na ishod učenja

S ciljem otkrivanja utjecaja veličine mini-grupa na ishod učenja, provedeno je ispitivanje na algoritmima *Steady State Tournament* i *Backpropagation*. Kao skup podataka za učenje korišten je skup 8192 *challenge-response* parova za učenje predikcije odziva PUF-a podijeljen u mini-grupe veličina 128, 512 i 2048. Mini-grupe stvaraju se prije početka učenja bez ponavljanja primjera, nakon slučajne permutacije cijelog skupa podataka za učenje. Tijekom učenja, pogreška modela računa se na temelju primjera iz samo jedne mini-grupe koje se izmjenjuju cirkularno kroz iteracije učenja. Ispitivanje je provedeno koristeći istu arhitekturu modela te iste parametre algoritama kao i u prethodnim primjerima.

U tablici 5.14 te na slici 5.9 prikazano je kretanje točnosti predikcije na evaluacijskom skupu po iteracijama učenja u ovisnosti o veličini mini-grupa za *Backpropagation* algoritam te u tablici 5.15 i na slici 5.10 za *Steady State Tournament* algoritam. Iz dobivenih podataka vidi se da manje veličine grupa imaju izrazito jak negativan učinak na uspješnost učenja kod algoritma *Steady State Tournament*, koji nije izražen kod algoritma *Backpropagation*. Standardna pogreška srednje vrijednosti prilikom uzorkovanja pada sublinearno s brojem korištenih uzoraka (obrnuto je proporcionalna korijenu broja uzoraka), stoga manji broj primjera za učenje nema velik utjecaj na procjenu pogreške i gradijenta, što je vidljivo kod oba algoritma u usporedbi korištenja 8192 parova (cijelog skupa za učenje) i korištenja mini-grupa od 2048 parova. Me-

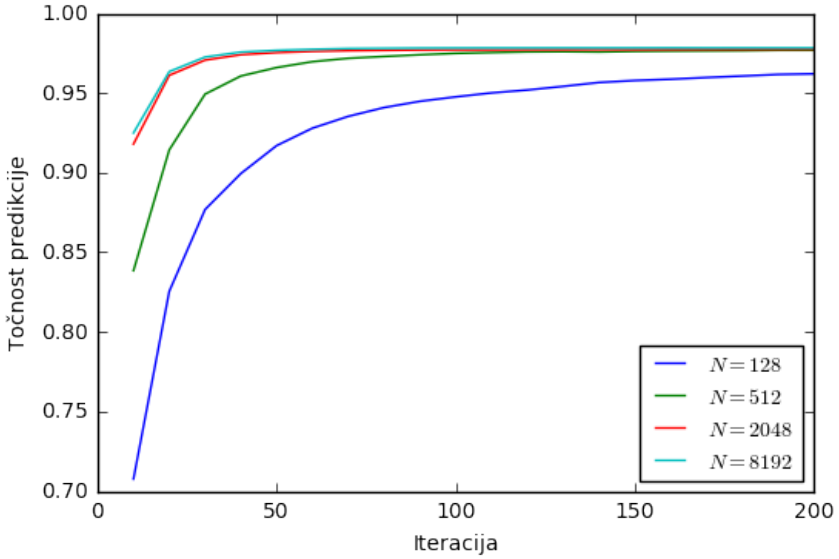
đutim, zbog vrlo spore konvergencije učenja algoritmom *Steady State Tournament* u usporedbi s algoritmom *Backpropagation*, i mali pad brzine konvergencije zbog malo manje preciznosti procjene pogreške ima vidljiv učinak.

Tablica 5.14: Točnost predikcije odziva PUF-a na evaluacijskom skupu podataka za model naučen algoritmom *Backpropagation* uz različite veličine mini-grupa

	Veličina mini-grupe		
Iteracija	128	512	2048
10	0.707637	0.838469	0.918000
50	0.917004	0.966012	0.975413
100	0.947597	0.974942	0.977161
500	0.969846	0.977869	0.977953
1000	0.971992	0.978208	0.978278
2000	0.974255	0.980066	0.979365
3000	0.975171	0.981588	0.980493
4000	0.976520	0.982329	0.981104
5000	0.976735	0.983093	0.981164

5.2.2. XOR PUF

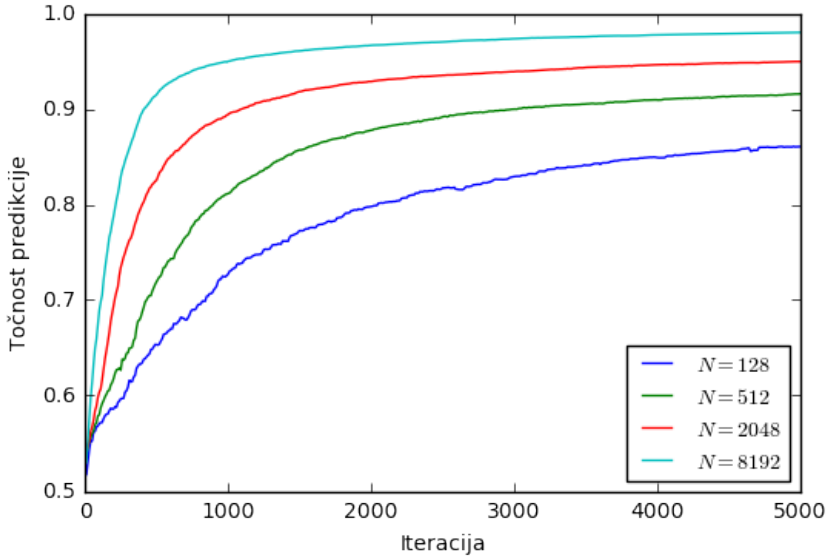
Dodavanjem nelinearnih elemenata u arhitekturu PUF-a može se postići veća otpornost na napade strojnim učenjem, što se uobičajeno provodi korištenjem više *Arbiter* PUF-ova nad čijim se izlazima provodi logička operacija isključivo ili (XOR). Svaki *Arbiter* PUF na upravljačke ulaze prima isti skup *challenge* bitova te se operacijom isključivo ili njihovi izlazi kombiniraju u jedan izlaz. Zbog povećanja kompleksnosti problema uvođenjem nelinearnosti, očekivano je da će za uspješno učenje i točnu predikciju biti potreban znatno veći broj *challenge-response* parova u odnosu na probleme predikcije odziva *Arbiter* PUF-a. S ciljem usporedbe rezultata učenja različitim algoritmima, provedeno je učenje odziva XOR PUF-a sastavljenog od 4 *Arbiter* PUF-a sa 64 razine kašnjenja na modelu s istom arhitekturom te istim algoritmima kao i u prethodnim primjerima. Korištenje 16384 *challenge-response* parova nije se pokazalo dovoljnim prilikom učenja odziva jer se algoritmom *Backpropagation* u najboljem slučaju mogla ostvariti točnost od 60% na evaluacijskom skupu. Međutim, povećanjem skupa podataka za učenje na 24576 *challenge-response* parova ostvareni su znatno bolji rezultati. Parametri algoritama bili su jednaki kao i u prethodnim primjerima, osim stope



Slika 5.9: Prikaz kretanja točnosti predikcije odziva PUF-a na evaluacijskom skupu primjera do 200. iteracije učenja za modele naučene algoritmom *Backpropagation* uz veličinu mini-grupe N

Tablica 5.15: Točnost predikcije odziva PUF-a na evaluacijskom skupu podataka za model naučen algoritmom *Steady State Tournament* uz različite veličine mini-grupa

	Veličina mini-grupe		
Iteracija	128	512	2048
10	0.517556	0.538690	0.542165
50	0.552579	0.560328	0.567331
100	0.570728	0.580126	0.603644
500	0.653284	0.719258	0.825394
1000	0.727154	0.811464	0.894205
2000	0.797964	0.877827	0.928753
3000	0.829252	0.899905	0.939167
4000	0.849094	0.909311	0.946318
5000	0.860660	0.915852	0.949619



Slika 5.10: Prikaz kretanja točnosti predikcije odziva PUF-a na evaluacijskom skupu primjera po iteracijama učenja za modele naučene algoritmom *Steady State Tournament* uz veličinu mini-grupe N

učenja koja je smanjena na vrijednost 10^{-4}, stope regularizacije koja je smanjena na vrijednost 10^{-3} te korišteni skup podataka za učenje nije bio podijeljen u mini-grupe.

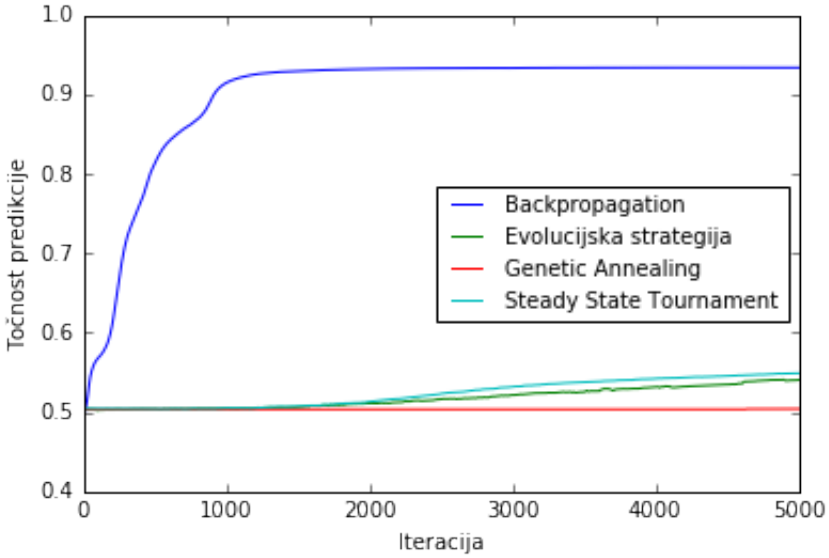
U tablici 5.16 te na slici 5.11 prikazano je kretanje točnosti predikcije odziva na skupu za evaluaciju koji se sastojao od 100000 primjera, a u tablici 5.17 prikazano je vrijeme izvođenja 5000 iteracija učenja za korištene algoritme. Iz dobivenih podataka može se primijetiti da korišteni populacijski algoritmi u samom početku učenja zapnu u vrlo lošem lokalnom minimumu te im je potreban vrlo velik broj iteracija za minimalno poboljšanje točnosti predikcije. Međutim, algoritam *Backpropagation* ostvaruje vrlo dobre rezultate predikcije s točnošću većom od 90% što potvrđuje da ni XOR PUF-ovi nisu otporni na napade strojnim učenjem, ali u odnosu na *Arbiter* PUF-ove, učenje njihovog odziva predstavlja znatno veći problem.

Tablica 5.16: Točnost predikcije odziva XOR PUF-a na evaluacijskom skupu podataka po iteracijama učenja

Iteracija	*Backpropagation*	ES	GAn	SST
10	**0.506684**	0.502961	0.503661	0.504593
50	**0.552273**	0.503342	0.503661	0.504591
100	**0.569336**	0.502645	0.503661	0.504591
500	**0.816632**	0.503899	0.503661	0.503987
1000	**0.915281**	0.505158	0.503661	0.505212
2000	**0.931813**	0.511287	0.503661	0.51364
5000	**0.933475**	0.540409	0.504187	0.549036

Tablica 5.17: Vrijeme izvođenja 5000 iteracija učenja predikcije odziva XOR PUF-a

Algoritam	$\bar{T}$ (s)	σ_T
Backpropagation	118.615	1.757
Evolucijska strategija	556.474	9.158
Genetic Annealing	532.530	1.077
Steady State Tournament	555.158	2.687



Slika 5.11: Prikaz kretanja točnosti predikcije odziva XOR PUF-a na evaluacijskom skupu primjera po iteracijama učenja

5.3. Klasifikacija slika

Klasifikacija slika u odnosu na prethodne probleme predstavlja znatno veći izazov u učenju — skupovi podataka znatno su kompleksniji i veći, potrebni su složeniji modeli te je potrebno vrijeme učenja znatno povećano u odnosu na prethodne probleme. Budući da su korišteni populacijski algoritmi u odnosu na *Backpropagation* algoritam pokazali znatno slabije performanse, očekivano je da će s povećanjem složenosti problema postići još slabije rezultate pri čemu potrebno vrlo dugo vrijeme učenja u potpunosti može onemogućiti njihovu primjenu. Primjerice, za klasifikaciju slika na skupu podataka MNIST, algoritam *Steady State Tournament* koji je u prethodnim problemima pokazao bolje rezultate od ostalih populacijskih algoritama, u 9000 iteracija učenja na relativno jednostavnom modelu (koji ima dovoljan kapacitet) ostvaruje tek otprilike 60% točnosti uz 40 minuta učenja, što u potpunosti onemogućuje njegovu samostalnu primjenu.

Novija istraživanja pokazala su da bi se hibridnim pristupom koji kombinira *Backpropagation* algoritam i druge metaheurističke algoritme mogli ostvariti bolji rezultati učenja u odnosu na samostalno korištenje gradijentnih postupaka (Rere et al., 2016), (Ayumi et al., 2016). Nakon korekcije parametara gradijentnim postupkom, nad dobivenim novim rješenjem može se provesti nekoliko iteracija učenja metaheurističkim algoritmom, čime metaheuristički algoritmi provode dodatno pretraživanje prostora rješenja. Rezultati primjene takvog hibridnog pristupa, koji *Backpropagation* algoritam kombinira s prethodno korištenim metaheurističkim algoritmima prikazani su u nastavku na dva vrlo često korištena skupa podataka za učenje — MNIST i CIFAR-10.

5.3.1. MNIST

Skup podataka MNIST[1] sastoji se od crno-bijelih slika rukom pisanih dekadskih znamenaka veličine 28×28 piksela, podijeljenih u skup za učenje koji sadrži 60000 primjera te skup za evaluaciju koji sadrži 10000 primjera. Trenutno najbolji modeli na ovom skupu podataka postižu točnost veću od 99.7% te se problem klasifikacije slika sadržanih u ovom skupu smatra lakim zadatkom. Međutim, zbog vrlo velikog broja dostupnih rezultata, na ovom je skupu vrlo lako provesti usporedbu različitih algoritama i modela učenja.

Učenje je provedeno na modelu koji se sastojao od dva konvolucijska sloja s veličinom konvolucijske jezgre 5×5 sa 16 i 32 konvolucijska filtra, uz aktivaciju zglobnicom

[1] `http://yann.lecun.com/exdb/mnist/`

i sažimanje maksimumom s prozorom veličine 2×2 uz pomak 2. Nakon konvolucijskih slojeva, u modelu su se nalazila 3 potpuno povezana sloja sa 120, 64 te 10 neurona uz aktivaciju zglobnicom između prvog i drugog te drugog i trećeg potpuno povezanog sloja, te funkcijom *softmax* na izlazu iz modela. Kao funkcija gubitka, korišten je gubitak unakrsne entropije uz L_2 regularizaciju, a model se ukupno sastojao od 209922 parametra. Učenje je provedeno hibridnim pristupom sa algoritmom *Backpropagation* uz korekciju težina Adam algoritmom sa stopom učenja 0.0005 te jednom iteracijom metaheurističkog algoritma za finije ugađanje vrijednosti parametara modela. *Genetic Annealing* algoritam korišten je nad samo jednom jedinkom sa početnom vrijednosti *demona* 5 i faktorom hlađenja 0.8. U populacijskim algoritmima, korekcija parametara *Backpropagation* / Adam algoritmom provođena je samo nad najboljom jedinkom u trentnoj iteraciji. Evolucijska strategija korištena je s populacijom od 10 roditelja od kojih se mutacijom stvara 10 potomaka, a *Steady State Tournament* korišten je s populacijom od 10 jedinki te veličinom turnira 3. Svi parametri križanja i mutacije bili su isti kao i u primjeru s učenjem predikcije odziva PUF-a. Početne vrijednosti rješenja generirane su po normalnoj razdiobi sa srednjom vrijednosti 0 i standardnim odstupanjem 0.1, uz ponovno generiranje vrijednosti ukoliko ona odstupa od srednje vrijednosti za više od dvije vrijednosti standardnog odstupanja (engl. *truncated normal distribution*). Učenje je provedeno uz mini-grupe od 100 primjera i stopu regularizacije 0.0001 kroz 12000 iteracija. Prije početka učenja, vrijednosti intenziteta piksela u skupu za učenje svedene su na normalnu razdiobu na temelju srednje vrijednosti i odstupanja na cijelom skupu za učenje. Prilikom učenja, svakih 300 iteracija spremane su trenutne vrijednosti parametara za najbolju jedinku u populaciji kako bi se moglo prikazati kretanje točnosti klasifikacije kroz iteracije učenja.

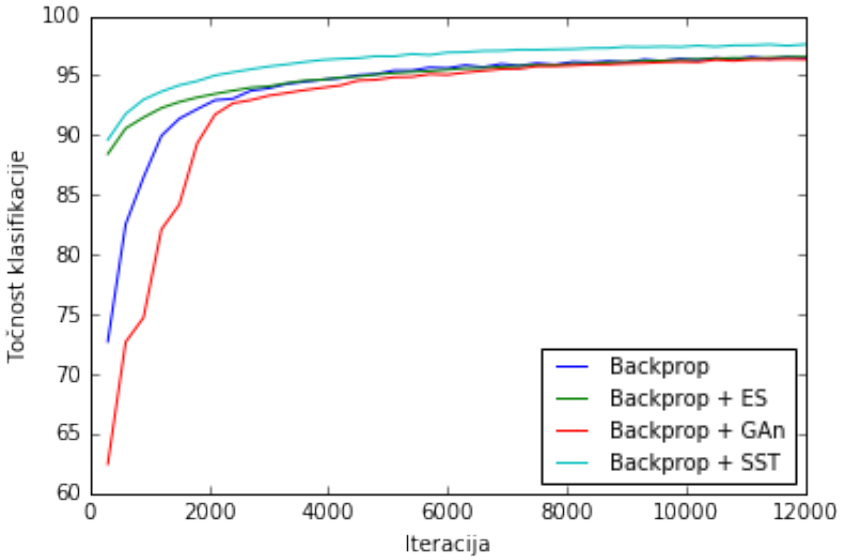
U tablici 5.18 te na slici 5.12 prikazano je kretanje točnosti klasifikacije kroz iteracije učenja, a u tablici 5.19 prikazano je vrijeme izvođenja 12000 iteracija učenja za korištene algoritme. Iz dobivenih podataka vidi se da hibridni pristup s algoritmom *Steady State Tournament* ostvaruje najbolje rezultate klasifikacije kroz sve iteracije učenja, pogotovo u usporedbi s samostalnim algoritmom *Backpropagation* tijekom prve epohe učenja (prvih 600 iteracija). Međutim, vrijeme potrebno za učenje *Steady State Tournament* algoritmom značajno odstupa od vremena izvođenja ostalih algoritama te razlika konačne točnosti klasifikacije algoritama nije velika, stoga će se tek na složenijim problemima moći bolje uočiti razlika u konačnoj točnosti klasifikacije te odrediti koliko veliki utjecaj može imati dulje vrijeme izvođenja.

Tablica 5.18: Točnost klasifikacije slika (MNIST) na evaluacijskom skupu podataka kroz iteracije učenja

	Backprop		Backprop+ES		Backprop+GAn		Backprop+SST	
Iteracija	A (%)	σ	A (%)	σ	A (%)	σ	A (%)	σ
300	72.72	23.04	88.43	1.89	62.47	32.21	**89.62**	1.52
600	82.55	15.18	90.58	1.52	72.69	32.51	**91.79**	1.39
1200	89.95	4.99	92.28	1.15	82.11	25.19	**93.65**	0.88
2400	93.06	2.56	93.73	0.91	92.68	2.27	**95.27**	0.71
3600	94.48	1.62	94.57	0.83	93.78	1.60	**96.13**	0.57
4800	95.12	1.53	95.09	0.75	94.64	1.20	**96.62**	0.37
6000	95.62	1.37	95.49	0.70	95.06	1.15	**96.93**	0.41
8400	96.10	1.17	95.99	0.66	95.93	0.98	**97.29**	0.32
12000	96.44	1.21	96.57	0.46	96.35	1.04	**97.61**	0.39

Tablica 5.19: Vrijeme izvođenja 12000 iteracija učenja klasifikacije slika (MNIST)

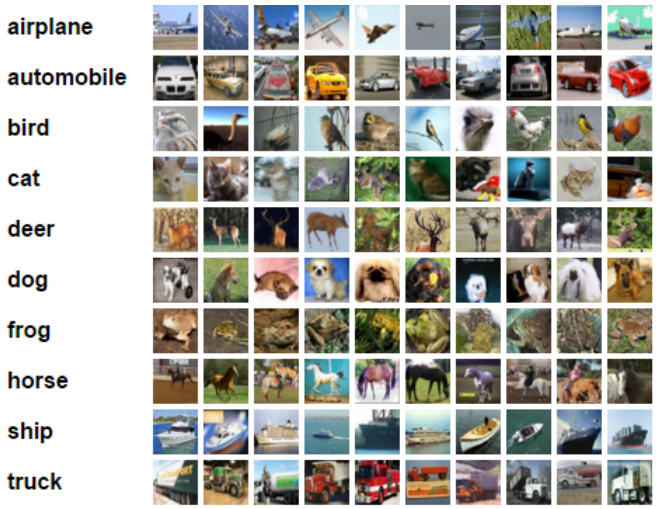
Algoritam	$\bar{T}$ (s)	σ_T
Backprop	191.391	3.747
Backprop + ES	665.306	5.064
Backprop + GAn	242.136	3.959
Backprop + SST	1553.394	15.465



Slika 5.12: Prikaz kretanja točnosti klasifikacije (MNIST) na evaluacijskom skupu podataka kroz iteracije učenja

5.3.2. CIFAR-10

Skup podataka CIFAR-10[2] sastoji se od 60000 slika u boji veličine 32×32 piksela podijeljenih u 10 razreda s 50000 primjera za učenje te 10000 primjera za evaluaciju. Primjeri slika po razredima prikazani su na slici 5.13.



Slika 5.13: Primjeri slika iz skupa podataka CIFAR-10

Problem klasifikacije slika iz ovog skupa podataka smatra se znatno složenijim problemom u odnosu na skup podataka MNIST te najbolji modeli na ovom skupu po-

[2] https://www.cs.toronto.edu/~kriz/cifar.html

dataka ostvaruju točnost od 96.53% (Graham, 2014). Koristeći modele slične modelu korištenom u prethodnom problemu moguće je ostvariti točnost od otprilike 70% kroz vrlo velik broj iteracija u usporedbi s drugim problemima čiji su rezultati prikazani u ovom poglavlju.

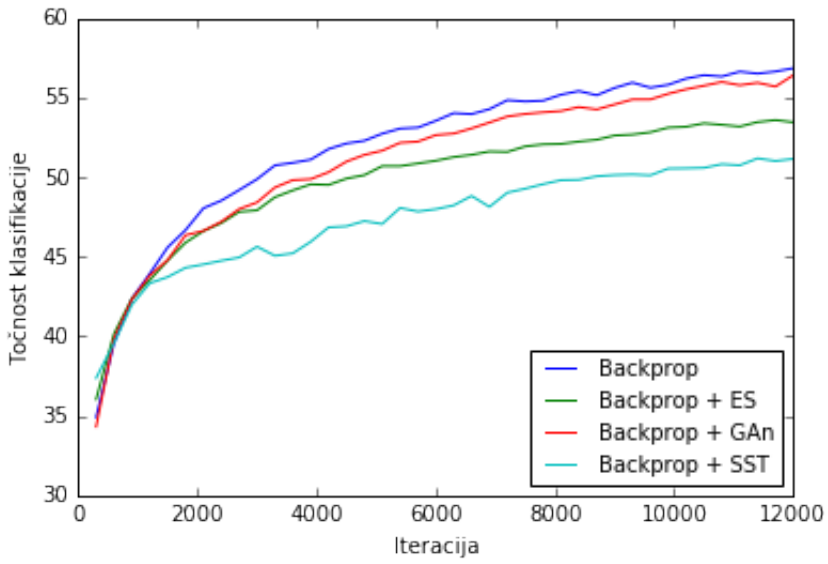
Učenje je provedeno na modelu s arhitekturom vrlo sličnoj arhitekturi modela korištenoj u klasifikaciji slika iz skupa podataka MNIST. Model se sastojao od dva konvolucijska sloja s veličinom konvolucijske jezgre 5×5 sa 16 i 32 konvolucijska filtra, uz aktivaciju zglobnicom te sažimanja maksimumom s prozorom veličine 2×2 i pomakom 2. Nakon konvolucijskih slojeva, u modelu se nalazio potpuno povezani sloj s 512 neurona te izlazni sloj s 10 neurona uz aktivaciju zglobnicom između slojeva te funkcijom *softmax* na izlazu iz modela. Korištena je ista funkcija gubitka kao i u prethodnom primjeru (gubitak unakrsne entropije s L_2 regularizacijom težina). Model se ukupno sastojao od 1068266 parametra. Parametri algoritama bili su isti kao i u prethodnom primjeru, osim stope učenja koja je smanjena na vrijednost 10^{-4}, stope regularizacije koja je povećana na vrijednost 0.05 te veličine mini-grupa koja je smanjena na 50 primjera. Učenje je provedeno kroz 12000 iteracija te je u tablici 5.20 i na slici 5.14 prikazano kretanje točnosti na evaluacijskom skupu podataka kroz iteracije algoritma te je potrebno vrijeme izvođenja 12000 iteracija učenja prikazano u tablici 5.21. Prikazani podaci dobiveni su na temelju samo 5 mjerenja zbog vrlo dugog vremena izvođenja. Dobivena točnost klasifikacije za sve algoritme je loša, pogotovo kada se uzme u obzir potrebno vrijeme učenja za relativno mali broj iteracija, posebno za kombinaciju s algoritmom *Steady State Tournament* za koju je vrijeme učenja kroz 12000 iteracija bilo u prosjeku dulje od 1.5 sati. Uz veći broj iteracija i bolje ugađanje parametara algoritama i modela mogli bi se ostvariti bolji rezultati, međutim, s obzirom na potrebno vrijeme učenja, iz dobivenih podataka se može zaključiti da hibridni pristup korištenja algoritma *Backpropagation* s metaheurističkim algoritmima nije primjenjiv u praksi.

Tablica 5.20: Točnost klasifikacije slika (CIFAR-10) na evaluacijskom skupu podataka kroz iteracije učenja

	Backprop		Backprop+ES		Backprop+GAn		Backprop+SST	
Iteracija	A (%)	σ	A (%)	σ	A (%)	σ	A (%)	σ
300	34.92	2.08	36.04	3.72	34.32	2.17	**37.38**	1.69
600	39.56	1.65	**40.15**	3.66	39.86	1.01	39.57	2.01
1200	**43.93**	1.64	43.53	3.43	43.82	1.11	43.33	0.97
2400	**48.52**	1.43	47.11	2.64	47.20	1.34	44.74	1.12
4800	**52.28**	1.23	50.12	1.67	51.39	0.84	47.24	0.75
6000	**53.54**	1.40	51.03	1.30	52.64	0.64	47.99	0.85
8400	**55.40**	0.49	52.23	1.31	54.40	0.73	49.83	1.15
12000	**56.84**	0.79	53.58	0.85	55.68	1.02	51.01	1.26

Tablica 5.21: Vrijeme izvođenja 12000 iteracija učenja klasifikacije slika (CIFAR-10)

Algoritam	$\bar{T}$ (s)	σ_T
Backprop	456.053	3.456
Backprop + ES	1884.249	6.213
Backprop + GAn	603.658	3.743
Backprop + SST	5606.254	33.912



Slika 5.14: Prikaz kretanja točnosti klasifikacije (CIFAR-10) na evaluacijskom skupu podataka kroz iteracije učenja

6. Zaključak

U ovom radu prikazani su rezultati učenja neuronskih mreža na različitim problemima koristeći različite algoritme. Rezultati su pokazali da je moguće uspješno riješiti probleme regresije te uspješno naučiti odzive *Arbiter* PUF-a sa svim korištenim algoritmima. Međutim, u problemima učenja odziva XOR PUF-a te klasifikaciji slika, zbog porasta složenosti problema, uspješnost učenja metaheurističkim algoritmima pada te njima nije moguće ostvariti dobre rezultate u razumnom vremenu.

Neuronske mreže su kroz svoju povijest doživjele brojne uspone i padove te su u posljednjih desetak godina zbog velikog napretka hardvera, pojavom velikih skupova podataka za učenje te pojavom novih tehnika učenja dobile brojne primjene u mnogim područjima. Metaheuristički algoritmi su kroz povijest pokazali uspješnost u učenju neuronskih mreža zahvaljujući boljim rezultatima u odnosu na gradijentne metode zbog problema nestajućeg gradijenta. Međutim, zbog novih tehnika učenja koje su uklonile problem nestajućeg gradijenta, gradijentnim metodama je moguće postići znatno bolje rezultate u učenju, prvenstveno zbog znatno manjeg potrebnog vremena za učenje u odnosu na metaheurističke algoritme.

LITERATURA

Vina Ayumi, L. M. Rasdi Rere, Mohamad Ivan Fanany, i Aniati Murni Arymurthy. Optimization of convolutional neural network using microcanonical annealing algorithm. *CoRR*, abs/1610.02306, 2016. URL http://arxiv.org/abs/1610.02306.

Georg T. Becker. *The Gap Between Promise and Reality: On the Insecurity of XOR Arbiter PUFs*, stranice 535–555. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015. ISBN 978-3-662-48324-4. doi: 10.1007/978-3-662-48324-4_27. URL http://dx.doi.org/10.1007/978-3-662-48324-4_27.

Michael Creutz. Microcanonical monte carlo simulation. *Phys. Rev. Lett.*, 50(19): 1411–1414, Svibanj 1983. doi: 10.1103/PhysRevLett.50.1411. URL http://link.aps.org/doi/10.1103/PhysRevLett.50.1411.

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

Benjamin Graham. Fractional max-pooling. *CoRR*, abs/1412.6071, 2014. URL http://arxiv.org/abs/1412.6071.

Diederik P. Kingma i Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL http://arxiv.org/abs/1412.6980.

Kenneth V. Price. Genetic annealing, 1994. URL http://www.drdobbs.com/database/genetic-annealing/184409333?pgno=10. *Dr. Dobbs - the world of software development*, 3.6.2017.

L. M. Rasdi Rere, Mohamad Ivan Fanany, i Aniati Murni Arymurthy. Metaheuristic algorithms for convolution neural network. *Intell. Neuroscience*, 2016:2–, Lipanj 2016. ISSN 1687-5265. doi: 10.1155/2016/1537325. URL https://doi.org/10.1155/2016/1537325.

David E. Rumelhart, Geoffrey E. Hinton, i Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, Listopad 1986. URL `http://dx.doi.org/10.1038/323533a0`.

El-Ghazali Talbi. *Metaheuristics: From Design to Implementation*. Wiley Publishing, 2009. ISBN 0470278587, 9780470278581.

Xin Yao. Optimization by genetic annealing. U *Proc. of Second Australian Conf. on Neural Networks*, stranice 94–97, 1991.

Primjena stohastičkih algoritama optimizacije u učenju dubokih neuronskih mreža

Sažetak

U ovom radu opisane su neuronske mreže kao tehnika strojnog učenja te različiti optimizacijski algoritmi primijenjeni u njihovom učenju. U sklopu rada implementirana je programska podrška za učenje neuronskih mreža koristeći programske okvire TensorFlow i ECF. U radu je prikazana usporedba rezultata učenja neuronskih mreža različitim algoritmima u problemima regresije, učenju predikcije odziva PUF-ova te klasifikaciji slika.

Ključne riječi: neuronske mreže, duboko učenje, optimizacijski algoritmi, PUF, TensorFlow, ECF

Application of Stochastic Optimization Algorithms in Deep Learning

Abstract

This paper describes neural networks as a machine learning technique and several optimization algorithms applied in deep learning. This paper also includes a description of software developed for deep learning, based on TensorFlow and ECF frameworks, and performance comparison between algorithms in regression tasks, PUF attacks and image classification.

Keywords: neural networks, deep learning, optimization algorithms, PUF, TensorFlow, ECF