SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1969

Optimizirana arhitektura klasifikatora temeljenog na umjetnim neuronskim mrežama u domeni implementacijskih napada na kriptografske uređaje

Juraj Juričić

Zagreb, lipanj 2019.

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

Zagreb, 8. ožujka 2019.

DIPLOMSKI ZADATAK br. 1969

Pristupnik: **Juraj Juričić (0036489629)**
Studij: Računarstvo
Profil: Računarska znanost

Zadatak: **Optimizirana arhitektura klasifikatora temeljenog na umjetnim neuronskim mrežama u domeni implementacijskih napada na kriptografske uređaje**

Opis zadatka:

Proučiti neuroevolucijske metode u izgradnji topologije umjetne neuronske mreže. Usporediti postojeće neuroevolucijske algoritme: načine prikaza umjetne neuronske mreže u evolucijskom algoritmu, evolucijski algoritam i korištene parametre. Uz pomoć odabrane neuroevolucijske metode izgraditi klasifikator kojim će se provesti ispitivanje na skupovima DPAv2 i DPAv4 te odrediti mjere kvalitete klasifikatora: točnost, preciznost, odziv te F mjere. Usporediti učinkovitost ostvarenih postupaka s postojećim rješenjima iz literature. Radu priložiti izvorne tekstove programa, dobivene rezultate uz potrebna objašnjenja i korištenu literaturu.

Zadatak uručen pristupniku: 15. ožujka 2019.
Rok za predaju rada: 28. lipnja 2019.

Mentor:

Prof. dr. sc. Domagoj Jakobović

Djelovođa:

Izv. prof. dr. sc. Tomislav Hrkać

Predsjednik odbora za diplomski rad profila:

Doc. dr. sc. Marko Čupić

Veliku zahvalnost dugujem svojem mentoru, prof. dr. sc. Domagoju Jakoboviću i voditelju diplomskog rada, mag. ing. Karlu Kneževiću, koji su mi svojim savjetima i idejama pomagali kroz cijeli diplomski studij, a posebice pri izradi ovog diplomskog rada.

Zahvaljujem svim svojim kolegama i prijateljima na svim ugodnim trenucima i podršci kroz cijeli studij, na društvu u svim neprospavanim ispitnim noćima[1] *te na svoj pruženoj pomoći, kako u studiju, tako i u životu. Posebno hvala upućujem Ivanu, Jeleni i Kim na lekturi i iscrpnim savjetima tijekom pisanja ovoga diplomskog rada.*

Veliko hvala upućujem i svojoj obitelji – majci Sabini, ocu Igoru, bratu Andriji, nonama Grazielli i Marselini, nonotu Sergiu te očuhu Damiru. Uvijek su mi bili nezamjenjiva podrška te bez njih ne bih bio gdje jesam, niti postigao sve što jesam.

I konačno, zahvaljujem i Vama, čitatelju ovoga rada, što ste odvojili vrijeme kako biste pročitali barem dio rada koji predstavlja sukus mojega cijelog dosadašnjeg obrazovanja.

Svima spomenutima od srca hvala!

[1] poput ove u kojoj dovršavam svoj diplomski rad

SADRŽAJ

1. Uvod

U kontekstu kriptografije, **implementacijski napadi** su kriptografski napadi koji iskorištavaju svojstva i slabosti implementacije kriptografskih algoritama u kriptografskom uređaju. Potpuno uspješan napad na kriptografski uređaj znači otkrivanje tajnoga ključa pohranjenog na uređaju.

Umjetne neuronske mreže predstavnici su konektivističkog pristupa umjetnoj inteligenciji koje svoje ideje temelje na mehanizmu rada živčanog sustava u sisavaca. Danas su vrlo rasprostranjene i među najkorištenijim porodicama modela u umjetnoj inteligenciji i strojnom učenju [15]. Koriste se za rješavanje velikog broja problema klasifikacije i regresija, a u ovom diplomskom radu korištene su kao klasifikatori očitanja potrošnje električne energije.

Jedan od glavnih hiperparametara slojevite neuronske mreže je njezina **arhitektura** – broj slojeva i broj neurona po sloju neuronske mreže. Dvije neuronske mreže mogu davati potpuno drugačije rezultate ako imaju različite arhitekture te je odabir dobre arhitekture ključan u izradi uspješno klasificirajuće neuronske mreže.

Evolucijsko računarstvo grana je računarstva koja se bavi proučavanjem i izradom algoritama zasnivanih na pristupima inspiriranim biološkom evolucijom i Darwinovom teorijom o postanku vrsta [9]. Evolucijsko računarstvo najčešće se koristi za rješavanje optimizacijskih problema, iako to ne mora uvijek biti slučaj [19].

U ovom diplomskom radu, pristupi evolucijskog računarstva korišteni su za pronalazak (optimizaciju) arhitekture slojevite neuronske mreže koja klasificira očitanja potrošnje električne energije, a s ciljem pronalaska tajnoga kriptografskog ključa. Postupak evolucije arhitekture umjetnih neuronskih mreža spada u granu zvanu **neuroevolucija**.

Korištena su dva pristupa. Algoritam *NEAT* [30] evoluira rijetko povezanu neuronsku mrežu, ali ne uspijeva postići veliku točnost pri klasifikaciji. S druge strane, prilikom korištenja dvije verzije[1] generacijskog genetskog algoritma koji evoluira arhitekturu slojevite neuronske mreže postižu se vrlo dobri rezultati.

Ovaj rad podijeljen je na 7 poglavlja, uključujući ovaj uvod te zaključak. U 2. poglavlju opisana je diferencijalna analiza potrošnje električne energije te osnove kriptografije potrebne za provođenje napada. U 3. poglavlju opisane su ideje iza umjetnih neuronskih

[1] sa segregacijom po broju slojeva te bez segregacije

mreža te njihova rada. 4. poglavlje opisuje korištene ideje evolucijskoga računarstva te korištene algoritme. Detalji implementacije, pseudokôdovi i korišteni hiperparametri opisani su u 5. poglavlju. Konačno, 6. poglavlje prikazuje rezultate eksperimenata provedenih s implementiranim algoritmima.

2. Opis napada zasnovanog na sporednim svojstvima uređaja

Kriptografski uređaj je elektronički uređaj na kojem se izvršava kriptografski algoritam [24]. Kod takvih uređaja, tajni ključ pohranjen je na samom uređaju, što dovodi do novog problema u sigurnosti. Time područje interesa za sigurnost u kriptografiji više nije samo otpornost algoritma na kriptoanalizu, već otpornost cijelog sustava u kojem se algoritam izvodi. Primjeri nekih kriptografskih uređaja su pametna kartica, *USB* uređaji i čipovi.

Konačan cilj kriptografskog napada je otkrivanje tajnoga ključa. Pokušaj neovlaštenog otkrivanja tajnoga ključa pohranjenog na uređaju naziva se **napad**, a osoba koja pokušava saznati tajni ključ **napadač**. Osim samoga tajnog ključa, ishod napada može biti i neka informacija o tajnom ključu.

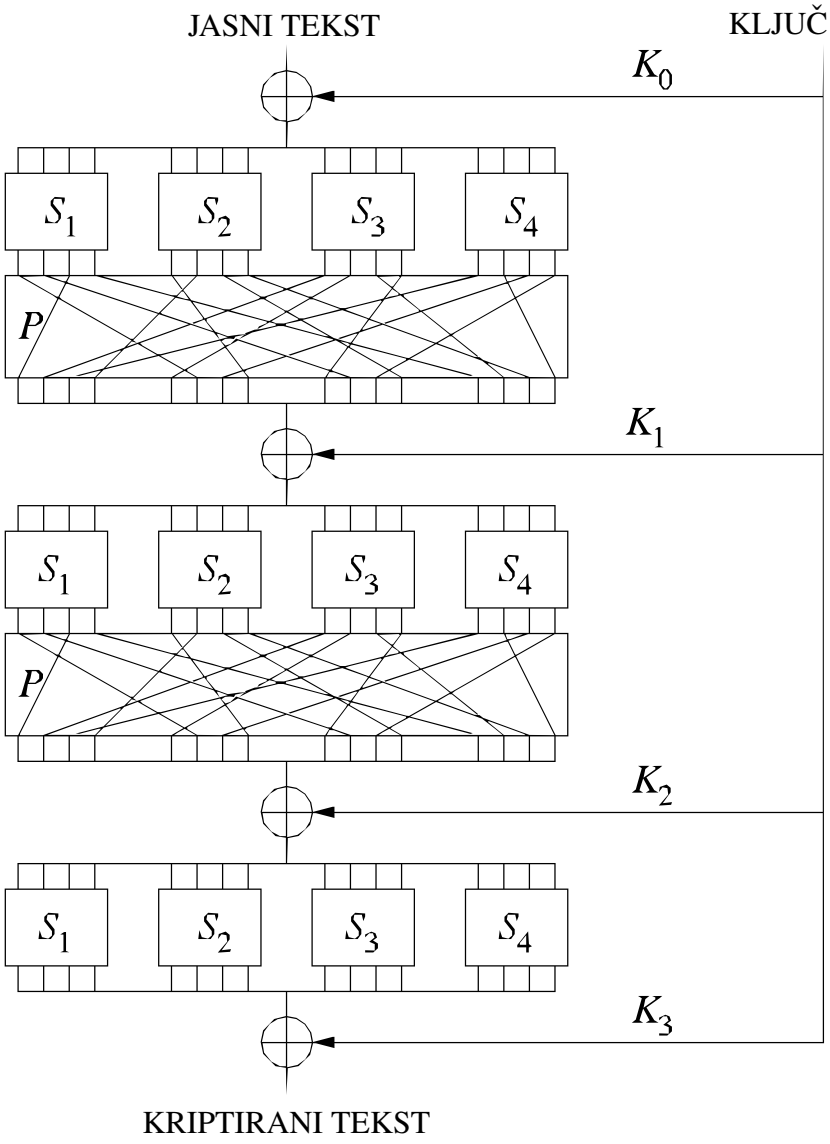
Prilikom ocjene sigurnosti kriptografskog uređaja moraju se uvesti pretpostavke o informacijama koje bi napadač mogao imati. Najjača pretpostavka koja se nadovezuje na Kerckhoffsovo načelo[1] je da napadač zna sve implementacijske detalje o kriptografskom uređaju [26].

2.1. Kriptoalgoritam AES

Kriptoalgoritam AES (engl. *Advanced Encryption Standard*) je simetričan[2] kriptoalgoritam proglašen standardom 2001. godine. Podskup je kriptoalgoritma *Rijndael* kojega su razvili Rijmen V. i Daemen J., belgijski kriptografi. *Rijndael* je porodica kriptoalgoritama koji se razlikuju po veličini ključa i veličini bloka. AES kriptira blokove jasnog teksta fiksne veličine 128 bitova [3].

[1] Kriptoalgoritam mora biti siguran čak i kada su sve informacije o sustavu, osim tajnoga ključa, javno dostupne.

[2] Simetrični kriptoalgoritmi koriste isti ključ za enkripciju i dekripciju.



Slika 2.1: Supstitucijsko-permutacijska mreža [25].

2.1.1. Supstitucijsko-permutacijska mreža

AES je baziran na konceptu zvanom supstitucijsko-permutacijska mreža (engl. *substitution-permutation network, SPN*). Mreža na ulazu prima blok **jasnog teksta** (engl. *plaintext*) i **ključ** te primjenjuje nekoliko *rundi* (ili slojeva) supstitucije (tzv. S-kutije; engl. *S-box*) i permutacije (tzv. P-kutije; engl. *P-box*) kako bi generirala izlaz – **kriptirani tekst**.

Slika 2.1 prikazuje skicu supstitucijsko-permutacijske mreže s 3 runde. Ova mreža kriptira jasni tekst veličine 16 bitova u kriptirani tekst jednake veličine. S-kutije su označene s $S_i, i \in [1, 4]$, P-kutije (u oba slučaja jednake) s P, a pojedini potključevi s $K_i, i \in [1, 3]$. Pojedinačne S-kutije i P-kutije obavljaju pretvaranje ulaznih blokova bitova u izlazne blokove. Izlazima iz para S- i P- kutije (odnosno samo S- kutije u zadnjoj rundi) nakon svake runde se dodaje novi potključ, najčešće operacijom isključivo-ili (engl. *exclusive OR, XOR*).

Difuzija i konfuzija U kriptografiji, difuzija i konfuzija dva su bitna svojstva kriptoalgoritama koja su nužna kako bi algoritam bio siguran. **Difuzija** u kriptoalgoritmu, poznata i kao efekt lavine (engl. *avalance effect*). znači da se promjenom jednoga bita jasnog teksta statistički mijenja polovica bitova u kriptiranom tekstu, i obrnuto. Prema svojstvu **konfuzije**, svaki bit kriptiranog teksta trebao bi ovisiti o više bitova jasnog teksta i ključa, tako da veza između bita kriptiranog teksta i jasnog teksta nije jasna.

S-kutija S-kutija deterministički zamjenjuje manji blok bitova (ulazni blok u S-kutiju) s drugim blokom bitova. S-kutija u supstitucijsko-permutacijskoj mreži mora biti invertibilna kako bi se mogla provoditi dekripcija. Dobro dizajnirana S-kutija imat će izraženo svojstvo difuzije. S-kutije kod AES-a rade nad blokovima veličine 8 bitova.

P-kutija P-kutija provodi permutaciju bitova: na ulazu prima izlazne bitove iz S-kutije, permutira ih te ih prosljeđuje u S-kutiju iduće runde. Dobro dizajnirana P-kutija ima svojstvo da bitove jedne S-kutije prosljeđuje u što više S-kutija iduće runde.

2.1.2. Opis kriptoalgoritma AES

Kriptoalgoritam AES temelji se na ideji supstitucijsko-permutacijske mreže. Postupak kriptiranja obavlja se u rundama, a broj rundi ovisi o veličini ključa: za ključ veličine 128 bita algoritam se provodi u 10 rundi, za ključ veličine 192 bita algoritam se provodi u 12 rundi, a za ključ veličine 256 bita se algoritam provodi u 14 rundi. U svakoj rundi, osim u zadnjoj, obavljaju se četiri operacije: supstitucija (*SubBytes*), posmak redova (*ShiftRows*), miješanje stupaca (*MixColumns*) te dodavanje potključa (*AddRoundKey*). U zadnjoj rundi se ne obavlja miješanje stupaca. Međurezultat kriptiranja koji se prosljeđuje između pojedinih funkcija naziva se **stanje** (engl. *state*) [8, 3].

Na početku rada algoritma, blok jasnog teksta i ključ smještaju se u pravokutne nizove bajtova u 4 retka, a broj stupaca ovisi o duljini bloka odnosno ključa. Primjer rasporeda ključa i bloka u takve pravokutne nizove dan je tablicom 2.1. Ovakva struktura koristi se u daljnjim koracima algoritma.

Tablica 2.1: Primjer 192-bitnog ključa (a) i 128-bitnog bloka podataka (b) [3].

a_{00}	a_{01}	a_{02}	a_{03}	a_{04}	a_{05}
a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}
a_{20}	a_{21}	a_{22}	a_{23}	a_{24}	a_{25}
a_{30}	a_{31}	a_{32}	a_{33}	a_{34}	a_{35}

b_{00}	b_{01}	b_{02}	b_{03}
b_{10}	b_{11}	b_{12}	b_{13}
b_{20}	b_{21}	b_{22}	b_{23}
b_{30}	b_{31}	b_{32}	b_{33}

Potključevi Prije ulaska u runde algoritma, ekspanzijom ključa generiraju se potključevi potrebni u rundama kriptiranja. AES-u je potrebno $N_r + 1$ potključeva, pri čemu je N_r broj rundi algoritma. Potključevi su po veličini jednaki veličini bloka.

Supstitucija Funkcija *SubBytes* provodi supstituciju bitova koristeći supstitucijsku tablicu (S-kutiju opisanu ranije).

Posmak redova Funkcija *ShiftRows* rotira (kružno posmiče) znakove u lijevo unutar bloka za unaprijed poznat broj mjesta. Pritom se prvi redak ne posmiče.

Miješanje stupaca U funkciji *MixColumns*, blokovi po 4 bajta u svakom retku međusobno se kombiniraju primjenjujući invertibilnu linearnu transformaciju u konačnom Galoisovom polju $GF(2^8)$. Funkcije *MixColumns* i *ShiftRows* zajedno imaju ulogu permutacije (P-kutija u SPN-u) te su u kriptoalgoritmu AES izvor difuzije [29].

Dodavanje potključa U zadnjem koraku runde provodi se dodavanje potključa funkcijom *AddRoundKey*. U toj funkciji stanje se kombinira s potključem za tu rundu provodeći operaciju isključivo-ili (engl. *exclusive OR, XOR*).

2.2. Implementacijski napadi

Implementacijski napad na kriptografski uređaj je napad koji iskorištava poznata svojstva kriptografskog uređaja koji se napada s ciljem pronalaska i iskorištavanja slabosti. Uzevši u obzir iskoristivost, cijenu i učinkovitost, implementacijski napadi su među najdominantnijim napadima na kriptografske uređaje [26]. Kod implementacijskih napada, nužno je da je tajni ključ u nekom obliku pohranjen u samom uređaju te da se taj tajni ključ aktivno koristi za vrijeme provođenja napada.

Najčešće vrste implementacijskih napada su: **napadi koji koriste sporedna fizikalna svojstva kriptografskih uređaja (engl. *side-channel attacks, SCA*)**, napadi sondiranjem (engl. *probing attacks*) te napadi koji analiziraju pogreške koje se javljaju u radu kriptografskih uređaja (engl. *fault attacks, FA*) [24]. Implementacijski napadi dijele se na aktivne i pasivne.

Pasivni napad Kod pasivnog napada, kriptografski uređaj u velikoj mjeri radi prema specifikacijama te se tajni ključ otkriva mjerenjem fizikalnih svojstava uređaja (npr. potrošnja energije ili vrijeme izvođenja kriptografskih operacija; engl. *timing attack*). Pasivni napadi najčešće su neinvazivni te prate samo pojave koje kriptografski uređaj uzrokuje svojim radom.

Aktivni napad Kod aktivnog napada kriptografski uređaj se fizički modificira kako bi se doveo izvan opsega ponašanja definiranog specifikacijom. Uređaj doveden u takvo stanje može postati ranjiv te se takva ranjivost može iskoristiti za otkrivanje tajnog ključa.

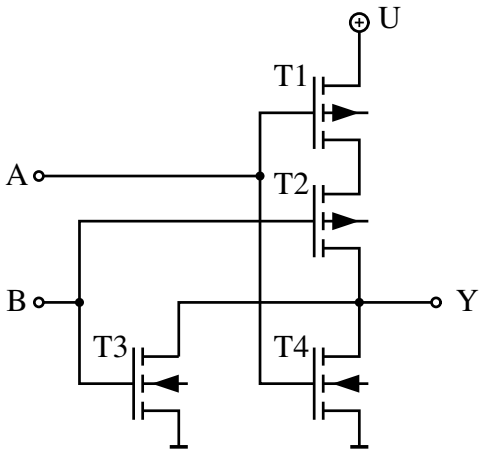
2.3. Potrošnja električne energije

Napadi koji iskorištavaju sporedna fizikalna svojstva kriptografskih uređaja pripadaju pasivnim napadima. Takvi napadi koriste spoznaje o **potrošnji električne energije (engl. *power analysis attack, PAA*)**, elektromagnetskom zračenju, vremenu izvođenja ili zvuku koji uređaj proizvodi, a s ciljem otkrivanja tajnoga ključa pohranjenog na uređaju [26]. U nastavku će biti opisan napad koji iskorištava spoznaje o potrošnji električne energije.

Osnovna ideja iza napada analizom potrošnje električne energije (*PAA*) je iskoristiti činjenicu da, u svakom trenutku, potrošnja električne energije kriptografskog uređaja ovisi o podacima koje kriptografski algoritam trenutno koristi i o kriptografskim operacijama koje se izvode. Diferencijalna analiza potrošnje električne energije jedan je od direktnih *PAA* napada.

2.3.1. Elektroničko sklopovlje

Moderni digitalni uređaji se, na niskoj razini, sastoje od velikog broja jednostavnih logičkih vrata koja su najčešće implementirana tehnologijom *CMOS*. Slika 2.2 prikazuje logička vrata *NILI* (engl. *NOR*) implementirana u tehnologiji *CMOS*. U gornjem dijelu mreže (engl. *pull-up network, PUN*; tranzistori T1 i T2) implementirana je funkcija *NILI*, dok je u donjem dijelu mreže (engl. *pull-down network, PDN*; tranzistori T3 i T4) implementiran njezin komplement *ILI* (engl. *OR*), po čemu je tehnologija i dobila naziv *CMOS* (engl. ***Complementary** metal–oxide–semiconductor*).



Slika 2.2: Logička vrata *NILI* implementirana pomoću tehnologije *CMOS* [1].

Kod promjene stanja izlaza mreže, tranzistori zbog kašnjenja jedno kratko vrijeme ostaju propuštajući za napon **U**, čime dolazi do kratkoga spoja. Taj kratki spoj očituje se u velikoj potrošnji električne energije. Samim time, što više sklopova mijenja stanje, to je veća disipacija električne energije.

2.3.2. SCA razlučitelji

Napadi koji koriste sporedna fizikalna svojstva kriptografskih uređaja mogu se podijeliti u dvije skupine prema pretpostavci odnosa između mjerenja fizikalnih svojstava uređaja i tajnoga ključa: napadi koji pretpostavljaju čvrstu vezu između mjerenja i tajnoga ključa te oni koji zaključke donose temeljem statističkog modela koji uzima u obzir mjerenja. Ti statistički modeli u drugoj skupini nazivaju se **SCA razlučitelji** (engl. *SCA distinguishers*).

Najčešći SCA razlučitelji kod DPA su: razlika srednjih vrijednosti (engl. *Difference of Means, DOM*), T-test, test varijance, Pearsonova korelacija te korelacija Spearmanovog ranga (engl. *Spearman's Rank Correlation*) [12].

2.3.3. Modeli emisije električne energije

Modeli emisije električne energije (engl. *power leakage model*) uvode pretpostavke o vezi između stanja kriptoalgoritma i potrošnje električne energije uređaja. Dva najpoznatija modela kod SCA su model Hammingove težine (engl. *Hamming Weight, HW*) i model Hammingove udaljenosti (engl. *Hamming distance, HD*). Model Hammingove težine pretpostavlja da je potrošnja električne energije korelirana s Hammingovom težinom stanja kriptografskog algoritma, a model Hammingove udaljenosti [2] pretpostavlja da je potrošnja električne energije ovisna o broju promjene bitova između dva stanja.

2.3.4. Opis DPA s razlučiteljem razlike srednjih vrijednosti

Postupak DPA započinje izgradnjom dvije matrice: **matrice tragova** i **matrice hipoteza**. Broj redaka u obje matrice jednak je broju tragova[3]. Broj stupaca u matrici tragova jednak je broju uzoraka prikupljenih mjerenjem, a matrica hipoteza ima onoliko stupaca koliko je mogućih vrijednosti jednog bajta ključa.

Pretpostavi se hipotetska vrijednost ključa Ψ. Selekcijska funkcija $D(H, b)$ računa vrijednost ciljanog bita b jednog okteta kriptiranog teksta H, pomoću jednog okteta ključa Ψ. Tragovi se zatim podijele u dvije grupe: tragovi za koje vrijedi $D(H, b) = 0$ smještaju se u jednu grupu (G_0), a oni za koje vrijedi $D(H, b) = 1$ u drugu (G_1). Ako je pretpostavljena vrijednost jednog okteta ključa Ψ bila točna, prosječna vrijednost tragova u G_1 bit će

[3]Za provođenje postupka diferencijalne analize potrošnje električne energije potreban je veći broj tragova ($\approx 10^3$).

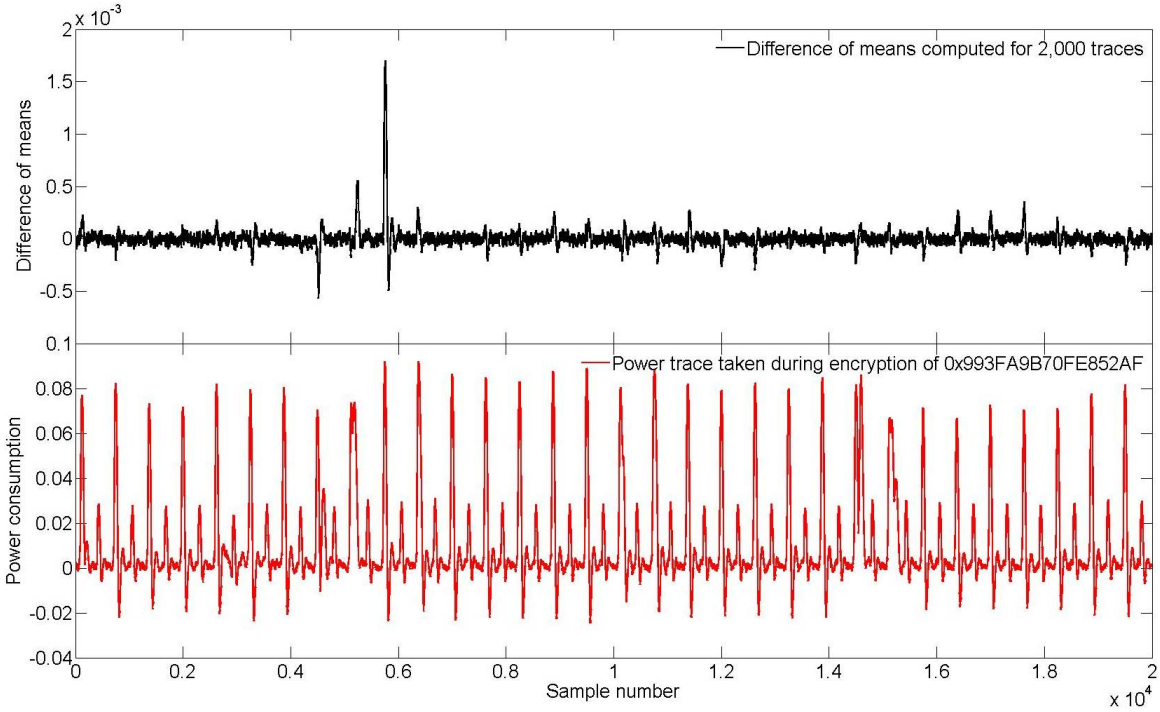
veća od prosječne vrijednosti tragova u G_0. U suprotnom će funkcija $D(H, b)$ s jednakom vjerojatnošću ($p = \frac{1}{2}$) svrstavati tragove u G_0 i G_1. Tada će se prosječni tragovi poklapati.

Pseudokôdom 2.1 opisan je postupak otkrivanja tajnog ključa u kriptoalgoritmu AES koristeći napad analizom diferencije potrošnje energije. Ovakav DPA napad, uz razlučitelj razlike srednjih vrijednosti, prvi su opisali Kocher i ostali 1999. godine [20]. Složenost ovog postupka ovisi o broju okteta koji čine tajni ključ, najvećoj vrijednosti koju ključ može poprimiti, broju tragova te broju uzoraka koje trag sadrži.

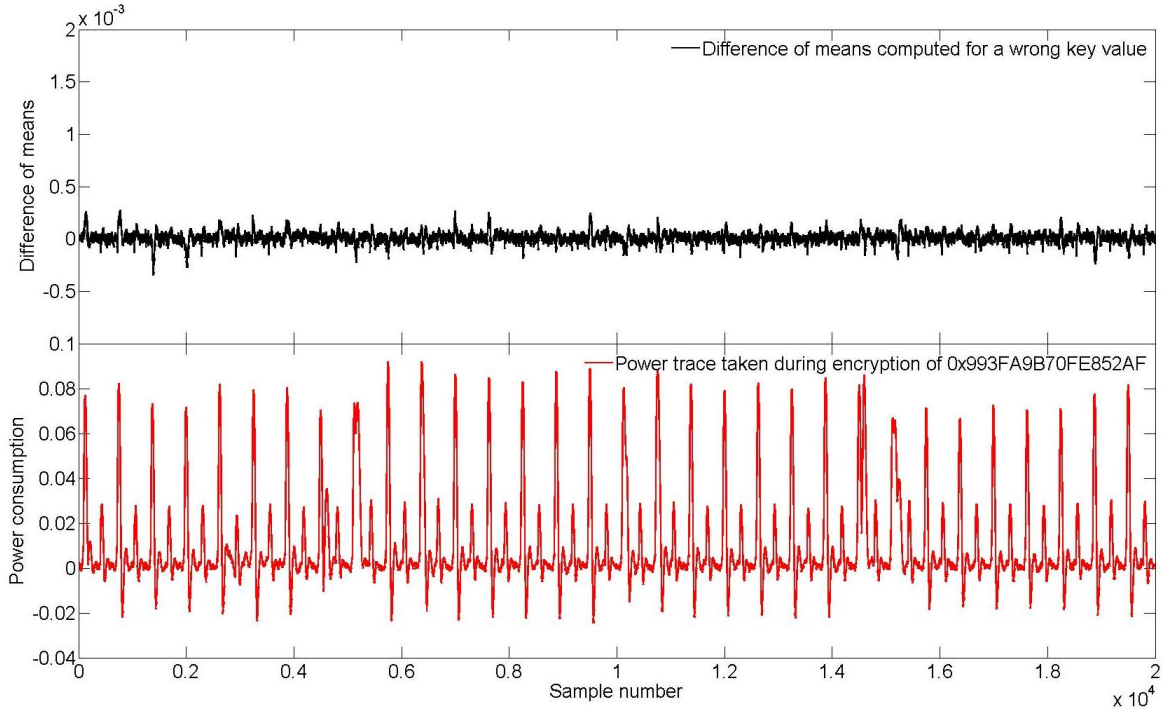
Pseudokôd 2.1: Otkrivanje tajnog ključa u kriptoalgoritmu AES napadom diferencijalnom analizom potrošnje električne energije uz korištenje razlučitelja razlike srednjih vrijednosti.

```
1  za svaki β u [1..16]:
2    inicijaliziraj hipotetsku matricu H[k × 256]
3    inicijaliziraj matricu razlika Δ[256 × ts]
4
5    za svaki Ψ u [1..256]:
6      H[:, Ψ] = M[:, β] ⊕ Ψ
7      H[:, Ψ] = S_kutija(H[:, Ψ])
8
9      inicijaliziraj G0[1, ts]
10     inicijaliziraj G1[1, ts]
11
12     za svaki Λ u [1..k]:
13       bit = D(H[Λ, k])
14       G_bit = G_bit + T[Λ, :]
15
16     Ḡ0 = prosjek(G0)
17     Ḡ1 = prosjek(G1)
18
19     Δ[Ψ, :] = |Ḡ0 − Ḡ1|
20
21   K[1, β] = redak(max(Δ))
22 vrati K
```

Slike 2.3 i 2.4 prikazuju vrijednosti matrica razlika Δ za jedan oktet Ψ tajnog ključa K. Na slici 2.3 vidljiv je *skok* u razlici između grupa G_0 i G_1 što je znak da je pogođena vrijednost Ψ. S druge strane, na slici 2.4 odstupanja ne postoje, iz čega se može zaključiti da je pretpostavljena vrijednost Ψ pogrešna.



Slika 2.3: Primjer traga T_i i vrijednosti u matrici Δ za **ispravno** pretpostavljenu vrijednost Ψ [17].

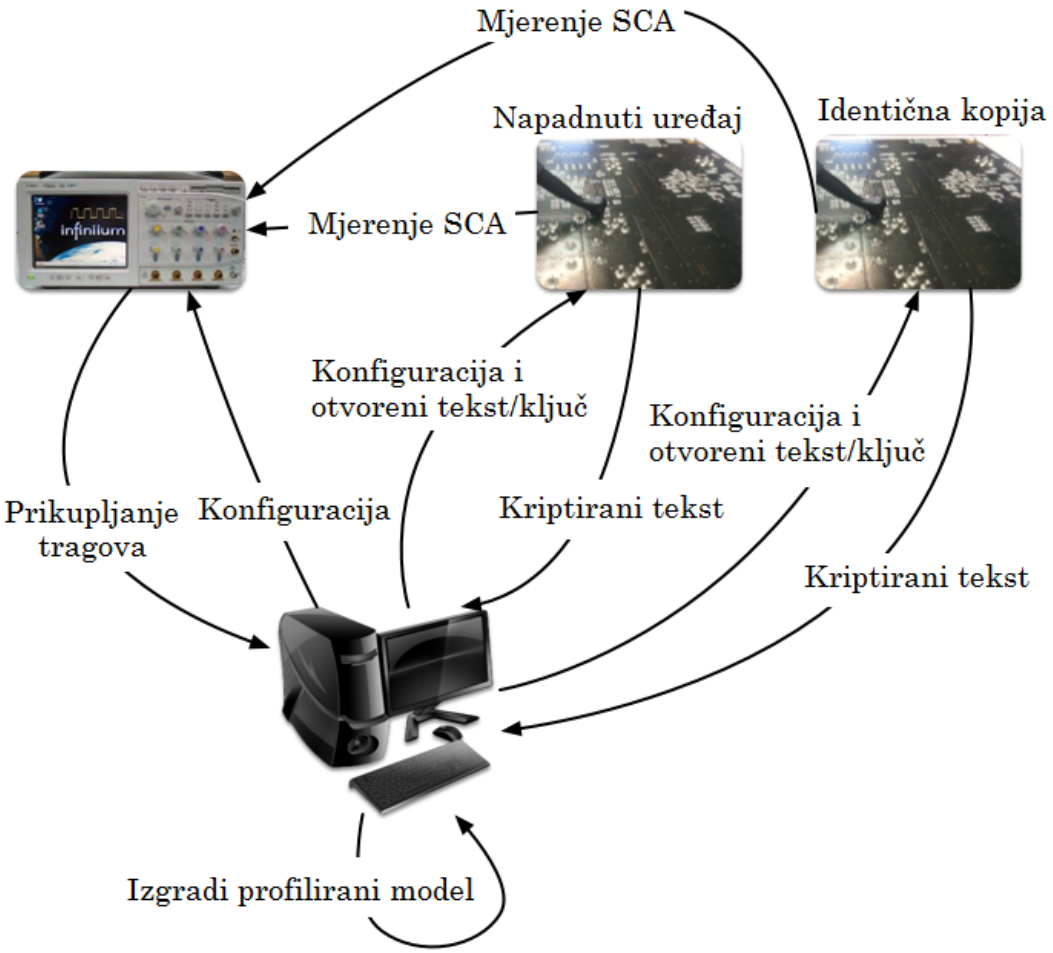


Slika 2.4: Primjer traga T_i i vrijednosti u matrici Δ za **pogrešno** pretpostavljenu vrijednost Ψ [17].

2.3.5. Profilirani napad

Prethodno opisani postupak spada u neprofilirane napade analizom potrošnje električne energije. Analiza potrošnje električne energije može se koristiti i u **profiliranim napadima** pri čemu napadač posjeduje kopiju kriptografskog uređaja te može naučiti model koji prema potrošnji električne energije klasificira tragove. Pritom kod klasifikacije trag može karakterizirati Hammingovu težinu izlaza algoritma ili egzaktnu vrijednost ključa. Jednom tako naučeni model može se zatim iskoristiti na originalnom kriptografskom uređaju.

Slika 2.5 prikazuje primjer profiliranog napada. Ovaj diplomski rad bavi se upravo profiliranim napadom analizom potrošnje električne energije, odnosno problemom klasifikacije tragova.



Slika 2.5: Prikaz profiliranog napada [26].

2.4. Opis skupova podataka

U ovom diplomskom radu korištena su dva glavna skupa podataka koji se razlikuju primarno u količini šuma: *DPAcontest v2* [32] i *DPAcontest v4* [33]. Originalni skupovi sadrže preko 3200 značajki, no korištenjem Pearsonove korelacije između pojedinih značajki i razreda, izabran je reprezentativni skup od 50 značajki s kojim se eksperimentira. Vrijednosti značajki u skupovima predstavljaju mjerene naponske razine.

2.4.1. DPAcontest v2

DPAcontest v2 (ili samo *DPAv2*) skup podataka opisuje mjerenja nezaštićene hardverske implementacije kriptoalgoritma AES. Ova mjerenja su vrlo šumovita te je omjer signala i šuma između 0.0069 i 0.0096. Ovaj skup podataka se sastoji od 100 000 tragova, od kojih se svaki trag sastoji od 50 uzoraka. Podaci su podijeljeni u omjeru $2:1$ u skup za učenje i skup za testiranje.

Postoje dvije inačice ovoga skupa podataka s obzirom na broj razreda: 9 i 256 razreda. U slučaju s 9 razreda, razredi odgovaraju Hammingovoj težini izlazne vrijednosti algoritma AES (8 bita može poprimiti Hammingovu težinu od 0 do 8). U slučaju s 256 razreda, razred odgovara samoj vrijednosti koju izlazna vrijednost algoritma poprimi (8 bita može poprimiti vrijednosti u dekadskoj bazi od 0 do 255).

2.4.2. DPAcontest v4

DPAcontest v4 (ili samo *DPAv4*) skup podataka opisuje mjerenja potrošnje električne energije uređaja koji koristi maskiranu implementaciju kriptoalgoritma AES. Ovaj skup podataka predstavlja nezaštićenu softversku implementaciju kriptoalgoritma AES. U odnosu na prethodno opisani skup podataka, ovaj skup sadrži puno veći omjer signala i šuma pa se može opisati kao manje šumovit od prethodnog.

Kao i prethodni skup, sastoji se od 100 000 tragova, od kojih se svaki trag sastoji od 50 uzoraka. Podjela na skup za učenje i testiranje je također $2:1$. Skup dolazi u dvije inačice: s 9 ili 256 razreda, pri čemu razredi imaju istu semantiku kao i u skupu podataka *DPAv2*.

3. Umjetne neuronske mreže

Razvoj umjetnih neuronskih mreža započeo je kao pokušaj objašnjavanja inteligentnog ponašanja ljudi i životinja. Konektivistički pristup smatra da mentalna stanja i ponašanje proizlaze iz interakcije velikog broja međusobno povezanih jednostavnih obradbenih jedinica [31]. Istraživanja su pokazala da se u ljudskom mozgu nalazi oko 10^{11} neurona, međusobno povezani s oko 10^{15} veza [28].

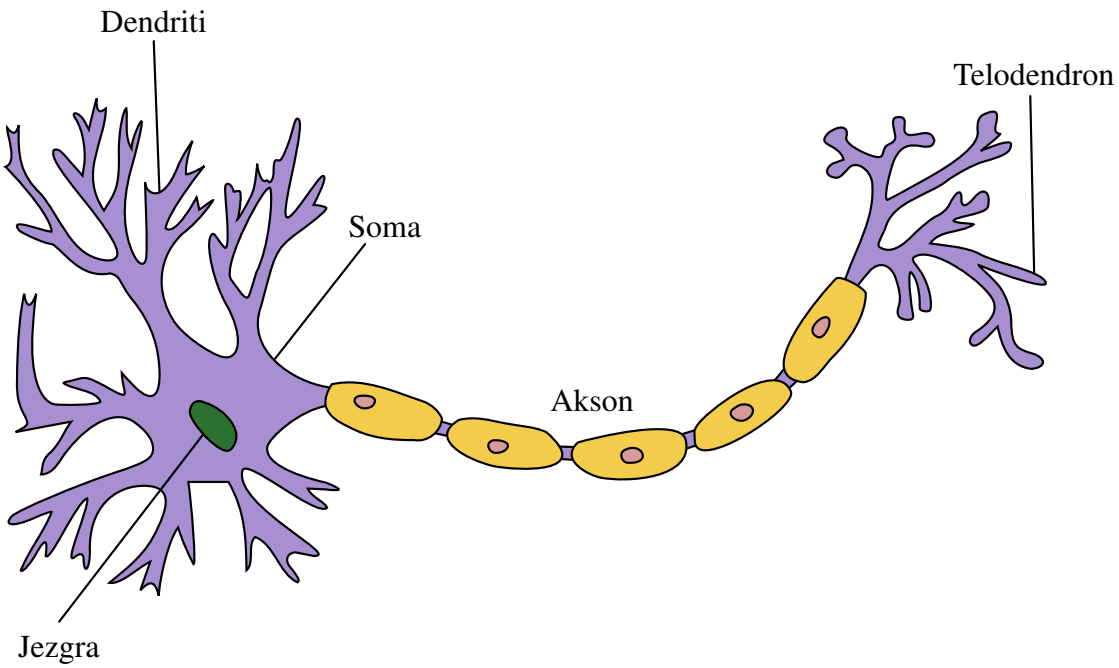
Umjetna neuronska mreža je računalni sustav za učenje koji koristi mrežu povezanih funkcija za *shvaćanje* i pretvaranje ulaza u neki izlaz, često u drugom obliku [10]. Koncept neuronskih mreža inspiriran je biologijom rada ljudskoga mozga te su umjetne neuronske mreže predstavnici konektivizma.

3.1. Biološki živčani sustav

Živčani sustav u kralježnjaka dijeli se na **periferni živčani sustav** i **središnji živčani sustav**. Periferni živčani sustav čine živci izvan mozga i leđne moždine te tvore vrlo gustu i razgranatu mrežu koja se proteže kroz cijelo tijelo kralježnjaka. Središnji živčani sustav najveći je dio živčanog sustava te se sastoji od mozga i leđne moždine. U središnjem se živčanom sustavu **obrađuju** i pohranjuju informacije dobivene od osjetila [21].

Osnovna jedinica živčanog sustava je neuron. Pojednostavljeno, biološki neuron je stanica koja prenosi električne impulse. Biološki neuron može se zamisliti kao sklopku s ulazom i izlazom. Ulaz i izlaz su povezani s drugim neuronima u sustavu. Neuron će se kao sklopka aktivirati ako na ulaz dobije dovoljnu pobudu od drugih neurona. U tom se slučaju šalje impuls na izlaz, koji je povezan s drugim neuronima u tijelu [21].

Na slici 3.1 je prikazana struktura neurona. Neuron se sastoji od **tijela stanice (some)**, **dendrita** i **aksona**. Iz **some** se granaju **dendriti** koji primaju impulse iz više različitih izvora kroz **sinapse** – veze među neuronima koje prenose električne ili kemijske signale od neurona do drugog neurona ili stanice [28]. Ti impulsi prenose se u jezgru stanice. Nakon što jezgra stanice primi aktivirajuće ili inhibirajuće signale, soma ih akumulira. Kad akumulirani signali prijeđu određeni prag (engl. *threshold value*), soma odašilje električni impuls koji



Slika 3.1: Struktura biološkog neurona [16].

se kroz akson prosljeđuje povezanim stanicama. **Akson** je dugačak[1] nastavak some koji je sinapsama povezan s dendritima drugih neurona ili s drugim vrstama stanica (npr. mišićnim stanicama).

Sinapse mogu biti električne i kemijske. **Električne sinapse** se nalaze u središnjem živčanom sustavu te su u suštini samo električni vodiči koji prenose električni impuls s jedne strane sinapse na drugu. Električne sinapse čine jaku i konstantnu vezu između odašiljača i primaoca signala. Obično kad se spominju sinapse, misli se na kemijske sinapse. **Kemijska sinapsa** se sastoji od praznog prostora (sinaptičke pukotine) te se impuls prenosi kemijskim putem pomoću neurotransmitera, molekula koje prenose signal kroz sinaptičku pukotinu. Kemijske sinapse su, za razliku od električnih, prilagodljive – postoji velik broj različitih neurotransmitera koji mogu biti otpuštani u različitom broju te mogu imati različito djelovanje[2]. Ova prilagodljivost može značajno varirati te je jedan od glavnih faktora pri učenju u mozgu – s vremenom veze između neurona mogu jačati ili slabiti te se smatra da je to ključno u procesu učenja [21].

3.2. Kratka povijest neuroračunarstva

1943. godine su W. McCulloch i W. Pitts u svojemu radu *A Logical Calculus of Ideas Immanent in Nervous Activity* definirali model umjetnog neurona za izračun logičkih funkcija **i**, **ili**

[1] Akson u čovjeka može biti dugačak i do 1m, npr. u leđnoj moždini.

[2] Neurotransmiteri mogu stimulirati jezgru postsinaptičke stanice ili smanjiti (inhibirati) stimulaciju.

i **ne**, nazvanog *TLU perceptron*. Kombiniranjem tih funkcija može se izgraditi proizvoljna Booleova funkcija, što znači da se TLU perceptron može koristiti za konstrukciju proizvoljne Booleove funkcije. Ipak, ovdje je riječ o konstrukciji, a ne učenju [7].

Donald O. Hebb, britanski biolog, 1949. godine je formulirao spoznaju poznatu kao **Hebbovo pravilo** (engl. *Hebbian rule*). Hebb tvrdi da se veza među neuronima jača kada neuroni pale zajedno. U to vrijeme, Hebb je postulirao ovu ideju, ali zbog nedostataka neurološkog istraživanja, nije ju mogao provjeriti [21]. Ovo pravilo ukazuje na to da se pri učenju mijenjaju jačine veza među neuronima.

Hebbovu ideju da učiti znači mijenjati jakost veza između neurona iskoristio je američki psiholog Frank Rosenblatt kako bi definirao algoritam učenja TLU perceptrona. To je napravio kroz izvještaj *The Perceptron: A Perceiving and Recognizing Automaton* iz 1957. godine i knjigu *Principles of Neurodynamics* iz 1962. godine [7].

Paul Werbos je 1974. u svojoj doktorskoj disertaciji iznio osnovu algoritma propagacije pogreške unazad (engl. *backpropagation of errors*). Desetljeće kasnije (1986. godine), D. Rumelhart, G. Hinton i R. Williams eksperimentalno su pokazali da ovaj pristup može generirati korisnu internu reprezentaciju ulaznih podataka u skrivenim slojevima neuronske mreže, što je analogno učenju u mozgu. Linearno neodvojivi problemi mogli su se riješiti višeslojnim perceptronom [21].

3.3. Umjetna neuronska mreža

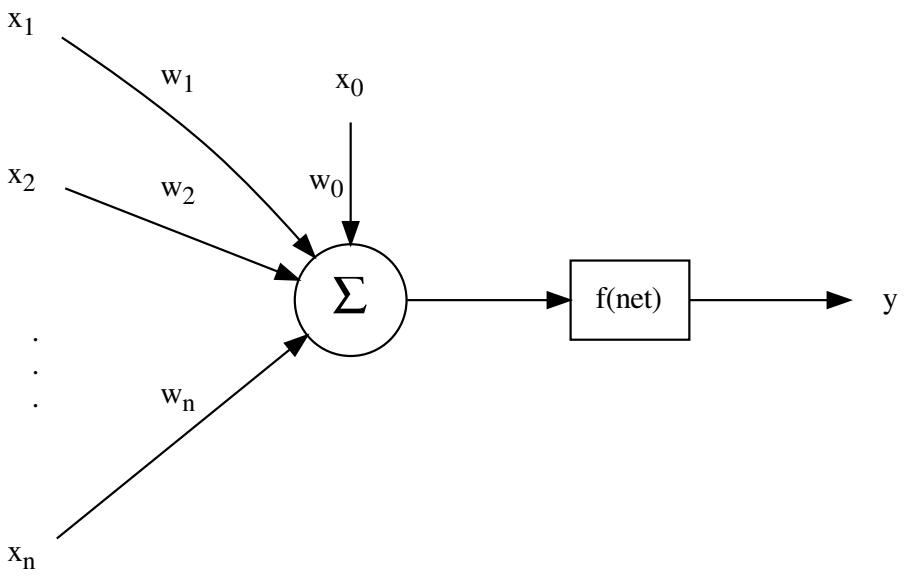
3.3.1. Umjetni neuron

Slika 3.2 prikazuje osnovni model umjetnog neurona. Umjetni neuron se sastoji od ulaza, težina, kombinacijskog operatora te prijenosne funkcije. Na ulaze se dovode ulazni podaci (označeni s $x_1..x_n$), a svaki ulaz ima svoju težinu $w_1..w_n$ koje određuju u kojoj mjeri svaki od ulaza pobuđuje neuron. Te se vrijednosti kombiniraju u ukupnu pobudu (net) te se ta vrijednost prosljeđuje u prijenosnu funkciju ($f(net)$). Izlaz iz prijenosne funkcije ujedno je i izlaz y iz neurona. Dodatno, neuron ima i još jedan, „fiktivni" ulaz x_0 na koji je uvijek dovedena vrijednost 1 te se njegovom težinom w_0 određuje prag paljenja (engl. *threshold*) neurona [7].

Ukupna pobuda net najčešće se računa kao težinska suma prema izrazu 3.1:

$$net = \sum_{i=0}^{n} w_i \cdot x_i \tag{3.1}$$

Zgodno je uočiti sličnosti sa strukturom biološkog neurona – ulazi s težinama predstavljaju dendrite koji su prethodno povezani sinapsama; ulazne vrijednosti se kombiniraju u onome što bi bilo tijelo stanice; a prijenosna funkcija i izlaz su analogni aksonu.



Slika 3.2: Model umjetnog neurona.

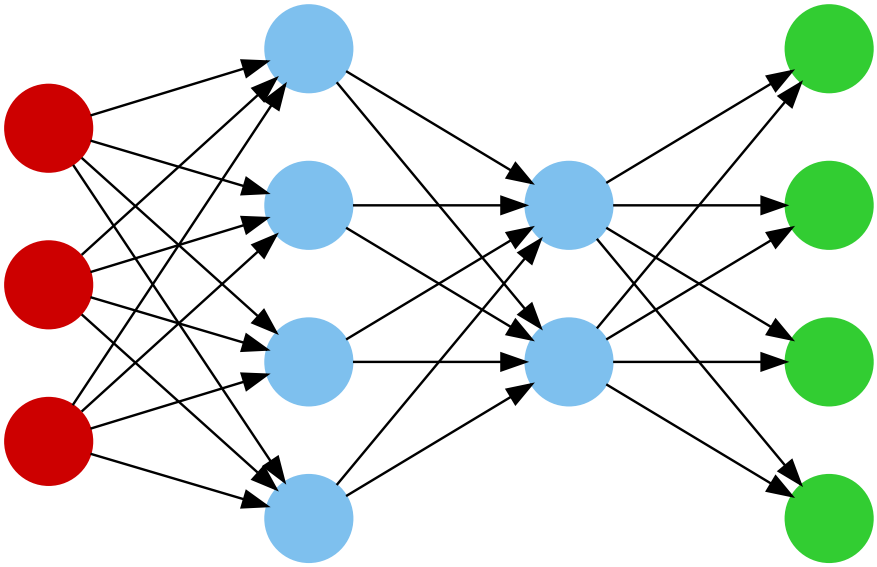
3.3.2. Umjetna neuronska mreža

Jedan neuron može obavljati samo jednostavnu obradu podataka, ali se kombiniranjem jednostavnih neurona može se dobiti složena struktura koja može obrađivati podatke na proizvoljan način. Takva struktura u kojoj su povezani jednostavni neuroni naziva se **umjetna neuronska mreža**.

U općenitoj neuronskoj mreži razlikujemo **ulazne neurone**, **izlazne neurone** te **unutarnje neurone**. Na ulazne neurone se dovodi neobrađeni podatak te oni ne rade nikakvu obradu već taj podatak prosljeđuju ostatku mreže – izlaz ulaznog neurona je sam ulazni podatak. Unutarnji neuroni su oni neuroni kojima su i ulaz i izlaz spojeni na neki drugi neuron. Oni rade samu obradu podataka te propagiraju obrađeni podatak kroz mrežu. Konačno, izlazni neuroni primaju vrijednosti na svojim ulazima od unutarnjih (ili ulaznih) neurona te dostavljaju rezultat obrade okolini – izlazi izlaznih neurona ujedno su izlazi neuronske mreže.

Ovisno o strukturi i načinu povezivanja neurona, razlikuju se vrste neuronskih mreža. Različitih tipova neuronskih mreža danas postoji mnogo, no ovdje će biti promatrane samo unaprijedne slojevite neuronske mreže (tzv. višeslojni perceptron; engl. *multilayer perceptron*) i, u nešto manjem opsegu, rijetke neuronske mreže.

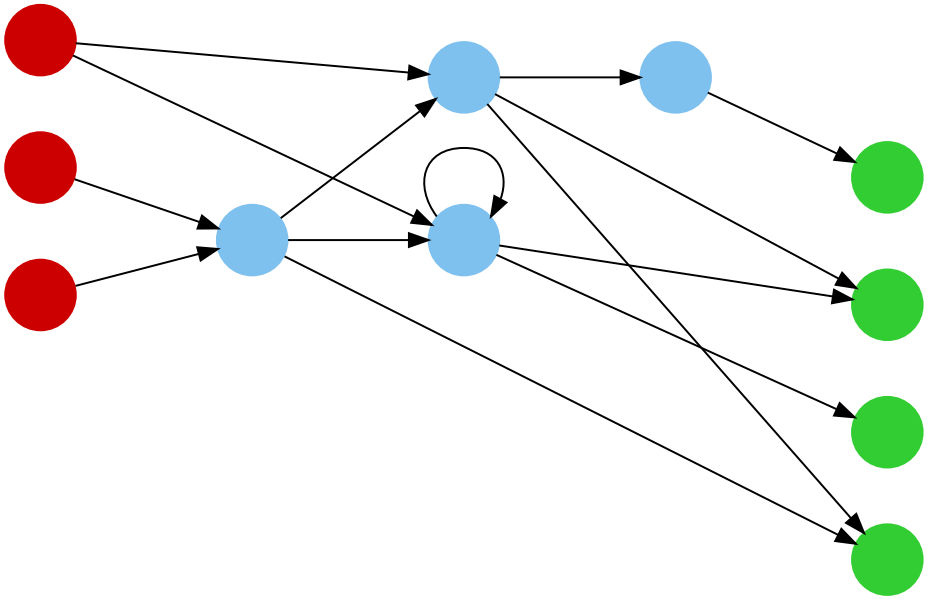
Unaprijedne slojevite neuronske mreže Višeslojni perceptron (prikazan na slici 3.3) sastoji se od ulaznog sloja (označen crveno), izlaznog sloja (označen zeleno) te proizvoljnog broja skrivenih slojeva (označeni plavo). Neuronska mreža je **unaprijedna** (engl. *feed-forward neural network*) ako i samo ako nema ciklusa. **Slojevita neuronska mreža** poseban je slučaj unaprijedne neuronske mreže za koji vrijedi da se izlazi neurona u sloju k računaju samo i isključivo temeljem izlaza neurona u sloju $k-1$, a lateralne veze nisu dozvoljene [7]. Algoritam *Backpropagation* za učenje neuronskih mreža primjenjiv je općenito na unaprijedne neuronske mreže, iako će se u ovom radu koristiti samo na slojevitim mrežama.



Slika 3.3: Višeslojni perceptron arhitekture **3x4x2x4**.

Rijeko povezane neuronske mreže Za razliku od slojevitih mreža, neuronska mreža je rijetko povezana ako ne vrijedi da je svaki neuron u sloju $k > 1$ povezan sa svim neuronima u sloju $k-1$. Štoviše, kod rijetkih neuronskih mreža nije nužno da postoje slojevi neurona te su dozvoljeni ciklusi i lateralne veze. Jednostavan primjer rijetko povezane neuronske mreže dan je na slici 3.4. Ova mreža, kao i ona na slici 3.3, ima 3 ulaza i 4 izlaza, ali unutarnji dio mreže nema jasno odvojene slojeve.

Rijetko povezane neuronske mreže nisu jednoznačno određene brojem unutarnjih neurona budući da postoji velik broj načina na koji se te neurone može povezati. Topologija neurona u rijetko povezanoj neuronskoj mreži jedan je od hiperparametara mreže.

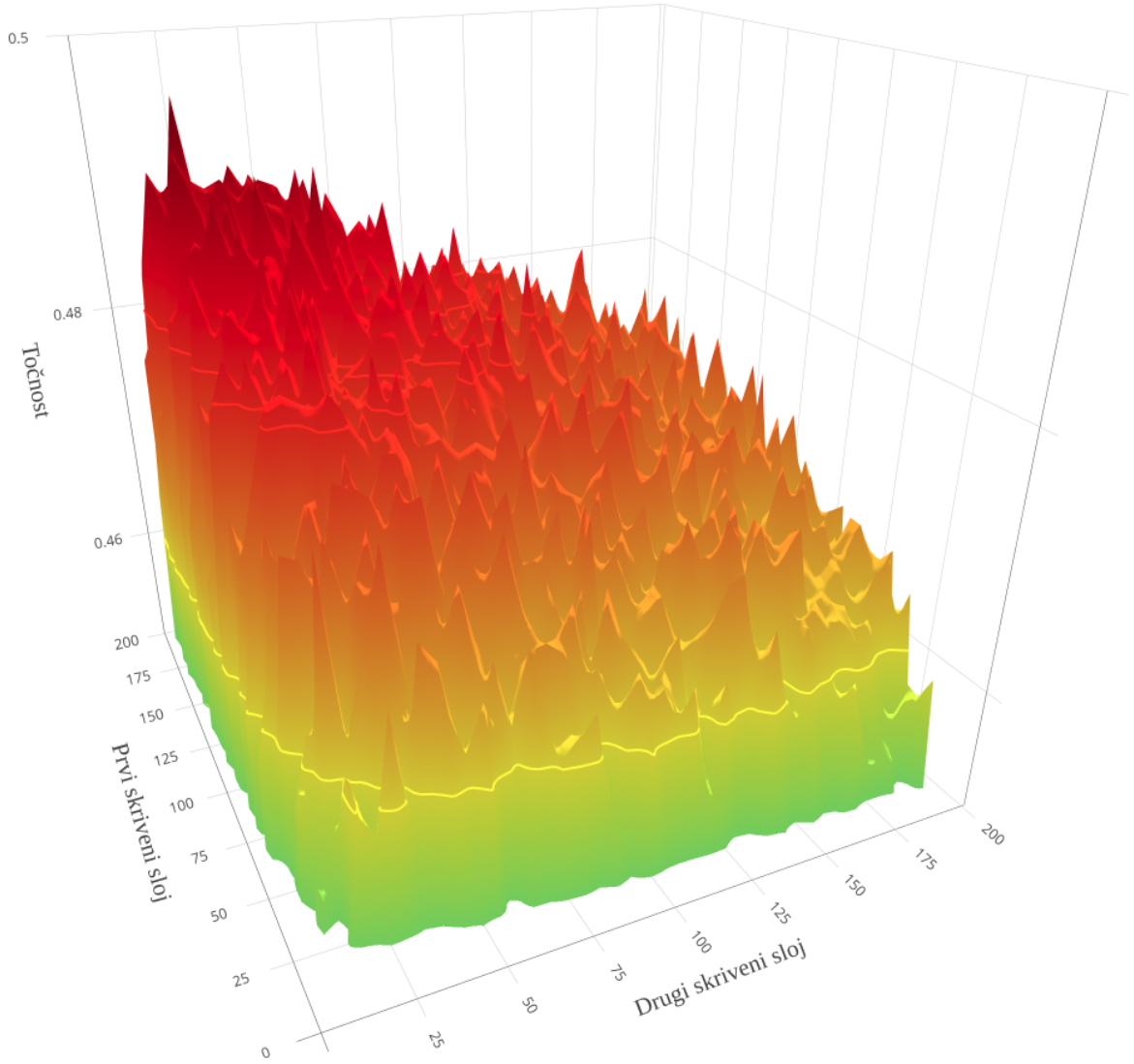


Slika 3.4: Primjer rijetko povezane neuronske mreže.

Zbog ciklusa u mreži, korištenje algoritma *Backpropagation* za učenje težina nije moguće te se za učenje ovakve mreže mora koristiti neki drugi optimizacijski algoritam.

3.3.3. Arhitektura višeslojnog perceptrona

Topologija višeslojnog perceptrona jednoznačno je određena s brojem slojeva i brojem neurona u svakom od slojeva. Arhitektura mreže prikazane na slici 3.3 je **3x4x2x4**. Ta mreža ima 4 sloja – ulazni sloj s 3 neurona, 2 skrivena sloja s po 4 i 2 neurona te izlazni sloj s 4 neurona. Za rješavanje pojedinog problema, vanjski slojevi (ulazni i izlazni sloj) obično su nepromjenjivi: veličina ulaznog sloja jednaka je broju značajki podatka koji se obrađuje, a veličina izlaznog sloja ovisi o vrsti problema – kod klasifikacije se, na primjer, može koristiti jednojediničnо kodiranje (engl. *one hot encoding*) pri čemu će veličina izlaznog sloja biti jednaka broju razreda. Arhitektura unutarnjih (skrivenih) slojeva je hiperparametar modela – mreža drugačije unutarnje arhitekture (npr. **3x5x5x4**) možda će biti bolja, a možda lošija u rješavanju danoga problema. Hiperparametri koji daju odlične rezultate na jednom problemu mogu biti vrlo loši na drugačijem problemu. Određivanje bolje arhitekture može se napraviti tek potpunim učenjem i uspoređivanjem vrijednosti neke mjere uspješnosti.



Slika 3.5: Točnost klasifikacije arhitektura umjetnih neuronskih mreža po rešetki do 200x200 dvodimenzionalnih mreža na skupu *DPAv4*.

Pretraživanje po rešetki Hiperparametar modela je vrijednost koja određuje svojstva modela te se ona postavlja prije početka procesa učenja modela. Kao što je već spomenuto, arhitektura neuronske mreže jedan je od hiperparametara. Uspješnost modela ovisi o izboru hiperparametara te odabir dobrog ili lošeg hiperparametra može značiti veliku razliku u radu modela strojnog učenja. Jedan od načina određivanja optimalne vrijednosti hiperparametra određenog modela je pretraživanje po rešetki (engl. *grid search*).

Kad se govori o uspješnosti modela, može se koristiti više mjera, no najčešće korištene su točnost (engl. *accuracy*) te F_1 mjera. Za potrebe ovog rada će se kao mjera uspješnosti koristiti **točnost klasifikacije**, osim ako to nije drugačije navedeno. Kod klasifikacije, točnost je udio točno klasificiranih primjera u skupu svih primjera.

Točnost klasifikacije značajno ovisi o arhitekturi skrivenih slojeva. Slika 3.5 prikazuje rezultate pretraživanja arhitekture višeslojnog perceptrona s dva skrivena sloja na problemu

klasifikacije skupa *DPAv4* u 256 razreda. X i Y osi označavaju broj neurona u dva skrivena sloja, a na Z osi je prikazana točnost klasifikacije. Pretraživana je rešetka arhitektura od 5x5 do 200x200 s kvantom 5, a svako mjerenje provedeno je 30 puta te su vrijednosti uprosječene.

Iz ove slike može se donijeti nekoliko zaključaka o ponašanju klasifikatora temeljenog na umjetnim neuronskim mrežama na ovom problemu: redoslijed slojeva jako utječe na uspješnost klasifikacije; većina arhitektura (izuzev rubnih) ima točnost klasifikacije između 44% i 50%; funkcija točnosti u ovisnosti o arhitekturi nije *glatka*; te točnost raste s povećanjem veličine prvoga skrivenog sloja mreže. Arhitektura koja je imala najbolju točnost pri klasifikaciji bila je **50x195x30x256** s točnošću 49.6%. S druge strane, arhitektura sa zamijenjenim redoslijedom slojeva – **50x30x195x256** – postizala je točnost klasifikacije od 45.4%, što je među lošijim rezultatima kod provjeravanih mreža. Generalno govoreći, na ovom problemu su se uspješnijima pokazale mreže koje imaju veći prvi skriveni sloj, a manji drugi skriveni sloj.

Pretraživanje ovoga područja zahtijevalo je učenje i evaluaciju 1600 mreža, što je na računalu s današnjom modernom grafičkom karticu trajalo ukupno 8 dana.

3.3.4. Algoritam *Backpropagation*

Algoritam *Backpropagation* najpoznatiji je algoritam za učenje težina unaprijednih neuronskih mreža. Taj algoritam je inačica algoritma gradijentnog spusta gdje se gradijenti težina računaju propagacijom unatrag. Da bi se mogao koristiti algoritam *Backpropagation*, **nužan uvjet je da je prijenosna funkcija derivabilna**.

Funkcija gubitka modela može se optimizirati kao funkcija težina W i pomaka b za sloj k. Izraz 3.2 prikazuje uporabu pravila ulančavanja za izračun težine w_{ij}, gdje je o_j izlaz prijenosne funkcije, a net_j težinska suma ulaza, odnosno ulaz u prijenosnu funkciju [23].

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \cdot \frac{\partial o_j}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ij}} \tag{3.2}$$

4. Evolucijsko računarstvo

Evolucijsko računarstvo je područje računarstva koje se bavi proučavanjem i izradom algoritama zasnivanih na pristupima inspiriranim biološkom evolucijom i Darwinovom teorijom o postanku vrsta [9]. Generalni algoritam evolucijskoga računarstva započinje s inicijalnom populacijom najčešće nasumičnih rješenja (jedinki) te kroz svaku iteraciju (generaciju) iz populacije uklanja manje poželjna rješenja, istovremeno dodajući nova kombiniranjem (križanjem) dobrih rješenja te uvođenjem malih, nasumičnih promjena (mutacija). U biološkom smislu, populacija jedinki se podvrgava prirodnoj selekciji i mutaciji. Ovim pristupom populacija postepeno *evoluira* prema boljim rješenjima.

Problem koji se u sklopu ovoga rada rješava evolucijskim računarstvom je određivanje arhitekture umjetne neuronske mreže – pronalazak optimalnog hiperparametra. Određivanje arhitekture može se predstaviti kao optimizacijski problem, a funkcija dobrote (engl. *fitness function*) može, na primjer, biti točnost klasifikacije.

4.1. NEAT i rijetka neuronska mreža

Algoritam *NEAT* (engl. *NeuroEvolution of Augmenting Topologies*) predložili su Stanley K. i Miikkulainen R. u svom radu *Evolving Neural Networks through Augmenting Topologies* kao metodu evolucije arhitektura rijetkih neuronskih mreža. Ovaj algoritam se pokazao izuzetno dobrim na problemima podržanog učenja [30]. Algoritam *NEAT* istovremeno optimira arhitekturu mreže i uči težine koristeći tehnike evolucijskog računanja.

Postupak evolucije počinje s minimalnom potpuno povezanom mrežom bez skrivenih slojeva te kroz proces evolucije razvija mrežu prema sve složenijim jedinkama. Budući da proces kreće od minimalnih mreža, rješenja teže biti minimalna. *NEAT* koristi povijesne oznake kako bi se moglo pratiti izvorište pojedine jedinke te uspoređivati slične jedinke sa sličnima. Kroz evoluciju arhitekture, algoritam koristi operatore križanja i mutacije i za optimiranje težina u mreži.

U ovom radu, *NEAT* je primijenjen na problem klasifikacije te se nije pokazao dobrim postupkom za učenje klasifikacijskih neuronskih mreža. Razlog ovome vjerojatno leži u učenju težina evolucijskim algoritmom nad velikim ($\approx 10^5$) brojem primjera. U poglavlju

6.2 prikazani su rezultati eksperimenata s algoritmom *NEAT*, no budući da su se oni preliminarno pokazali vrlo lošima, daljnja implementacija algoritma u sklopu ovog diplomskog rada je prekinuta u korist drugog pristupa, opisanog niže.

4.2. Genetski algoritam i višeslojni perceptron

Problem s implementacijom *NEAT*-a u 4.1 je istovremeno učenje težina i arhitekture, što čini *NEAT* lošim algoritmom za učenje (odn. evoluciju) klasifikatora. Drugi pristup može se dobiti odvajanjem koraka evolucije arhitekture i učenja težina.

Prvi korak bit će evolucija arhitekture višeslojnog perceptrona genetskim algoritmom. Drugi korak je klasično učenje težina algoritmom *Backpropagation* na mreži evoluiranoj u prvom koraku. Na ovaj način iskorištavaju se dobra svojstva algoritma propagacije pogreške unazad, a istovremeno mreže evoluiraju prema boljim hipermodelima.

Genetski algoritam je metaheuristika inspirirana procesom prirodne selekcije te Darwinovom teorijom evolucije. Genetski algoritam kodira potencijalna rješenja u strukturu zvanu *kromosom* te primjenjuje genetske operatore nad populacijom kromosoma (jedinki) kako bi *evoluirao* bolja rješenja. Genetski algoritam često se koristi za rješavanje raznih optimizacijskih problema [34].

4.2.1. Genetski algoritam: kodiranje rješenja

Budući da je cilj evoluirati arhitekturu slojevite mreže, različite arhitekture mogu se kodirati vektorima cijelih brojeva (engl. *integer vector*) različitih dimenzija. Za pojedini problem, sve mreže će imati nužno jednak broj ulaznih i izlaznih neurona – to su zapravo značajke i izlazni razredi. Budući da se oni ne mijenjaju, ne sprema ih se u gensku reprezentaciju rješenja.

Na primjer, neuronska mreža arhitekture **50x150x200x50x256** ima ulazni sloj od 50 neurona – to je ujedno i broj ulaznih značajki, izlazni sloj od 256 neurona – što je broj razreda u koji se trag pokušava klasificirati te tri skrivena sloja – prvi sa 150 neurona, drugi s 200 neurona te treći s 50 neurona. Kromosomska reprezentacija tog rješenja bit će vektor kao na slici 4.1. Slično, slika 4.2 prikazuje manju mrežu, arhitekture **50x150x50x256**. Ovakvo kodiranje omogućuje jednostavnu implementaciju genetskih operatora i intuitivnu interpretaciju zapisa.

150	200	50

Slika 4.1: Genski prikaz mreže **50x150x200x50x256**.

150	50

Slika 4.2: Genski prikaz mreže **50x150x50x256**.

4.2.2. Evaluacija mreže

Genetski algoritam svakoj jedinki (pojedinom rješenju) pridružuje njezinu dobrotu (engl. *fitness*) koja se određuje evaluacijom te jedinke. Bolje jedinke imaju veću šansu za napredovanjem kroz generacije te veću šansu da budu roditelji novim jedinkama. Za evaluaciju se koristi funkcija dobrote (engl. *fitness function*).

U ovom radu korištene su dvije funkcije dobrote. U oba slučaja se koristi prosječna mjera uspješnosti klasifikacije kroz 30 evaluacija pojedine mreže. Pritom se za učenje i evaluaciju koriste odvojeni skupovi podataka. Korištenje prosjeka 30 evaluacija pojedine mreže daje statistički značajne veličine.

Za probleme na skupu *DPAv4* kao funkcija dobrote koristi se prosječna točnost (engl. *accuracy*) klasifikacije, prikazano izrazom 4.1. Za probleme na skupu *DPAv2* koristi se prosječna F_1-mjera zbog velike količine šuma u podacima te varljivo visoke točnosti. Izraz 4.2 pokazuje izračun F_1 mjere pomoću mjera preciznosti (engl. *precision*) i odziva (engl. *recall*).

$$dobrota(x) = \frac{1}{30} \sum_{i=1}^{30} tocnost(x) \tag{4.1}$$

$$F_1 = 2 \cdot \frac{preciznost \cdot odziv}{preciznost + odziv} \tag{4.2}$$

4.2.3. Koncept vrsta (engl. *species*) i segregacija

Generalno govoreći, dvije arhitekture potpuno povezanih neuronskih mreža jednakih dimenzija moguće je međusobno uspoređivati, što je bilo vidljivo u poglavlju 3.3.3. Funkcija točnosti u tom slučaju nije linearna, ali nije deceptivna. U tom poglavlju vidi se da je mreža s arhitekturom **50x150x150x256** bolje klasificirala od mreže s arhitekturom **50x25x25x256**. Štoviše, uzmemo li dvije *dobre* mreže (npr. **50x80x50x256** i **50x150x150x256**) te ih uniformno križamo (npr. **50x150x50x256**), vjerojatnije ćemo dobiti *dobru* mrežu.

Tablica 4.1: Uspješnost mreža različitih arhitektura u klasifikaciji na skupu podataka *DPAv4* u 256 razreda

Arhitektura mreže	**Točnost**	$\boldsymbol{F_1}$
50x150x150x256	47.2%	46.2%
50x25x25x256	43.0%	41.9%
50x80x50x256	47.8%	47.0%
50x150x50x256	48.5%	47.7%
50x80x80x256	47.4%	46.6%
50x150x150x40x256	48.8%	47.9%
50x80x150x256	47.0%	46.2%

S druge strane, arhitekture različitih dimenzija nemaju takvu direktnu korelaciju. Mreža arhitekture **50x80x80x256** (2 skrivena sloja) dobro klasificira, kao i **50x150x150x40x256** (3 skrivena sloja). No, križamo li te dvije mreže, moguće je da ćemo završiti s mrežom **50x80x150x256**, koja daje nešto lošija rješenja od oba "roditelja". Tablica 4.1 prikazuje konkretne vrijednosti točnosti i F_1 mjere za spomenute mreže.

Iz ovoga se može zaključiti kako se križanje više isplati raditi nad jedinkama jednakih dimenzija. Zbog toga se uvodi segregacija populacije po vrstama (engl. *species*) koja osigurava upravo to.

4.2.3.1. Vrste

Klasični generacijski genetski algoritam u koraku selekcije odabire N jedinki od kojih stvara novu generaciju. Te se jedinke biraju iz cijele populacije koristeći neki operator selekcije.

Umjesto da se stvara nova generacija veličine N iz cijele prethodne populacije, populaciju se može prvo podijeliti na vrste. U ovom slučaju, **sve jedinke koje pripadaju istoj vrsti imat će jednaku dimenziju vektora**. Zatim se u koraku selekcije stvara nova generacija iz ovih vrsta – iduća generacija će nakon koraka selekcije sadržavati jednak broj jedinki svake vrste kao i prethodna. Tako sama selekcija osigurava da se raznolikost vrsti iz generacije u generaciju ne mijenja.

U sklopu ovog diplomskog rada usporedno su provođeni eksperimenti s algoritmom koji implementira vrsnu segregaciju i s klasičnim generacijskim genetskom algoritmu.

4.2.4. Genetski operatori

Osnovni operatori u genetskom algoritmu su **selekcija**, **križanje** i **mutacija**.

4.2.4.1. Selekcija

Operator selekcije je zadužen za odabir dobrih jedinki koje se prenose iz generacije u novu generaciju kao roditelji. U ovom se radu kao operator selekcije koristi **dvostruki turnir** (engl. *double tournament*) s klasičnim generacijskim genetskim algoritmom te **3-turnirska selekcija** s algoritmom koji implementira koncept vrsti.

K-turnirska selekcija K-turnirska selekcija iz cijele populacije izvlači nasumičan uzorak veličine k jedinki te se odabire ona s najvećom dobrotom u izvučenom uzorku [6]. Selekcijski pritisak raste s povećanjem parametra k, a ekstremni slučaj $k = N$ (pri čemu je N veličina populacije) uvijek vraća najbolju jedinku u cijeloj populaciji.

U ovom radu se koristi 3-turnirska selekcija s algoritmom koji implementira koncept vrsti tako da se turnir, umjesto nad cijelom populacijom, izvodi nad jedinkama iste vrste.

Dvostruki turnir Dvostruka turnirska selekcija [22] bira jedinku slično kao i klasična turnirska selekcija, ali sudionici u turniru se ne biraju iz cijele populacije nego se dohvaćaju iz *drugog* turnira temeljem njihove veličine kromosoma. Korištenje ove selekcije povećava parsimonijski pritisak[1].

U ovom radu, dvostruki turnir korišten je s klasičnim generacijskim genetskim algoritmom s vrijednostima $S_f = 3$ i $S_p = 1.5$.

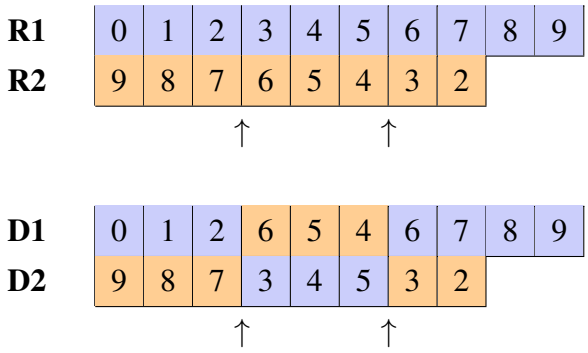
4.2.4.2. Križanje

Operator križanja zadužen je za kombinaciju dviju jedinki u nadi da će kombinacija dati rješenje koje je bolje – operator križanja pomiče rješenja prema (lokalnim) optimumima.

U ovom radu se kao operator križanja koristi **križanje s dvije točke prekida** (engl. *two point crossover*) s klasičnim generacijskim genetskim algoritmom te **diskretno križanje** (engl. *discrete crossover*) s algoritmom koji implementira koncept vrsti.

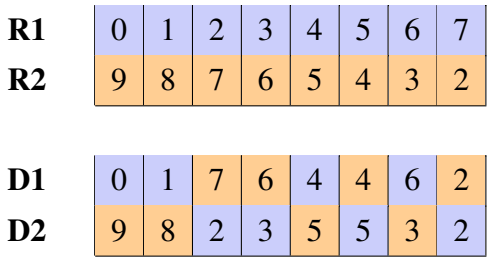
[1] Parsimonijski pritisak sprječava prekomjerni rast jedinki.

Križanje s dvije točke prekida Kod križanja s dvije točke prekida nasumično se odabiru dvije točke manjeg od dva kromosoma roditelja – točke kidanja. Potom se stvaraju dva djeteta D1 i D2: D1 dobiva prvi segment prvog roditelja (R1), drugi segment drugog roditelja (R2) te konačno treći segment prvog roditelja. D2 dobiva suprotne segmente. Ovo križanje prikazano je slikom 4.3.



Slika 4.3: Križanje s dvije točke prekida.

Diskretno križanje Diskretno se križanje provodi tako da se i-ta komponenta djeteta postavi na i-tu komponentu nasumično odabranog roditelja. Roditelj se iznova odabire za svaku od komponenti. Primjer diskretnog križanja dan je slikom 4.4.



Slika 4.4: Diskretno križanje.

4.2.4.3. Mutacija

Operator mutacije zadužen je za uvođenje noviteta u populaciju – mutacija pomiče rješenja izvan lokalnih optimuma.

U ovom radu koristi se više operatora mutacije, od kojih svaki ima određenu vjerojatnost ϕ da mutira jedinku. Svaka mutacija se može dogoditi na svakoj jedinki te je moguće da se na jednoj jedinki dogode i sve mutacije, kao i niti jedna mutacija.

Operatori mutacije koji se koriste u ovom radu su:

- **Brisanje elementa vektora** – svaki od elemenata vektora briše se iz vektora ima vjerojatnost ϕ da bude obrisan. $\phi = 0.1$. Ova mutacija u kontekstu topologije neuronskih mreža mijenja broj slojeva u mreži (te vrstu pojedine jedinke). Mutacija neće obrisati zadnji sloj u vektoru (odn. vektor ne može postati prazan kao rezultat ove mutacije).
- **Dodavanje elementa vektora** – vektoru se dodaje novi element na kraj vektora uz vjerojatnost ϕ. Novi element će imati nasumičnu vrijednost iz zadanog raspona. Slično kao i prethodni operator mutacije, ovaj operator mijenja broj slojeva u mreži te vrstu pojedine jedinke. U radu se koristi vjerojatnost $\phi = 0.35$ te raspon $[1, 400]$.
- **Normalna promjena elementa vektora** – svaki element vektora može biti promijenjen za nasumični broj iz normalne razdiobe $\mathcal{N}(\mu, \sigma^2)$ uz vjerojatnost promjene ϕ. Ova mutacija uvodi manje promjene u veličinu slojeva te može dovesti već dobro rješenje k još boljem. Vrijednosti koje se koriste u radu su $\phi = 0.5, \mu = 0, \sigma = 5$.
- **Uniformna promjena elementa vektora** – svaki element vektora može biti zamijenjen nekim nasumičnim brojem iz zadanog raspona uz vjerojatnost ϕ. Ovime se pojedini sloj neuronske mreže potpuno zamjenjuje novim. U radu se koristi $\phi = 0.1$ te raspon $[1, 400]$.

5. Implementacija

U ovom poglavlju, fokus će biti na implementaciji evolucije arhitekture višeslojnog perceptrona prateći ideje opisane u 4.2. Programska implementacija napravljena je u programskom jeziku Python. Biblioteka DEAP [11] je korištena za implementaciju genetskog algoritma, a biblioteka Keras [4] s Tensorflow pozadinom za učenje i evaluaciju neuronske mreže.

Cijeli postupak optimiranja može se podijeliti u tri cjeline: evoluciju populacije, evaluaciju jedinke te konačnu provjeru pobjednika. **Evolucija populacije** i **evaluacija jedinke** odvijaju se istovremeno, no evaluacija se može promatrati kao zaseban potprogram. Nakon što evolucijski dio postupka odredi najbolju arhitekturu, provodi se **provjera pobjednika** na do tada neviđenom skupu podataka – skupu za testiranje. Ovaj postupak naziva se unakrsna provjera (engl. *crossvalidation*).

Dva implementirana algoritma evolucije (sa segregacijom po vrstama i bez nje) razlikuju se samo u dijelu evolucije populacije te se sav prikazani pseudokôd odnosi na oba, osim ako to nije drugačije naznačeno.

Pseudokôd 5.1 prikazuje cijeli postupak određivanja arhitekture na visokoj razini. Ulazni parametri su veličina populacije **VEL_POP**, broj generacija **BROJ_GENERACIJA** te skupovi za učenje, validaciju i testiranje. U nastavku će biti detaljno objašnjeni korišteni potprogrami.

Pseudokôd 5.1: Cijeli postupak određivanja arhitekture.

```
1  POP = stvori_nasumicnu_populaciju(VEL_POP)
2  pobjednik = POP[0]
3
4  ponavljaj BROJ_GENERACIJA puta:
5    za svaku jedinku u POP:
6      jedinka.tocnost = evaluiraj(jedinka, skup_za_ucenje,
       skup_za_validaciju)
7      ako (jedinka.tocnost > pobjednik.tocnost):
8        pobjednik = jedinka
9    POP = stvori_iducu_generaciju(POP)
10
11 tocnost = evaluiraj(pobjednik, skup_za_ucenje, skup_za_testiranje)
12 vrati (pobjednik, tocnost)
```

5.1. Evolucija populacije

Za evoluciju populacije arhitektura koristi se vrsta **generacijskog genetskog algoritma**. Generacijski genetski algoritam iz koraka u korak iz populacije roditelja stvara novu populaciju djece koja potom postaju roditelji – stara populacija roditelja izumire [6]. Pseudokôdovi 5.2 i 5.3 opisuju verzije potprograma **stvori_iducu_generaciju** zaduženog za stvaranje populacije djece iz populacije roditelja.

Implementirane su dvije verzije evolucije populacije – s vrsnom segregacijom, opisanom u 4.2.3 te bez nje. U nastavku su opisane implementacije obje verzije algoritma.

5.1.1. Evolucija populacije bez vrsne segregacije

Pseudokôd 5.2: Evolucija populacije bez vrsne segregacije.

```
1  def stvori_iducu_generaciju(POP):
2    iduca_generacija = []
3    roditelji = selekcija(POP, velicina(POP)) # bira roditelje
4    za svake dvije jedinke (j1, j2) u roditelji:
5      d1, d2 = krizaj(j1, j2)
6      mutiraj(d1)
7      mutiraj(d2)
8      iduca_generacija += [d1,d2]
9
10   vrati iduca_generacija
```

Stvaranje iduće generacije započinje selekcijom roditelja iz cjelokupne populacije – roditelja se bira koliko ukupno ima jedinki u trenutnoj populaciji, budući da se križanjem dva roditelja dobivaju dva djeteta. Koristi se **dvostruka turnirska selekcija**, objašnjena u 4.2.4. Selekcija se provodi s ponavljanjima, što znači da ista jedinka može biti roditelj više puta u novoj generaciji.

Zatim se slijednim prolaskom po parovima odabranih roditelja provodi se križanje te se konačno potomci mutiraju. Nove jedinke **d1** i **d2** dodaju se u novu generaciju.

5.1.2. Evolucija populacije s vrsnom segregacijom

Pseudokôd 5.3: Evolucija populacije s vrsnom segregacijom.

```
1  def stvori_iducu_generaciju(POP):
2    # odvajanje vrsta
3    vrste = {}
4    za svaku jedinku u POP:
5      vrste[velicina(jedinka)] += [jedinka]
6
7    iduca_generacija = []
8    za svaku vrstu u vrste:
9      roditelji = selekcija(vrsta, velicina(vrsta)) # bira roditelje
10     za svake dvije jedinke (j1, j2) u roditelji:
11       d1, d2 = krizaj(j1, j2)
12       mutiraj(d1)
13       mutiraj(d2)
14       iduca_generacija += [d1,d2]
15
16   vrati iduca_generacija
```

Algoritam evolucije s vrsnom segregacijom je sličan onome bez vrsne segregacije, uz razliku da se generacijski postupak vrši zasebno nad vrstama. Prvi korak je podjela generacije roditelja na vrste s obzirom na veličinu kromosoma (vektora cijelih brojeva) – veličina kromosoma jednaka je broju skrivenih slojeva arhitekture. Vrste se spremaju u mapu **vrste** kojoj je ključ veličina kromosoma, a vrijednost je lista jedinki koje pripadaju toj vrsti. Zatim se provodi generacijski postupak za svaku vrstu.

Zatim se, slično prethodnom algoritmu, za svaku vrstu provodi selekcija[1] roditelja – bira se onoliko jedinki koliko sama vrsta ima jedinki. Tako će, zanemari li se mutaciju, iduća generacija imati jednak broj jedinki po vrstama kao i prethodna.

Ostatak algoritma jednak je evoluciji bez vrsne segregacije – provodi se križanje po parovima roditelja te se potomci mutiraju i dodaju u novu generaciju.

5.2. Evaluacija jedinke

Kao što je već rečeno u 4.2.2, za funkciju dobrote koristi se prosječna točnost (engl. *accuracy*) klasifikacije kroz 30 evaluacija pojedine mreže. Potprogram za evaluaciju 30 puta provodi izolirano stvaranje mreže, učenje težina algoritmom *Backpropagation* te konačno provjeru točnosti na skupu za testiranje. Točnost se tada uprosječuje i vraća kao dobrota. Ovaj postupak opisan je psudokôdom 5.4. Bitno za uočiti je kako se za učenje težina (u liniji

[1] k-turnirska selekcija objašnjena je u 4.2.4

7) i evaluaciju dobrote (u linijama 10-13) koriste različiti skupovi podataka.

Pseudokôd 5.4: Potprogram za evaluaciju mreže.

```
1  def evaluiraj(jedinka, skup_za_ucenje, skup_za_evaluaciju):
2    brojac = 0
3    tocnost = 0
4    dok brojac < 30:
5      brojac++
6      mreza = stvori_mrezu(jedinka)
7      mreza.backpropagation(skup_za_ucenje)
8
9      tocno = 0
10     za svaki primjer u skup_za_evaluaciju:
11       razred = mreza.klasificiraj(primjer[:-1])
12       ako razred == primjer[-1]: # primjer je ispravno
     klasificiran
13         tocno++
14     tocnost += (tocno / velicina(skup_za_evaluaciju)) / 30
15
16   vrati tocnost
```

Isti postupak koristi se i za evaluaciju dobrote jedinke (u liniji 6 u pseudokôdu 5.1) i za evaluaciju konačne točnosti pobjednika evolucijskog procesa (u liniji 11 u pseudokôdu 5.1).

5.3. Neuronska mreža

Poziv **stvori_mrezu** u liniji 6 pseudokôda 5.4 stvara instancu Keras modela unaprijedne slojevite neuronske mreže zadane arhitekture (kroz argument **jedinka**). Sve stvorene mreže imaju jednake hiperparametre, izuzev broja slojeva i broja neurona po sloju. U nastavku su opisani svi hiperparametri neuronske mreže i korištena svojstva algoritama za učenje, vidljivi i u tablici 5.1.

Tablica 5.1: Hiperparametri neuronske mreže.

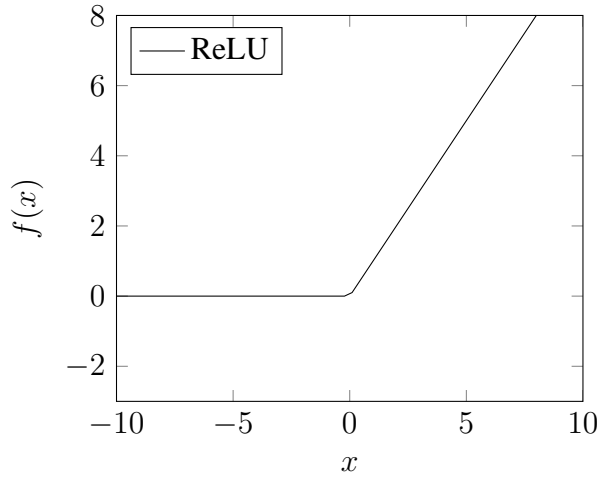
Hiperparametar	**Vrijednost**
Stopa učenja	0.001
Inicijalizacija težina	*Glorot*
Prijenosna funkcija	*ReLU*
Normalizacija	*Grupna normalizacija*
Rano zaustavljanje (*DPAv4*)	15 iteracija
Rano zaustavljanje (*DPAv2*)	40 iteracija
Optimizator	*Adam*

Stopa učenja Stopa učenja je hiperparametar algoritma *Backpropagation* koji određuje u kojoj se mjeri ažuriraju težine u svakom koraku algoritma. Korištena vrijednost stope učenja u ovom radu je **0.002**.

Inicijalizacija težina Pri korištenju algoritma *Backpropagation*, način na koji se postave početne težine može značiti razliku između konvergencije u razumnom vremenu i vrlo spore konvergencije sa znatno lošijim rezultatima. U ovom radu se za inicijalizaciju svih težina koristi Xavierova inicijalizacija [13]. Početne vrijednosti se postavljaju nasumično iz normalne distribucije $\mathcal{N}(0, \frac{1}{N})$ gdje je N broj ulaznih veza neurona.

Prijenosna funkcija Kao prijenosna funkcija koristi se **ReLU** (engl. rectified linear unit). Funkcija ReLU dana je izrazom 5.1 te prikazana na slici 5.1. Prednost korištenja funkcije ReLU kao prijenosne funkcije u odnosu na sigmoidalnu prijenosnu funkciju je manja izglednost pojave nestajućih gradijenata (engl. *vanishing gradient*) budući da je za pozitivne vrijednosti gradijent uvijek veći od nule [14].

$$f(x) = max(0, x) \tag{5.1}$$



Slika 5.1: Prijenosna funkcija ReLU.

Grupna normalizacija U svakom sloju mreže provodi se grupna normalizacija (engl. *batch normalisation*). To je metoda kojom se nastoji smanjiti promjenjivost distribucija aktivacija neurona u neuronskoj mreži. Ulazi u neurone pojedinog sloja slojevite neuronske mreže ovise o aktivacijama prethodnog sloja. Budući da se prilikom učenja neuronske mreže mijenjaju težine neurona, dolazi do promjene distribucija njihovih aktivacija. Težine u mreži se moraju prilagoditi novoj distribuciji zbog čega se otežava učenje težina u sljedećim slojevima. Grupna normalizacija normalizira podatke unutar jedne grupe (engl. *batch*) za učenje na razini pojedinog neurona [5].

Rano zaustavljanje Jedna od najčešćih metoda regularizacije, koja je korištena i u ovom radu, je rano zaustavljanje (engl. *early stopping*). Kod učenja neuronske mreže, greška na skupu za učenje postojano opada, ali greška na skupu za validaciju u nekom trenutku počne rasti što je znak prenaučenosti. Kako bi se izbjegla prenaučenost, prati se greška na skupu za validaciju. Tako se nakon učenja ne vraćaju zadnje postavljene težine nego one kad je greška na skupu za validaciju bila najmanja. Algoritam učenja se prekida kad se greška na **skupu za validaciju** nije smanjila više od nekoliko iteracija [15]. U ovom radu se rano zaustavljanje provodi nakon što se greška nije smanjila u **15 iteracija** na skupu podataka *DPAv4*, odnosno **40 iteracija** na skupu *DPAv2*.

Adam optimizator Adam (engl. *adaptive moment estimation* – procjena prilagodljivog momenta) [18] je inačica stohastičkog gradijentnog spusta koja u praksi može značajno ubrzati postupak učenja. Tijekom učenja, ovaj algoritam prati prosjek kretanja gradijenta i kvadrata gradijenta uz eksponencijalno zaboravljanje.

6. Rezultati

U sklopu ovoga rada, provođene su tri skupine eksperimenata: **određivanje egzaktne vrijednosti jednog okteta tajnoga ključa na skupu podataka *DPAv4*** – klasifikacija u 256 razreda, **određivanje Hammingove težine na skupu podataka *DPAv4*** – klasifikacija u 9 razreda te **određivanje Hammingove težine na skupu podataka *DPAv2*** – klasifikacija u 9 razreda.

Svaki eksperiment provođen je u izoliranom okruženju, neovisno o ostalim eksperimentima. Sve konačne vrijednosti dobivene su uprosječivanjem 30 mjerenja kako bi se dobile statistički značajne vrijednosti. Korišteni skupovi podataka, opisani u 2.4, veličine su 100 000 primjera. U ovom poglavlju prikazani su rezultati provedenih eksperimenata te su rezultati eksperimenata određivanja Hammingove težine ključa uspoređeni s rezultatima prikazanim u [27].

6.1. Opis eksperimenata

6.1.1. Određivanje egzaktne vrijednosti okteta ključa

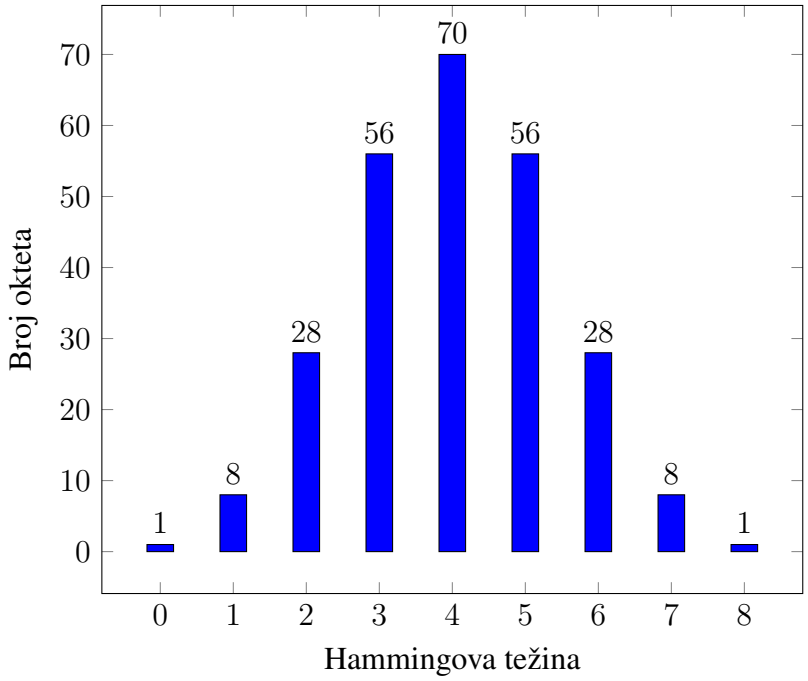
Konačan cilj kriptografskog napada je određivanje egzaktne vrijednosti tajnoga ključa. Ovaj napad pokušava postići upravo to – odrediti vrijednost jednog okteta tajnoga ključa. Budući da postoji $2^8 = 256$ različitih mogućnosti za vrijednost jednog okteta, ovo je problem klasifikacije tragova u **256 razreda**. Točnom klasifikacijom traga u jedan od 256 razreda tada se dobiva egzaktna vrijednost okteta ključa, što je i cilj napada. Distribucija uniformno odabranih ključeva po razredima je uniformna – podijeli li se svih 256 mogućih vrijednosti okteta u razrede, svaki razred imat će točno jedan oktet. Zbog toga je **točnost** dobra mjera uspješnosti ovoga napada.

6.1.2. Određivanje Hammingove težine okteta ključa

Ovom napadu cilj je odrediti Hammingovu težinu jednog okteta tajnoga ključa. Hammingova težina okteta je broj postavljenih bitova (bitova koji su 1). Na primjer, Hammingova težina okteta 10011000 je 3. Budući da u oktetu postoji 8 bitova, oktet se može klasificirati

u 9 različitih Hammingovih težina (0-8). Ovaj napad daje neku informaciju o ključu, ali ne i egzaktnu vrijednost ključa.

Za razliku od prethodnog napada, distribucija po razredima ovdje nije uniformna: raspodjela svih 256 mogućih vrijednosti okteta u 9 razreda Hammingovih težina prikazana je slikom 6.1. Iz ove slike je vidljivo da je razred 4 najzastupljeniji (sa 70 okteta), a razredi 0 i 8 najmanje zastupljeni (sa samo jednim oktetom). Kad bi neki klasifikator svaki trag klasificirao u 4. razred, točnost takvog modela bila bi **27%**, iako bi sam podatak o razredu u tom slučaju bio beskoristan u kontekstu DPA. Zbog toga je korisno prilikom evaluacije klasifikacije ovog problema provjeriti i matricu konfuzije. Tijekom evolucije se kao vrijednost dobrote koristila F_1 mjera.



Slika 6.1: Distribucija Hammingovih težina jednog okteta.

6.2. Algoritam *NEAT* i klasifikacija rijetkom neuronskom mrežom

Algoritam *NEAT* primijenjen je na evoluciju klasifikatora temeljenog na rijetko povezanim neuronskim mrežama nad skupom *DPAv4* (budući da je od dva skupa podataka u ovom radu ovaj lakši za uspješnu klasifikaciju) – klasifikacija u 9 razreda. Topologije evoluiranih klasifikatora imale su svojstva da imaju mali broj skrivenih neurona ($15 - 25$ neurona), a relativno veliki broj veza ($400 - 700$ veza), što sugerira tendenciju evolucije prema potpuno

(ili barem gušće) povezanoj mreži. To je također rezultiralo i velikim brojem težina za učenje evolucijskim algoritmom što je vremenski zahtjevno.

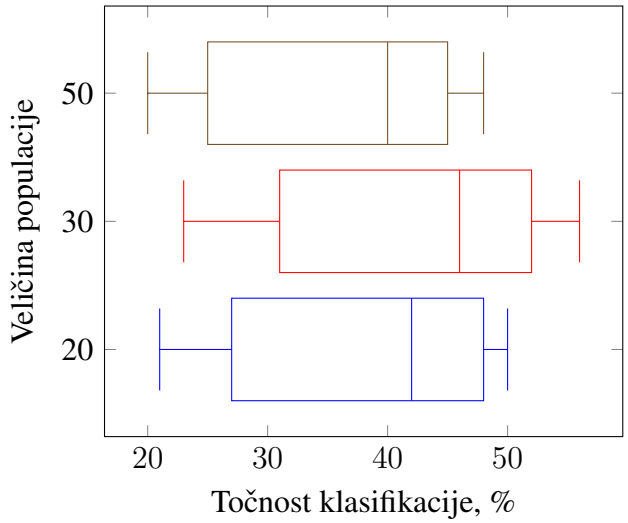
6.2.1. Određivanje Hammingove težine okteta ključa

Korišteni parametri algoritma dani su u tablici 6.1. Ove vrijednosti su odabrane po uzoru na vrijednosti predložene u [30] uz prilagodbu ovom problemu i ponašanju algoritma.

Tablica 6.1: Hiperparametri algoritma *NEAT* pri klasifikaciji u 9 razreda.

Hiperparametar	**Vrijednost**
Broj evaluacija	600 evaluacija
Koeficijent c_1	1.0
Koeficijent c_2	1.0
Koeficijent c_3	3.0
Prag kompatibilnosti δ_t	5.0
Vjerojatnost dodavanja veze	0.3
Vjerojatnost dodavanja neurona	0.05

Slika 6.2 prikazuje točnost klasifikacije klasifikatora evoluiranih algoritmom *NEAT* s tri različita parametra veličine populacije. U sva tri slučaja je evolucija bila ograničena fiksnim brojem evaluacija mreže. Provođeno je po 10 eksperimenata za svaku veličinu populacije.



Slika 6.2: Rezultati: klasifikatori evoluirani algoritmom *NEAT*; skup podataka *DPAv4*, klasifikacija u 9 razreda.

Prosječna točnost klasifikatora evoluiranih kroz veličinu populacije od 30 jedinki je 46%. Iz slike 6.2 je vidljiva i velika varijanca rezultata što ukazuje na nekonzistentnost i nepouzdanost ovakvog klasifikatora. Dobivena točnost klasifikacije od 46% je vrlo loša na ovom

problemu. U nastavku će biti pokazano da evoluirani višeslojni perceptron postiže mnogo višu točnost, kao i ostali radovi u području.

S obzirom na loše rezultate pri klasifikaciji u 9 razreda, eksperimenti klasifikacije u 256 razreda ovim pristupom nisu provođeni.

6.3. Genetski algoritam i klasifikacija višeslojnim perceptronom

Optimizacija arhitekture klasifikatora temeljenog na višeslojnom perceptronu rađena je varijantama genetskog algoritma. Ovisno o problemu, korištene su jedna ili obje varijante genetskog algoritma opisane u 5.1. Provođeno je više eksperimenata te su u ovom poglavlju prikazani relevantni rezultati.

U svim eksperimentima pretraživano je jednako područje – arhitekture veličine do 8 skrivenih slojeva, svaki sloj veličine do 400 neurona. Ukupna veličina područja pretraživanja je $6.57 \cdot 10^{20}$ arhitektura. U eksperimentima nad skupom podataka *DPAv2* je umjesto točnosti korištena F_1 mjera kao funkcija dobrote, kao što je opisano u 4.2.2.

6.3.1. Određivanje egzaktne vrijednosti okteta ključa – *DPAv4*

Za određivanje egzaktne vrijednosti okteta ključa provođeni su eksperimenti s različitim hiperparametrima evolucije te koristeći oba evolucijska algoritma opisana u 5.1.

Tablica 6.2: Rezultati: evolucija arhitekture višeslojnog perceptrona genetskim algoritmom; skup podataka *DPAv4*, klasifikacija u 256 razreda.

Algoritam	Broj gen.	Veličina pop.	Najbolja mreža	Točnost	F_1
$\mathcal{A}$	50	30	50x297x18x42x256	**49.5%**	48.6%
$\mathcal{A}$	50	30	50x266x13x29x50x256	**49.4%**	48.4%
$\mathcal{A}$	15	20	50x265x19x29x72x256	**49.4%**	48.4%
$\mathcal{A}$	10	15	50x260x26x45x256	**49.3%**	48.4%
$\mathcal{B}$	20	15	50x344x22x49x256	**51.1%**	50.2%
$\mathcal{B}$	15	10	50x384x27x51x256	**50.5%**	49.7%
$\mathcal{B}$	10	15	50x225x23x49x256	**50.3%**	49.5%

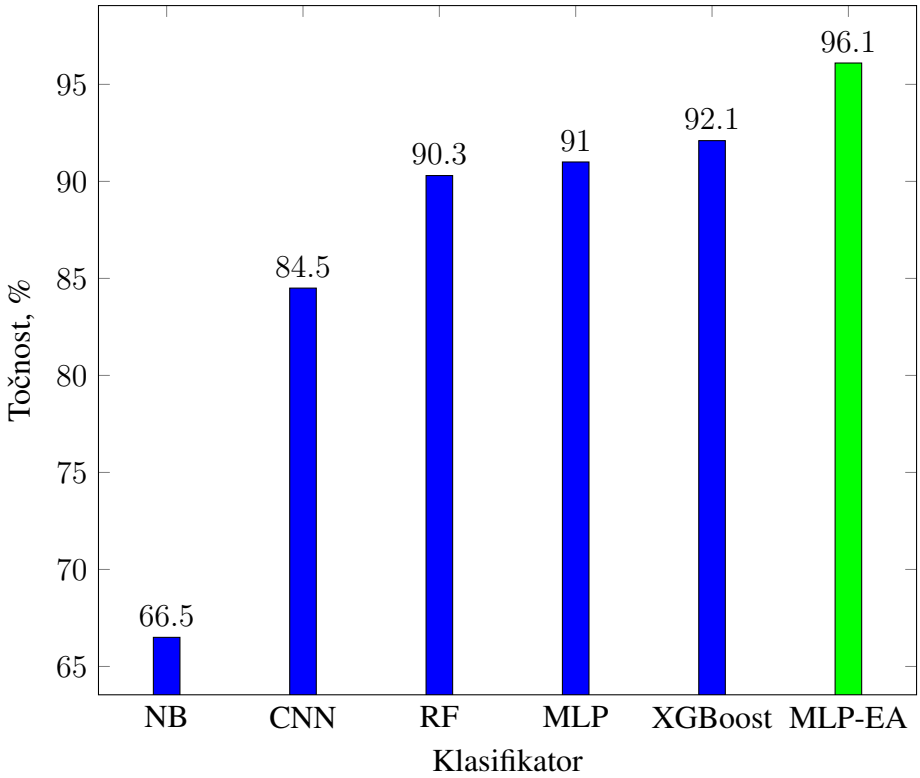
Tablica 6.2 prikazuje neke od mreža dobivenih u eksperimentima. Algoritam $\mathcal{A}$ u tablici označava algoritam bez vrsne segregacije, a algoritam $\mathcal{B}$ označava algoritam s vrsnom segregacijom. Iz tablice je vidljivo da se već sa 150 evaluacija u algoritmu bez vrsne segregacije

(10 generacija po 15 jedinki) dostiže točnost od $\approx 49\%$. Povećanjem ukupnog broja evaluacija točnost marginalno raste (unutar statističke pogreške). U tablici je prikazana i F_1 mjera koja se kod algoritma bez vrsne segregacije kreće oko $\approx 48.5\%$. Iz tablice je vidljivo da se algoritam s vrsnom segregacijom ponaša marginalno bolje u pronalasku arhitekture mreže (najbolja točnost 51.1% u odnosu na 49.5% te najbolja F_1 mjera 50.2% u odnosu na 48.6%), čak i uz manji broj evaluacija.

6.3.2. Određivanje Hammingove težine okteta ključa

6.3.2.1. Eksperimenti na skupu podataka *DPAv4*

Za napad određivanja Hammingove težine okteta ključa na skupu podataka *DPAv4* korišten je algoritam s vrsnom segregacijom koji se u prethodnom odjeljku pokazao jednako uspješnim kao i algoritam bez vrsne segregacije. Ovi eksperimenti provođeni su s **10 generacija** po **15 jedinki** (ukupno 150 evaluacija). Najbolja dobivena arhitektura mreže je **50x313x18x141x9**, čija je uprosječena točnost klasifikacije 96.1%.

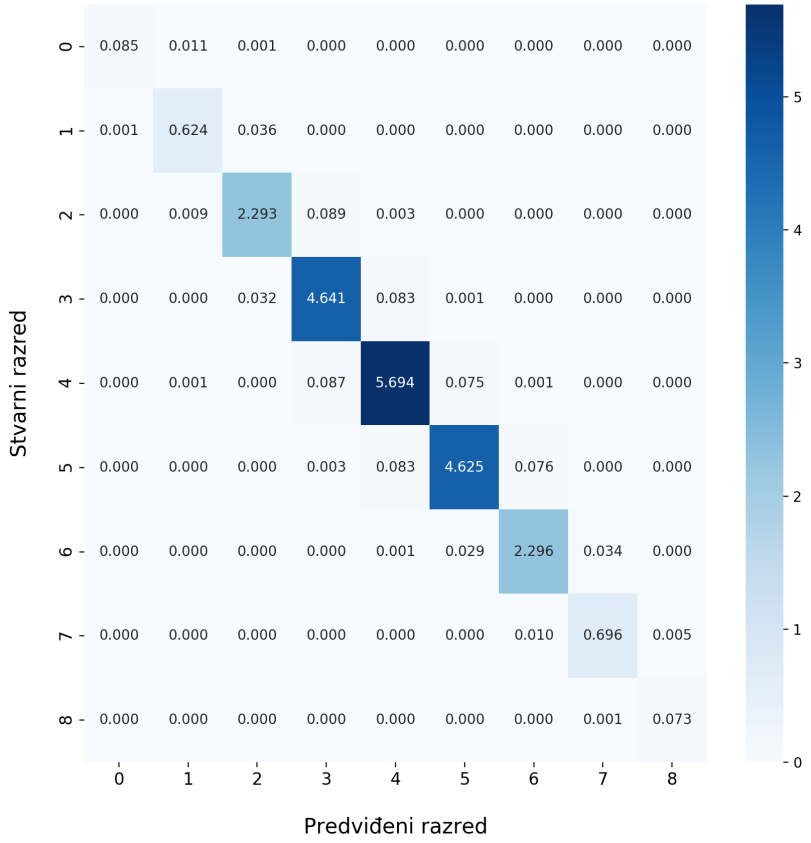


Slika 6.3: Usporedba rezultata s [27] na skupu podataka DPAv4.

Na slici 6.3 su uspoređeni rezultati dobiveni u radu [27] na istom skupu podataka. U tom radu korišteni su različiti klasifikatori, među kojima je i višeslojni perceptron (označen s *MLP*). Rezultat iz ovoga diplomskog rada prikazan je pod *MLP-EA* (*Multilayer Perceptron with Evolved Architecture*) te je vidljivo da neuronska mreža s evoluiranom arhitekturom

postiže veću točnost klasifikacije od ostalih klasifikatora prikazanih u referentnom radu.

Matrica konfuzije za klasifikator mrežom arhitekture **50x313x18x141x9** prikazana je na slici 6.4. Iako je već i iz visoke točnosti (96.1%) vidljivo da klasifikator dobro klasificira, matrica konfuzije to potvrđuje. Većina pogrešno klasificiranih tragova smješteno je u razrede 3, 4 i 5, što je i očekivano s obzirom na normalnu distribuciju Hammingovih težina.



Slika 6.4: Normalizirana matrica konfuzije za klasifikaciju u 9 razreda na skupu podataka *DPAv4* klasifikatorom arhitekture **50x313x18x141x9**.

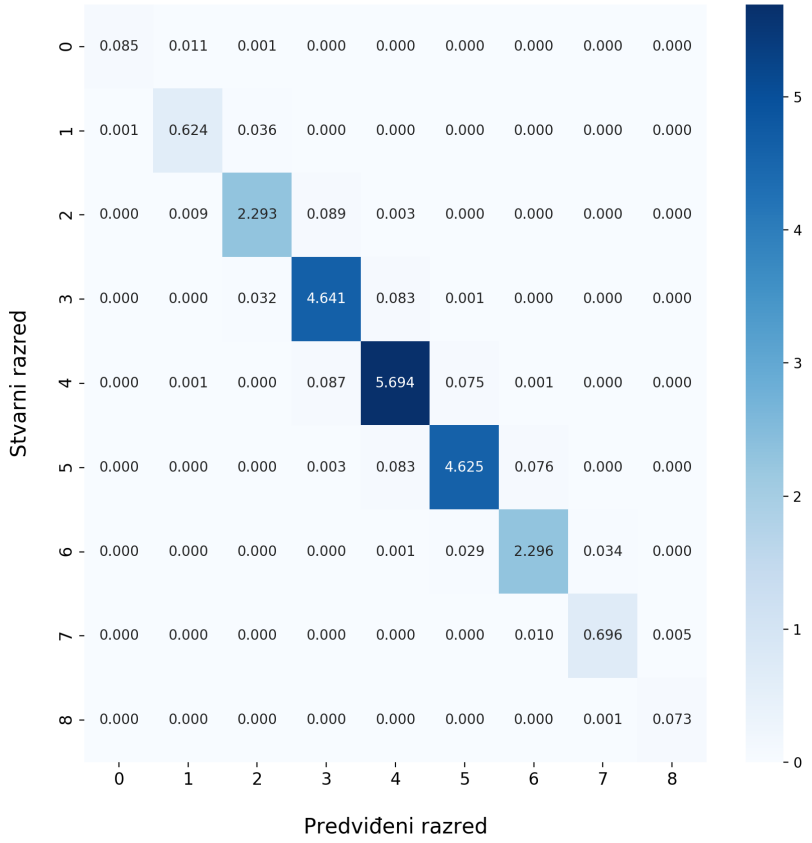
6.3.2.2. Eksperimenti na skupu podataka *DPAv2*

Skup podataka *DPAv2*, kao što je već spomenuto u 2.4, vrlo je šumovit te je zbog toga učenje modela na takvom skupu podataka otežano.

Za klasifikaciju kod napada određivanja Hammingove težine okteta ključa na skupu podataka *DPAv2* korišten je isti algoritam te slični hiperparametri[1] kao i kod skupa podataka *DPAv4*: **algoritam s vrsnom segregacijom**; evolucija kroz **20 generacija** po **15 jedinki** (ukupno 300 evaluacija). Kao funkcija dobrote korištena je F_1 mjera.

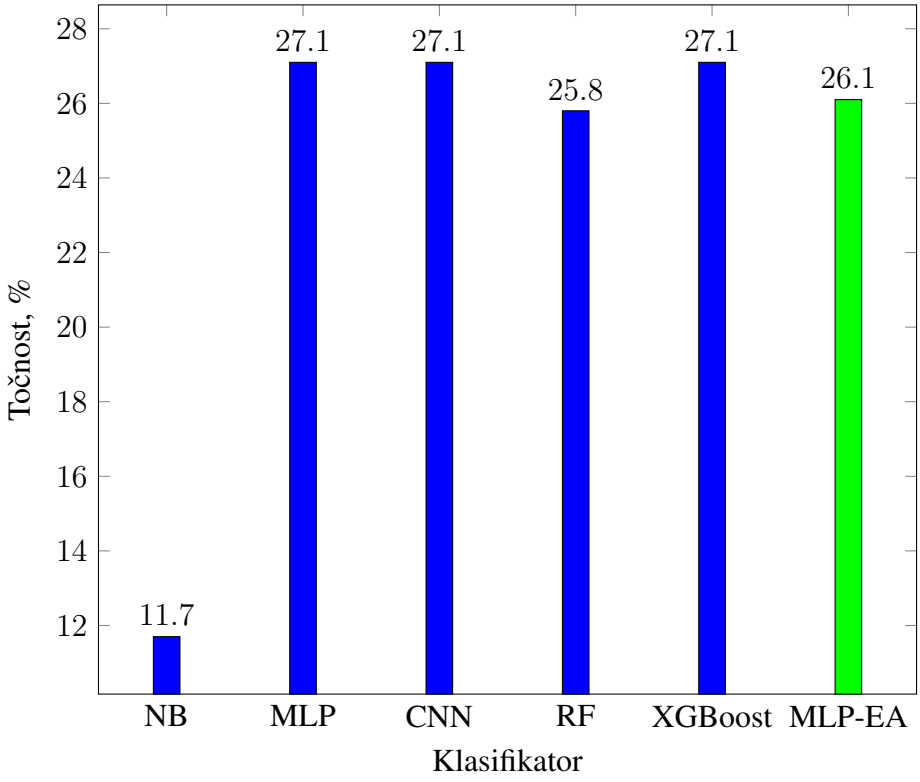
[1]Isprobane su i druge kombinacije, poput drugačije prijenosne funkcije i različitih stope učenja, ali se dobivani rezultati nisu pretjerano razlikovali.

Točnost klasifikacije dobivanih mreža bila je nešto niža od već spominjanih 27% što daje naslutiti da klasifikacija nije korisna u kontekstu Hammingove težine u DPA napadu. Po točnošću najbolje od evoluiranih mreža je **50x198x174x9** s vrijednošću F_1 mjere 18.1% i točnošću klasifikacije 26.1%. Iz matrice konfuzije, dane na slici 6.5, vidljivo je da klasifikator gotovo sve tragove smješta isključivo u razrede 3, 4 i 5, što ga čini neupotrebljivim u kontekstu DPA.



Slika 6.5: Normalizirana matrica konfuzije za klasifikaciju u 9 razreda na skupu podataka *DPAv2* klasifikatorom arhitekture **50x198x174x9**.

Slika 6.6 prikazuje usporedbu točnosti klasifikatora s rezultatima dobivenima u radu [27] na skupu podataka *DPAv2*. Iz toga je vidljivo da ovaj klasifikator postiže točnost u rangu s tim rezultatima, ali se iz matrice konfuzije (slika 6.5) moglo zaključiti da se ponaša vrlo loše na problemu klasifikacije s izuzetno velikim udjelom šuma.



Slika 6.6: Usporedba rezultata s [27] na skupu podataka *DPAv2*.

7. Zaključak

Arhitektura mreže jedan je od najvažnijih hiperparametara kod modela temeljenog na umjetnim neuronskim mrežama. Uspješnost modela strojnog učenja ovisi o korištenim hiperparametrima, a jedini način za sigurnu provjeru *kvalitete* neke kombinacije hiperparametara je učenje s tim konkretnim hiperparametrima i usporedba neke mjere uspješnosti. Klasična metoda pronalaska dobrih vrijednosti hiperparametara poput arhitekture mreže je **pretraživanje po rešetki**. Iako najpouzdaniji, ovaj pristup vrlo je spor zbog iscrpnog pristupa pretraživanju područja.

U ovom radu predstavljeni su neuroevolucijski pristupi za optimizaciju hiperparametra arhitekture klasifikatora temeljenog na umjetnim neuronskim mrežama. Pristup temeljen na algoritmu *NEAT* i rijetko povezanim neuronskim mrežama pokazao se lošim u rješavanju problema klasifikacije na skupu podataka *DPAv4*.

Korištenjem generacijskog i segregacijskog genetskog algoritma evoluirane su arhitekture višeslojnih perceptrona te su takvi klasifikatori korišteni na skupovima podataka *DPAv2* i *DPAv4*.

Na vrlo šumovitom skupu podataka *DPAv2*, evoluirani klasifikatori postižu loše rezultate, ali koji su u rangu s drugim rezultatima iz područja. Najbolji evoluirani klasifikator, gledano po točnosti, ima arhitekturu **50x198x174x9**, a postiže točnost od 26.1% te F_1 mjeru 18.1%. Iz matrice konfuzije vidljivo je da takav klasifikator sve tragove klasificira u tri najzastupljenija razreda što ga čini neupotrebljivim u kontekstu DPA.

S druge strane, prilikom klasifikacije na skupu podataka *DPAv4*, klasifikatori evoluirani genetskim algoritmima u kratko vrijeme dolaze do dobrih rezultata. Pretraživanje po rešetki prikazano u 3.3.3 pretraživalo je arhitekture višeslojnog perceptrona s dva skrivena sloja do dimenzija 200x200 s kvantom 5. Taj postupak trajao je 8 dana i zahtijevao 1600 evaluacija. Najbolja točnost klasifikacije – 49.6% – postignuta je mrežom **50x195x30x256**. Pristup temeljen na genetskom algoritmu prikazanom u 4.2 na istom problemu u samo 300 evaluacija (što bi na istom računalu trajalo nešto više od 36 sati) pronalazi arhitekturu koja postiže točnost od 51.1%.

Neuroevolucijski pristupi mogu u kraćem vremenu pretražiti veća područja hiperparametara zbog toga što inherentno eliminiraju lošija rješenja. Iako ne garantiraju pronalazak optimalne arhitekture, u većini praktičnih slučajeva dat će *dovoljno dobre* rezultate.

LITERATURA

[1] Biezl. *CMOS NOR gate*. 2008. URL: https://commons.wikimedia.org/wiki/File:Cmos_nor.svg.

[2] Eric Brier, Christophe Clavier i Francis Olivier. „Correlation power analysis with a leakage model". *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer. 2004, str. 16–29.

[3] Leo Budin i dr. *Operacijski sustavi*. Element, 2010.

[4] François Chollet i dr. *Keras*. 2015. URL: https://github.com/fchollet/keras.

[5] Denis Čaušević. „Modeliranje supojavljivanja semantičkih oznaka uvjetnim slučajnim poljima". Mag. rad. Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva, 2016.

[6] Marko Čupić. *Prirodom inspirirani optimizacijski algoritmi. Metaheuristike.* Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva. 2013. URL: http://java.zemris.fer.hr/nastava/pioa/knjiga-0.1.2013-12-30.pdf.

[7] Marko Čupić, Bojana Dalbelo Bašić i Marin Golub. *Neizrazito, evolucijsko i neuroračunarstvo*. Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva. 2013. URL: http://java.zemris.fer.hr/nastava/nenr/knjiga-0.1.2013-08-12.pdf.

[8] Joan Daemen i Vincent Rijmen. *The design of Rijndael: AES — the Advanced Encryption Standard*. Springer-Verlag, 2002, str. 238.

[9] Charles Darwin. *On the Origin of Species by Means of Natural Selection*. London: Murray, 1859.

[10] DeepAI. *Neural Network*. 2018. URL: https://deepai.org/machine-learning-glossary-and-terms/neural-network (pogledano 14. 5. 2019).

[11] Félix-Antoine Fortin i dr. „DEAP: Evolutionary Algorithms Made Easy". *Journal of Machine Learning Research* 13 (srpanj 2012), str. 2171–2175.

[12] Benedikt Gierlichs i dr. „Empirical comparison of side channel analysis distinguishers on DES in hardware”. *Circuit Theory and Design, 2009. ECCTD 2009. European Conference on*. IEEE. 2009, str. 391–394.

[13] Xavier Glorot i Yoshua Bengio. „Understanding the difficulty of training deep feedforward neural networks”. *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010, str. 249–256.

[14] Xavier Glorot, Antoine Bordes i Yoshua Bengio. „Deep sparse rectifier neural networks”. *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. 2011, str. 315–323.

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

[16] Quasar Jarosz. *Diagram of neuron*. 2009. URL: `https://commons.wikimedia.org/wiki/File:Neuron_Hand-tuned.svg`.

[17] k0resh. *Power analysis basics*. 2014. URL: `https://research.kudelskisecurity.com/2014/07/16/power-analysis-basics/`.

[18] Diederik P. Kingma i Jimmy Ba. „Adam: A method for stochastic optimization”. *arXiv preprint arXiv:1412.6980* (2014).

[19] Karlo Knežević. „Combinatorial Optimization in Cryptography”. *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. Svibanj 2017, str. 1324–1330. DOI: `10.23919/MIPRO.2017.7973628`.

[20] Paul Kocher, Joshua Jaffe i Benjamin Jun. „Differential power analysis”. *Annual International Cryptology Conference*. Springer. 1999, str. 388–397.

[21] David Kriesel. *A Brief Introduction to Neural Networks*. 2007. URL: `http://www.dkriesel.com`.

[22] Sean Luke i Liviu Panait. „Fighting bloat with nonparametric parsimony pressure”. *International Conference on Parallel Problem Solving from Nature*. Springer. 2002, str. 411–421.

[23] Leon Luttenberger. *Generiranje opisa slika primjenom arhitekture LSTM*. 2018.

[24] Stefan Mangard, Elisabeth Oswald i Thomas Popp. *Power analysis attacks: Revealing the secrets of smart cards*. Sv. 31. Springer Science & Business Media, 2008.

[25] Gabor Pete. *Sketch of an SPN*. 2009. URL: `https://commons.wikimedia.org/wiki/File:SubstitutionPermutationNetwork2.png`.

[26] Stjepan Picek. *Applications of Evolutionary Computation to Cryptology*. Uitgeverij BOXPress, 2015.

[27] Stjepan Picek i dr. „On the performance of convolutional neural networks for side-channel analysis". *International Conference on Security, Privacy, and Applied Cryptography Engineering*. Springer. 2018, str. 157–176.

[28] Philip Bradfield; Steve Potter. *Edexcel International GCSE (9-1) Biology Student Book*. Pearson Education Limited, 2017.

[29] Claude E. Shannon. „A mathematical theory of cryptography". *Mathematical Theory of Cryptography* (1945).

[30] Kenneth O. Stanley i Risto Miikkulainen. „Evolving neural networks through augmenting topologies". *Evolutionary computation* 10.2 (2002), str. 99–127.

[31] Jan Šnajder i Bojana Dalbelo Bašić. *Strojno učenje*. Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva. 2014. URL: https://www.fer.unizg.hr/predmet/su/materijali.

[32] TELECOM ParisTech SEN research group. *DPA Contest (2^{nd} edition)*. 2009 – 2010. URL: http://www.DPAcontest.org/v2/.

[33] TELECOM ParisTech SEN research group. *DPA Contest (4^{th} edition)*. 2013 – 2014. URL: http://www.DPAcontest.org/v4/.

[34] Darrell Whitley. „A genetic algorithm tutorial". *Statistics and computing* 4.2 (1994), str. 65–85.

Optimizirana arhitektura klasifikatora temeljenog na umjetnim neuronskim mrežama u domeni implementacijskih napada na kriptografske uređaje

Sažetak

U diplomskom radu opisan je postupak optimizacije arhitekture klasifikatora temeljenog na umjetnim neuronskim mrežama u domeni implementacijskih napada na kriptografske uređaje koristeći tehnike evolucijskog računarstva. Objašnjen je implementacijski napad analizom diferencijalne potrošnje električne energije kriptografskog uređaja te primjena metoda strojnog učenja u istome.

Implementirani su algoritam *NEAT* te dvije varijante genetskog algoritma – generacijski genetski algoritam i genetski algoritam sa segregacijom po vrstama. Dobiveni rezultati uspoređeni su s rezultatima prikazanim u Picek i dr. [27]. Algoritam *NEAT* se nije pokazao uspješnim na problemima klasifikacije, dok su obje varijante genetskog algoritma davale rezultate usporedive s prijašnjim radovima u području.

Ključne riječi: kriptografija, DPA, SCA, strojno učenje, evolucijsko računarstvo, neuroevolucija, genetski algoritam

Optimising architectures of artificial neural network classifiers in the domain of side-channel cryptographic attacks

Abstract

The thesis describes approaches for optimising architectures of artificial neural network classifiers in the domain of side-channel cryptographic attacks using evolutionary computation techniques. Differential power analysis attack is explained, along with a way for the utilisation of machine learning methods.

NEAT and two variations of the genetic algorithm – generational genetic algorithm and genetic algorithm with species segregation – are implemented and their performance is benchmarked. The results are compared with the results shown in Picek et al. [27]. *NEAT*'s performance at classification is subpar at best, while both approaches with the genetic algorithm yield the results in line with others.

Keywords: Cryptography, DPA, SCA, Machine Learning, Evolutionary Computation, Neuroevolution, Genetic Algorithm