SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 4832

Klasifikacija srčanih bolesti na temelju EKG signala uz pomoć strojnog učenja podržanog evolucijskim računarstvom

Danijel Pavlek

Zagreb, lipanj 2017.

Zahvaljujem mentoru, prof.dr.sc. Domagoju Jakoboviću (kralju genetskog programiranja) na savjetima, beskonačnoj količini strpljenja i mogućnosti samostalnog odabira teme završnog rada, kao i obitelji i prijateljima na podršci. Također, veliko hvala Karlu Kneževiću, mag.ing.comp., na iznimno korisnim sugestijama tijekom izrade i doc.dr.sc. Alanu Joviću na pomoći kod prikupljanja i oblikovanja testnih uzoraka.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA ZAVRŠNI RAD MODULA

Zagreb, 5. ožujka 2017.

ZAVRŠNI ZADATAK br. 4832

Pristupnik: **Danijel Pavlek (0036486877)**
Studij: Računarstvo
Modul: Računarska znanost

Zadatak: **Klasifikacija srčanih bolesti na temelju EKG signala uz pomoć strojnog učenja podržanog evolucijskim računarstvom**

Opis zadatka:

Opisati problem klasifikacije EKG signala u medicinskoj dijagnostici. Ostvariti programski sustav za klasifikaciju signala uz pomoć neuronskih mreža i stroja s potpornim vektorima. Ostvariti i komentirati rezultate modela učenih evolucijskim algoritmima. Ocijeniti učinkovitost ostvarenih metoda na dostupnim podacima. Posebnu pažnju posvetiti mogućnostima u oblikovanju ulaza modela te učinkovitosti modela u ovisnosti o algoritmima učenja. Radu priložiti izvorne tekstove programa, dobivene rezultate uz potrebna objašnjenja i korištenu literaturu.

Zadatak uručen pristupniku: 10. ožujka 2017.
Rok za predaju rada: 9. lipnja 2017.

Mentor:

Izv. prof. dr. sc. Domagoj Jakobović

Predsjednik odbora za završni rad modula:

Prof. dr. sc. Siniša Srbljić

Djelovođa:

Doc. dr. sc. Tomislav Hrkać

SADRŽAJ

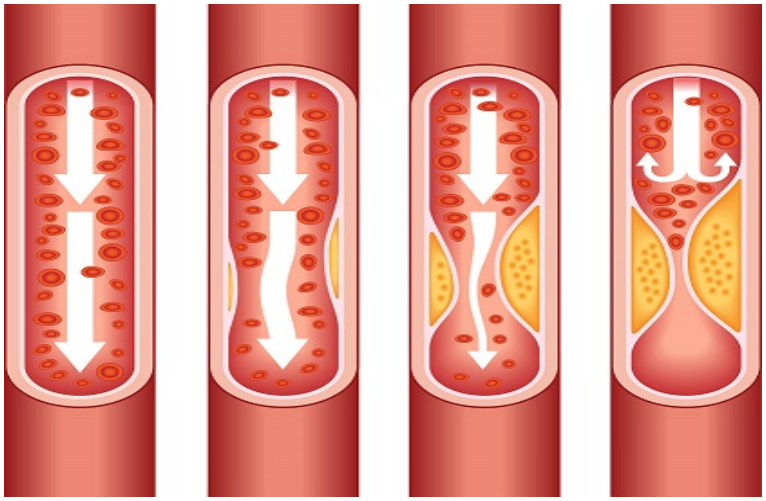
1. Uvod

Srčani udar glavni je uzrok smrti u svijetu sa 17.3 milijuna žrtava godišnje, što ga čini smrtonosnijim od AIDS-a i raka zajedno. Loše prehrambene navike, premalo tjelesne aktivnosti, kao i pušenje te konzumiranje prevelike količine alkohola dovode do lošeg stanja u organizmu. Velik problem sa time imaju razvijene zemlje, pa su tako u Sjedinjenim Američkim Državama kardiovaskularne tegobe glavni uzrok smrti sa 375000 umrlih godišnje. Veliku ulogu u toj priči ima i stres, koji doduše ne utječe na kolesterol, već sužuje krvne žile. Sve te pojave sustavno uzrokuju probleme u kardiovaskularnom sustavu. Postoji mnogo vrsta takvih bolesti. Neke ni ne moraju biti razlog lošeg stila života, nego urođenih nepravilnosti koje se javljaju u samoj građi srca. [1]

Slika 1.1: Najčešći uzroci smrti, podaci iz 2013.g.

Živimo u vremenu u kojem nam računarstvo pomaže u raznim aspektima života, pa tako i u medicini. Umjetna inteligencija grana je računarstva koja se bavi proučavanjem algoritama pomoću kojih je na temelju poznatih uzoraka moguće implementirati detekciju nekih ovisnosti koje su čovjeku često prekomplicirane za shvatiti, posebice zbog toga što je nerijetko ta ovisnost funkcija vrlo velikog broja varijabli. Cilj ovog

završnog rada je realizirati programsku potporu za čim efikasniju klasifikaciju nekih srčanih tegoba. Postoji više metoda pomoću kojih se ocjenjuje stanje srčanog mišića. Neke od njih su elektrokardiogram, ultrazvuk i analiza sastava krvi. U ovom radu razmatra se uočavanje nepravilnosti u obliku i frekvenciji električnih impulsa koje proizvodi srce tijekom rada (EKG). Kao što je prethodno navedeno, uzorci će imati važnu ulogu pri realizaciji programa zato što se neka vrsta anomalije vrlo često može povezati sa točno određenom bolešću.



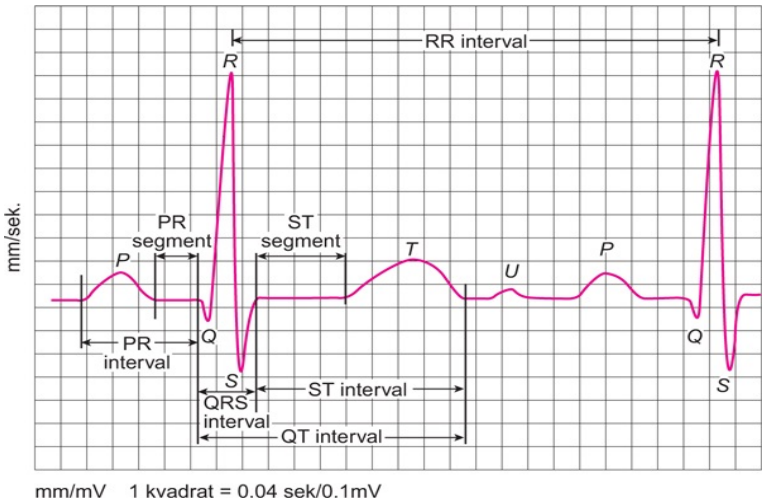
Slika 1.2: Štetne nakupine masnoća u žilama

U ovom radu opisani su načini raspoznavanja lošeg stanja srca na temelju izmjerenog signala pomoću algoritama strojnog učenja. U poglavlju o strojnom učenju razmotrene su neuronske mreže i učenje njihovih parametara pomoću algoritma propagacije greške unazad. Zatim je dan kratak osvrt na stroj s potpornim vektorima i generalne probleme koje susrećemo u učenju modela. U poglavlju o evolucijskom računanju opisani su stohastički algoritmi s naglaskom na genetske algoritme i razvoj topologije neuronske mreže i njezinih parametara. Potom su razrađene mogućnosti razvoja neuronske mreže sa fiksnom strukturom i parametara jezgri SVM-a, kao i važnost dobrog odabira vjerojatnosti za pojedinu operaciju u genetskom algoritmu. Na kraju su prikazani rezultati neuronske mreže koja je učena genetskim algoritmom i evolucijskom strategijom, kao i rezultati algoritma NEAT i stroja s potpornim vektorima.

2. Struktura EKG signala

2.1. EKG signal

EKG (elektrokardiogram) je signal koji prikazuje ovisnost električne aktivnosti srca o vremenu. Taj koncept otkrio je 1901.g. Willem Einthoven kako bi 1924.g. za svoje otkriće bio nagrađen Nobelovom nagradom za medicinu. Električni potencijal između pozitivne i negativne elektrode moguće je izmjeriti na dvanaest različitih pogleda, od kojih su 6 transverzalni, a 6 horizontalni. Prema konvenciji, EKG zapis podijeljen je na P–val, PR–interval, QRS–kompleks, QT– interval, ST–segment, T–val i U–val. [14]



Slika 2.1: Struktura EKG signala

2.2. Značajke EKG-a kod infarkta

Pod pojmom infarkt stručnjaci podrazumijevaju prestanak dotoka krvi u neki organ. U kontekstu ovog rada najopasnija trauma od navedenih je svakako infarkt miokarda,

poznatiji kao srčani udar. Do njega dolazi začepljenjem koronarnih arterija, krvarenjem iz stjenke krvnih žila, nekad čak i jak udarac u središte prsa. Ako se infarkt preživi, dolazi do slabljenja srčanog mišića i stvaranje tvorevine poput ožiljka na mjestu koje je pogođeno. Važno je uočiti nepravilnosti koje se javljaju prije samog udara. Na EKG signalu on se očituje povišenom ili sniženom krivuljom na intervalu ST i pojavom patoloških Q valova. Ti valovi mogu se pojaviti i nakon traume, a tada vrlo rijetko nestaju. Interpretiraju kao nestanak električne aktivnosti na dijelu srca pa time dolazi do veće razlike potencijala između elektroda budući da su obje bile na visokom.

2.3. Mogući pristupi rješavanju problema klasifikacije bolesti

Postoji više načina za detekciju bolesti srca. Prvi način je analiza srčanih otkucaja uzimanjem određenog broja uzoraka iz samog signala. Izuzev ovog pristupa, može se izgraditi ekspertni sustav. To je sustav koji koristi već poznate informacije iz specifičnog područja (u ovom slučaju medicine) i na temelju istih rješava probleme klasifikacije. Ovo može biti bolje od uzorkovanja zato što bi model prilikom uzorkovanja imao dosta ulaza, čemu se može doskočiti smanjenjem frekvencije uzorkovanja. U domeni ovog rada, zbog nedostatka značajki, implementirana je upravo metoda uzimanja već izmjerenog EKG signala i detekcija koristeći kao ulaze njegove vrijednosti u ovisnosti o vremenu (ekstrakcija značajki vrlo je složen postupak i može biti tema doktorske disertacije). [6]

3. Strojno učenje

Strojno učenje grana je umjetne inteligencije koja se bavi ostvarenjem algoritama za rješavanje čovjeku trivijalnih zadataka. Postoji mnogo problema iz stvarnog svijeta koji se mogu shvatiti kao funkcija ili smještanje u razrede (kasnije će biti korišteni izrazi regresija i klasifikacija, respektivno). Nadalje, vrste problema koje se rješavaju algoritmima strojnog učenja dijelimo najčešće na dvije skupine: računalni vid i obrada prirodnog jezika, premda je u zadnje vrijeme obrada zvuka i govora postala vrlo popularna, posebice kad govorimo o generiranju sekvenci. Pod računalnim vidom podrazumijevamo probleme koji se svode na obradu slike (npr. prepoznavanje i klasifikacija prijeloma na temelju rendgenske snimke), dok su programi za otkrivanje raspoloženja na društvenim mrežama iz poruka ili detekciju plagijata na tekstu područje obrade prirodnog jezika. [5]

3.1. Nadzirano i nenadzirano učenje

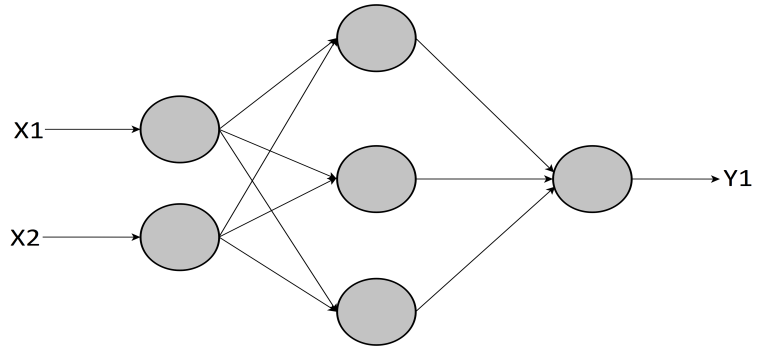
Nadzirano učenje u procesu koristi primjerke za učenje. To su poznate informacije koje predstavljamo vektorima: svaki se uzorak sastoji od niza ulaza i očekivanih izlaza. Ulazi su varijable o kojima ovisi izlaz klasifikatora. Izlaz je niz vrijednosti koje se očekuju kao rezultat za specifični ulazni vektor. Konkretno, u ovom radu ulazi će biti N uniformno odabranih točaka iz EKG signala. Izlazi se mogu formirati na dva načina: ako imamo M klasa, tj. bolesti koje se pokušavaju detektirati, onda možemo izlaze tretirati kao M-dimenzijski vektor gdje će r-ta komponenta imati vrijednost 1, a ostale 0 ako se radi o bolesti čiji je izlaz na r-tom mjestu u izlaznom vektoru.
S druge strane, nenadzirano učenje nema označene uzorke: imamo jedino podatke, tj. ulaze, i na temelju tih podataka želimo shvatiti kako su oni smješteni u vektorskom prostoru i labelirati ih, odnosno konstruirati izlazni vektor. Takav postupak nazivamo grupiranjem. Drugi primjer je izgradnja pravila pomoću poznatih podataka, što nazivamo asocijacijom. Ima mnogo algoritama u domeni nenadziranog učenja, a neki od njih su Kohonenov SOM (samoorganizirajuća mreža) i algoritam širećeg neuronskog

plina. Budući da su podaci o već izmjerenim EKG signalima dostupni i označeni, nepotrebno je raditi grupiranje. [5]

3.2. Neuronske mreže

3.2.1. Općenito

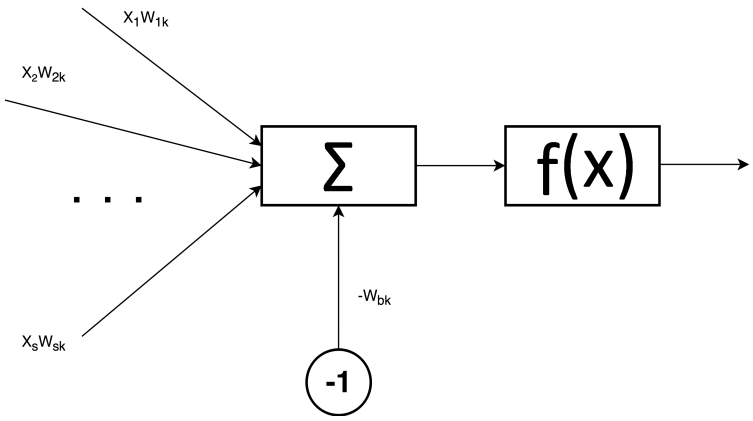
Neuronske mreže matematički su modeli koji u domeni umjetne inteligencije rješavaju vrlo složene probleme. Probleme koje rješavamo neuronskim mrežama najčešće je vrlo teško, a i u dosta slučajeva nemoguće rješiti egzaktnim algoritmima u realnom vremenu. Kao što je slučaj u mnogim algoritmima iz umjetne inteligencije, i one su inpirirane prirodnim procesom. Riječ je naravno, o mozgu. On se sastoji od mnoštva neurona koji su međusobno povezani živčanim vezama koje nazivamo sinapsama. Dakle, kad se neki neuron pobudi, on generira električni impuls koji putuje sinapsom do ostalih neurona i tako se stvara namjera da tijelo napravi neku akciju. Vrlo je važno napomenuti da osim strukture, neuronskoj mreži i ljudskom mozgu slično je da uče temeljem pokušaja i pogrešaka. One tokom učenja isprva rade vrlo velike greške, međutim, kako proces učenja napreduje, tako mreža poboljšava svoje sinapse. Učenje traje toliko dugo dok se mreža ne isprofilira u dovoljno generalizirani model koji obavlja željenu zadaću. Preciznost se u ovom kontekstu odnosi na uspješnost obavljanja zadaće, dok sposobnost generalizacije definira mogućnost mreže da temeljem naučenog, izračuna nešto nikad viđeno, a ne da generira sitnu grešku samo za mali broj testova. Inače, neuronska mreža aproksimira neku vrlo složenu funkciju sa mnoštvom varijabli, a najčešće i više izlaza. Sastavni su dijelovi neuroni i sinapse, koji su na slici ispod prikazani krugovima, odnosno strelicama.



Slika 3.1: Unaprijedna potpuno povezana mreža

Kao što se može uočiti, struktura je izgrađena slojevito: neuroni su smješteni u stupce. Neuronska mreža može imati više skrivenih slojeva. Mreže gdje je svaki ne-

uron povezan sa svakim iz sljedećeg sloja, kao što je slučaj u gornjoj slici, nazivamo unaprijednim slojevitim neuronskim mrežama. One su pogodne za široki raspon problema, dok za potrebe učenja nekih funkcija koje ovise o vremenu koristimo povratne modele, primjerice prepoznavanje osoba u videozapisima.



Slika 3.2: Model neurona

Mreža računa izlaznu vrijednost u $i$tom sloju na sljedeći način: prvo se izračuna težinska suma koja je zapravo skalarni produkt vektora ulaza i vektora težina (na prethodnoj je slici ovo prikazano ulazima u operator zbrajanja). Toj sumi se oduzima slobodni član koji se zapravo također optimira u procesu učenja: taj parametar nazivamo slobodnim koeficijentom, ima ga svaki skriveni i izlazni neuron, a njegova je uloga opisana u sljedećem odlomku. Nakon što je suma izračunata, ona prolazi kroz operator koji će nad tom sumom obaviti nelinearno preslikavanje. Postupak se provodi jednostavno od prvog do zadnjeg sloja, a jedina je razlika u tome što ulazni neuroni nemaju bias te ne primjenjuju nelinearnu funkciju nad težinskom sumom, već kao aktivaciju koriste identitet, $f(x) = x$.

3.2.2. Uloga bias neurona

Uloga slobodnog koeficijenta može se shvatiti proučavanjem jednostavne funkcije, npr. sinusne. Ukoliko želimo dobiti punu ekspresivnost, trebamo imati skaliranje po osi x, skaliranje po osi y, kao i pomak po obje osi. Dobivena sinusoida imala bi formu

$$f(x) = Asin(\omega t + \phi) + C,$$

gdje je C upravo taj slobodni parametar koji dodaje isti efekt kao i bias aktivacijskoj funkciji. U implementaciji je zgodno podrazumijevati da postoji jedan neuron koji uvijek ima konstantnu vrijednost na izlazu, a iznosi -1.

3.2.3. Aktivacijska funkcija

Aktivacijska funkcija određuje kako se ulazi preslikavaju na izlaze neurona. Na ulaznom sloju uvijek se koristi aktivacijska funkcija $f(x) = x$, što znači da ulazni neuroni samo proslijeđuju vrijednosti koje je korisnik postavio inicijalno. Drugi slojevi koriste neke specifične aktivacijske funkcije. U praksi se može vidjeti da je to vrlo često funkcija koja može biti razdioba neke slučajne varijable. Dakle, omeđena je sa donje strane sa 0, a s gornje sa 1. Monotono je rastuća. Vrlo često nema prekide, što je izvrsno kad se dotična funkcija mora derivirati. Aktivacijska funkcija koristi se kao indikator koji detektira koji je neuron prisutan, a koji ne. Primjerice, ako želimo da se aktiviraju jedino neuroni koji su na sebi dobili težinsku sumu veću od 1.5, mogli bismo koristiti step funkciju koja ima skok upravo u toj točki. Čest je slučaj da želimo nelinearnu aktivacijsku funkciju, pa da mreža radi s nekom nelinearnošću (kad već imamo težinsku sumu koja je suma produkata realnih brojeva). Kasnije će biti potrebno moći izračunati derivaciju funkcije u nekoj točki, pa bi spomenuta step funkcija zakazala u točki prekida budući da tamo derivacija teži u beskonačno, tj. nedefinirana je. Najpoznatiji primjer je svakako sigmoidalna aktivacijska funkcija, $f(x) = \frac{1}{1+e^{-x}}$.

3.2.4. Učenje numeričkim postupcima - algoritam backpropagation

Algoritam propagacije greške unazad postupak je učenja težina neuronske mreže čija se ideja temelji upravo na greški koju mreža generira prilikom izračunavanja. Naime, prilikom inicijalizacije, sve vrijednosti težina se postave na slučajno odabrane brojeve, najčešće iz jednolike razdiobe. Zatim slijedi niz iteracija algoritma gdje se u svakoj iteraciji napravi propagacija unaprijed i zatim jedan korak učenja, propagacija unazad. U jednoj epohi to se napravi koristeći mnoštvo testnih uzoraka. Na kraju epohe se ocjenjuje koliko dobro mreža obavlja danu zadaću, provodeći mjerenje greške na skupu uzoraka koji nije korišten za učenje. Ima više funkcija za aproksimaciju greške, a najpoznatija je srednje kvadratno odstupanje (tzv. mean squared error), koje je pogodno zato što veće odstupanje znači drastično veću grešku:

$$E = \frac{1}{2N} \sum_{i=1}^{N} \sum_{j=1}^{O} (t_j - y_j)^2,$$

gdje je t očekivani izlaz, a y izračunati. Broj uzoraka je označen sa N, a broj neurona izlaznog sloja sa O. Pogreška je funkcija težina, a učenje se provodi korekcijom istih sitnim koracima. To je moguće ostvariti gradijentnim spustom: pomaci imaju suprotni

predznak od derivacije kako bi algoritam išao prema minimumu. Globalni minimum pronalazimo pomoću derivacije prethodne funkcije po težini w_{ij}. Izraz za korekciju težine je:

$$w_{ij} = w_{ij} + \Delta w_{ij},$$

gdje je Δw_{ij} iznos kojeg težini treba dodati, a do njega se dolazi pomoću izraza:

$$\Delta w_{ij} = \eta \delta y_i,$$

gdje je η stopa učenja i obično iznosi $\frac{1}{2N}$, y_i je izlaz neurona, a δ iznos gradijenta, koji se računa kao:

$$\delta_i = (t_i - y_i) f'(x_i)$$

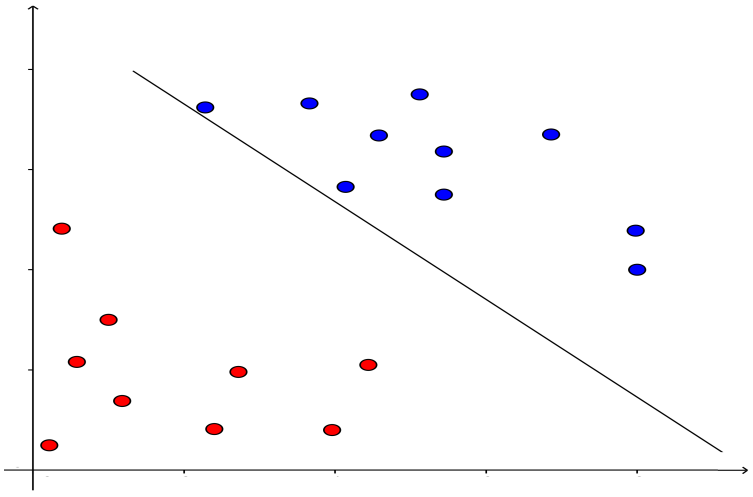
za izlazni sloj, dok za ostale slojeve vrijedi:

$$\delta_j = f'(x_j) \sum_{i=1}^{K} \delta_i w_{ij}$$

[7]

3.3. Stroj s potpornim vektorima

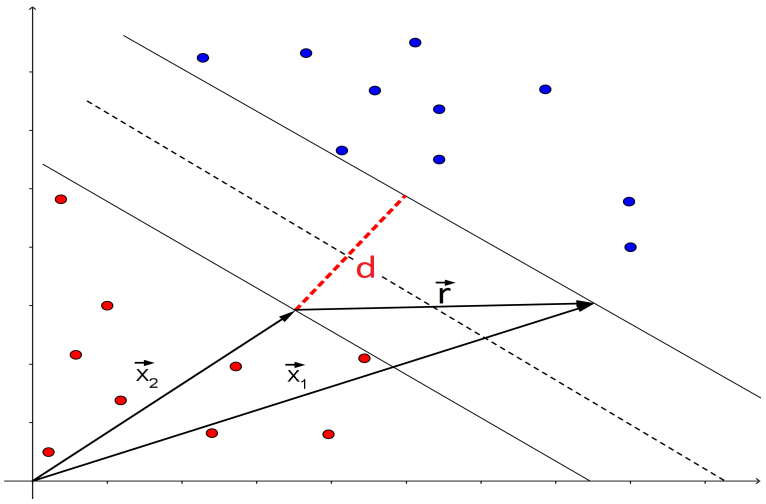
Problem koji se javlja kod neuronskih mreža je lako zaglavljenje u lokalnim minimumima. Njih vrlo često ima mnogo, pogotovo ako mreža ima velik broj težina, dakle, prostor pretraživanja ima mnogo dimenzija. Nadalje, izabrati optimalnu arhitekturu mreže može biti kompliciranо. Ideju strojeva s potpornim vektorima su iznijeli znanstvenici Vapnik i Chervonenkis, a njom se pokušava riješiti klasifikacija koja tijekom učenja u obzir uzima i udaljenost separacije od najbližih uzoraka iz obje klase, što je prikazano na slici ispod:



Slika 3.3: Problem kod klasifikacije perceptronom

3.3.1. Linearni SVM

Stroj potpornih vektora je matematički model koji se najčešće koristi u klasifikaciji, dok je rijeđe zastupljen u zadaćama regresije. Problem kojeg želimo riješiti jest svrstavanje nekog uzorka u jedan od dva razreda (jednostavnom tehnikom moguće je proširiti koncept na više njih – svaki uzorak se svrstava u dvije grupe – ili pripada nekom razredu ili ne). Kod strojeva vektora potpore podaci su predstavljeni sa N značajki, što nam govori da je svaki uzorak zapravo N-dimenzijski vektor. Algoritam možemo konstruirati tako da on pronalazi hiperravninu koja razdvaja klase. Budući da njih ima mnogo, uzima se ona koja ima najveću moguću udaljenost od najbližih uzoraka s obje strane.[15]



Slika 3.4: Problem maksimalne margine

Izvod algoritma počinje definiranjem pravila za svrstavanje uzorka u jednu, odnosno u drugu klasu: prvo uzimamo vektor **w** sa nepoznatim dimenzijama koji je okomit na margine. Definiramo pravilo svrstavanja nepoznatog uzorka **w** pomoću konstante w_0 te skalarnog produkta vektora težina, **w**, sa tim uzorkom **x** koji se nalazi unutar margina:

$$\mathbf{w} \cdot \mathbf{x} + w_0 \geq 0$$

Nadalje, definiramo strože pravilo odluke:

$$\mathbf{w} \cdot \mathbf{x}_+ + w_0 \geq 1$$

$$\mathbf{w} \cdot \mathbf{x}_- + w_0 \leq -1,$$

odnosno:

$$\mathbf{w} \cdot \mathbf{x}_+ \geq 1 - w_0$$

$$\mathbf{w} \cdot \mathbf{x}_- \leq -(1 + w_0)$$

Radi jednostavnijeg proračuna, uvodimo novu varijablu y za koju vrijedi:

$$y(\mathbf{x}) = \begin{cases} 1, \mathbf{x}_+ \\ -1, \mathbf{x}_- \end{cases}$$

što kombinacijom sa početnim pravilima vodi do sljedeće jednakosti:

$$y(\mathbf{x})\left[\mathbf{w}\cdot\mathbf{x} + w_0\right] \geq 0$$

Ako sada uzmemo takva dva vektora koji predstavljaju uzorke (jedan uzorak pripada jednoj, a drugi drugoj klasi: $\mathbf{x}_{g+}$ i $\mathbf{x}_{g-}$) i nalaze se točno na margini, možemo doći do udaljenosti između margina: u tome nam upravo može pomoći razlika tih dvaju vektora: $\mathbf{r} = \mathbf{x}_{g+} - \mathbf{x}_{g-}$. Razlika će, međutim, biti vektor koji ne mora, a najčešće i nije okomit na granice. Budući da je udaljenost između dva pravca definirana kao duljina dužine koja ih spaja i okomita je na njih, tu duljinu možemo izraziti na sljedeći način:

$$d = \mathbf{r}\frac{\mathbf{w}}{\|\mathbf{w}\|} = (\mathbf{x}_{g+} - \mathbf{x}_{g-})\frac{\mathbf{w}}{\|\mathbf{w}\|} = \frac{\mathbf{w}\mathbf{x}_{g+} - \mathbf{w}\mathbf{x}_{g-}}{\|\mathbf{w}\|} = \frac{1 - w_0 + 1 + w_0}{\|\mathbf{w}\|} = \frac{2}{\|\mathbf{w}\|}$$

Na početku izvoda uzeli smo **w** i za njega definirali da je okomit na granice: u gornjoj jednadžbi iskoristili smo tu pretpostavku i pomnožili razliku jediničnim vektorom $\frac{\mathbf{w}}{\|\mathbf{w}\|}$ i konačno dobili skalar koji predstavlja konkretnu vrijednost tražene udaljenosti. Budući da tražimo onu hiperravninu koja ima najveću moguću udaljenost između dva najbliža uzorka iz obje klase, trebamo maksimizirati gornji izraz. Zbog toga što je d obrnuto proporcionalan $\|\mathbf{w}\|$, nastojimo pronaći minimalnu normu vektora **w**, kvadriranu i pomnoženu sa 0.5. Uz minimizaciju tog parametra, moramo uzeti u obzir uvjet sa hipotezom.

Kod minimizacije funkcije koja postavlja ograničenja koristimo Lagrangeove multiplikatore:

$$L = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n} \alpha_i[y_i(\mathbf{w}\mathbf{x}_i) + w_0] - 1$$

Sada možemo primjeniti ovu ideju na problem minimizacije parametra $\|\mathbf{w}_i\|$. Potrebno je derivirati ovaj izraz po varijablama **w** i w_0. Dobiva se:

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i$$

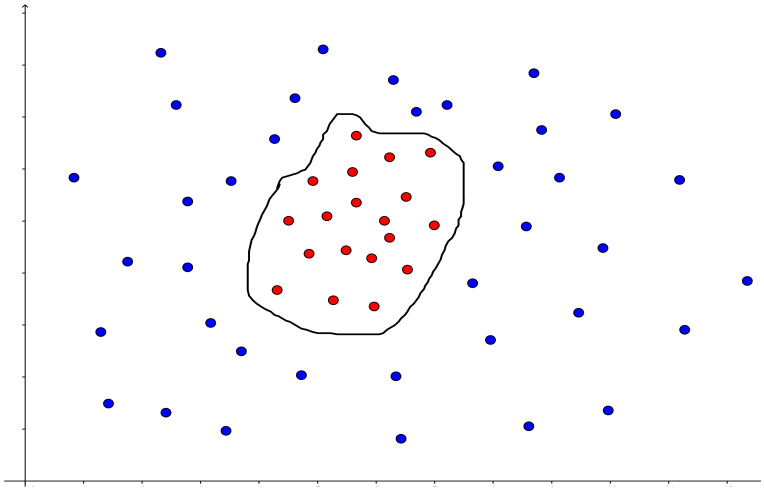
$$\frac{\partial L}{\partial w_0} = -\sum_{i=1}^{N} \alpha_i y_i$$

Prethodni izrazi za parcijalne derivacije izjednačene s nulom te izraz za odlučivanje kombiniran s pomoćnom varijablom y uvrštava se u funkciju L kako bi konačan izraz glasio:

$$L^{'} = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \mathbf{x}_j$$

3.3.2. Nadogradnja SVM-a za linearno neodvojive probleme

Poteškoće se javljaju ako uzorci nisu linearno odvojivi, kao što je prikazano na slici ispod:



Slika 3.5: Nelinearni SVM

U tu svrhu uvodi se ideja transformacije uzoraka u vektorski prostor sa više dimenzija. Ta transformacija se vrši funkcijom $\phi(\mathbf{x})$. Konačni izraz kod nelinearnih strojeva s potpornim vektorima je vrlo sličan onome kod linearnih, štoviše, jedina je razlika što sad u izrazu umjesto identiteta imamo neko preslikavanje. Potencijalni problem može biti računalna složenost izračunavanja tih funkcija. Nadalje, memorijska složenost može doći do izražaja ako se koristi preslikavanje $\phi : \mathbb{R}^N \to \mathbb{R}^M$ gdje je M velik broj. Povrh svega, ne znamo koliko će ta funkcija biti kvalitetna u algoritmu. Ideja za rješavanje ovih poteškoća leži u činjenici da skalarni produkt $\phi(\mathbf{x})\phi(\mathbf{y})$ možemo zapisati kao funkciju ovisnu o vektorima $\mathbf{x}$ i $\mathbf{y}$:

$$k(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x})\phi(\mathbf{y})$$

Ova ideja naziva se jezgrenim trikom, a funkcija k jezgrenom funkcijom. [9]

3.3.3. Jezgrene funkcije

Postoji više vrsta jezgrenih funkcija s obzirom na način na koje one ovise o ulaznim parametrima. Prema toj raspodjeli razlikujemo skalarne produktne jezgre, $f(\mathbf{x}, \mathbf{y}) = f(\mathbf{x} \cdot \mathbf{y})$, radijalne bazne funkcije, $f(\mathbf{x}, \mathbf{y}) = f(\|\mathbf{x} - \mathbf{y}\|)$ i stacionarne jezgre, $f(\mathbf{x}, \mathbf{y}) = f(\mathbf{x} - \mathbf{y})$. Popis nekih jezgara je naveden u sedmom poglavlju, gdje je istaknuta efikasnost SVM-a sa specifičnim funkcijama.
U prethodnom poglavlju je definirano koja funkcija može biti kernel, a koja ne, pa je u skladu s tim moguće kombinirati više jezgrenih funkcija i tako formirati ispravnu jezgru. Primjeri takvih ispravnih kompozicija su:

$$K(\mathbf{x}, \mathbf{y}) = Ck(\mathbf{x}, \mathbf{y})$$

$$K(\mathbf{x}, \mathbf{y}) = f(\mathbf{x})f(\mathbf{y})k(\mathbf{x}, \mathbf{y})$$

$$K(\mathbf{x}, \mathbf{y}) = e^{k(\mathbf{x}, \mathbf{y})}$$

$$K(\mathbf{x}, \mathbf{y}) = k_1(\mathbf{x}, \mathbf{y})k_2(\mathbf{x}, \mathbf{y})$$

$$K(\mathbf{x}, \mathbf{y}) = k_1(\mathbf{x}, \mathbf{y}) + k_2(\mathbf{x}, \mathbf{y})$$

gdje je K funkcija nastala kompozicijom, C realna konstanta, k_i ispravni kernel, a f bilo koja realna funkcija. [2]

3.4. Problemi u strojnom učenju

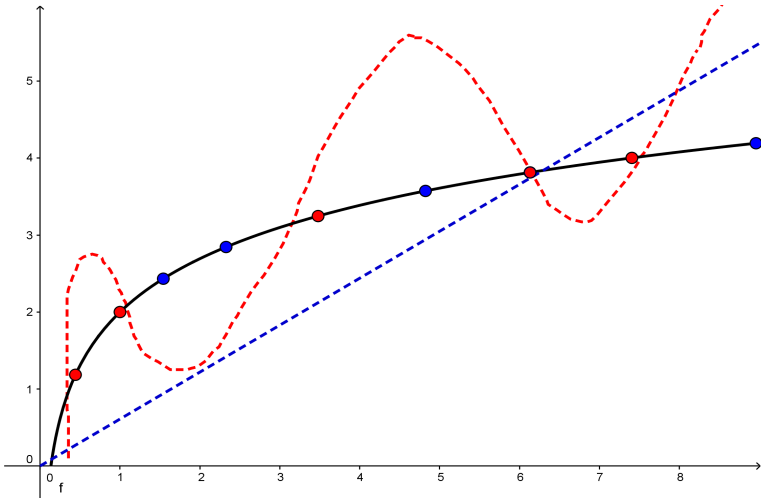
3.4.1. Podnaučenost

U slučaju da ne učimo model kroz dovoljan broj epoha, jasno je kako njegovi parametri neće biti dovoljno naučeni. Također, moguće je da neki model jednostavno ne može naučiti rješavati problem - zgodan primjer je jednostavan XOR problem - ako koristimo algoritam koji ne sadrži nelinearne elemente, uzorci neće moći biti separirani. [4]

3.4.2. Prenaučenost

Prenaučenost je stanje koje se manifestira prejakom prilagodbom samo specifičnim ulazima (testnim uzorcima). Uzmimo sljedeći primjer na razmatranje: studenti i studentice uče neko gradivo na fakultetu kako bi se kasnije mogli lakše zaposliti. Pitanja koja se postavljaju na predavanju (za vježbu) predstavljala bi skup za učenje. Testni skup sadržao bi ispitna pitanja (konkretno, ispit čija pitanja nisu prije viđena), dok bi

se validacija vršila na samom poslu u kojem se koristi naučeno. Budući da živimo u vremenu gdje razvoj informatičkih tehnologija pospješuje protok informacija, za ispitni set možemo gotovo sa sigurnošću reći da ne može ostati skriven. Problem koji se javlja jest učenje na ispitnim uzorcima, što dovodi do specifikacije učenja samo na tim pitanjima. Kasnije, prilikom validacije, do izražaja dolazi nemogućnost prilagodbe na novu, nikad viđenu problematiku. Kažemo da model nema dobru sposobnost generalizacije, što je zapravo cilj učenja, a ne reproduciranje odgovora na poznata pitanja.
U kontekstu klasifikacije/regresije, taj problem se nastoji riješiti odvajanjem skupa testnih uzoraka koji će biti korišteni jedino u svrhu ocjenjivanja koliko je model dobar. Tu grupu nazivamo validacijskim skupom uzoraka. Često je višestruko manja od uzoraka koji se koriste tijekom učenja.



Slika 3.6: Prenaučenost i podnaučenost kod regresije

Slika iznad prikazuje stanje nakon učenja regresije funkcije $f(x) = 2 + ln(x)$ kod dva regresijska modela: crna linija prikazuje funkciju koja se uči, plava iscrtkana linija predstavlja model koji je nedovoljno naučen, dok crvena označuje prenaučen model. Crveni kružići su testni primjerci, a plavi nekorišteni u procesu učenja.

4. Evolucijsko računarstvo

4.1. Općenito

Evolucijsko računanje grana je računarstva koja se bavi proučavanjem algoritama koji su inspirirani stvarnom evolucijom nekog procesa. Ti procesi mogu djelovati nad jednom jedinkom, međutim, češći je slučaj u evolucijskom računarstvu da se promatra skup, tj. populacija jedinki iste vrste. Ideja ovih algoritama je da svaka jedinka predstavlja jedno moguće rješenje nekog problema. Ako se radi o jednoj jedinci, ista se nastoji kroz epohe poboljšati, dok se analogno radi sa skupom jedinki ako se koristi populacijski algoritam.

4.2. Genotip

Genotipom nazivamo prikaz rješenja problema u populaciji koji je razumljiv računalu - primjerice, ako je traženo rješenje neka točka funkcije, zgodno je koristiti vektorski prikaz gdje je svaka dimenzija tog vektora broj u nekoj bazi koji zapravo predstavlja moguće rješenje u npr. decimalnom sustavu. Preslikavanje iz vektora u traženi oblik mora biti takvo da je svako rješenje iz domene moguće prikazati genotipom, a svaki se genotip mora moći dekodirati. Za razliku od genotipa, koji kodira jedinke, fenotip je reprezentacija tog kodiranog kromosoma.

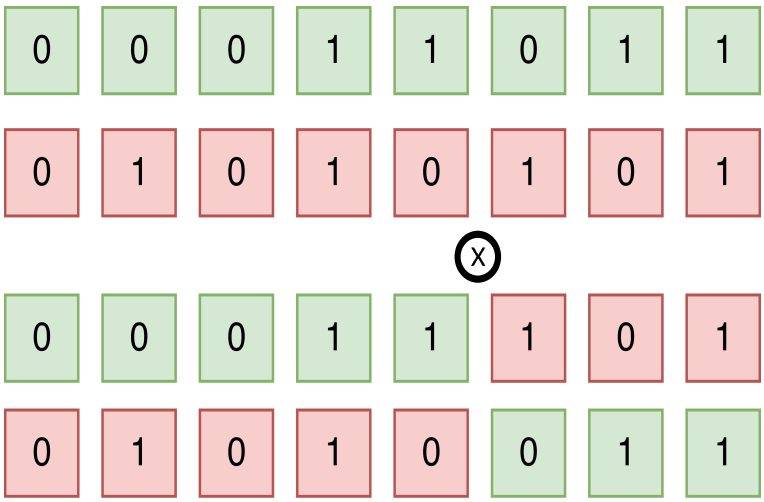
4.3. Operator selekcije

Operator selekcije, kao što sam naziv sugerira, izabire jedinke iz populacije i prenosi ih u sljedeću generaciju. Također, on može izabrati jedinke koje će biti izbačene, ako se radi o eliminacijskom algoritmu. Ovime se zapravo simulira stvarni (biološki) proces preživljavanja – preživjet će samo one jedinke koje se uspješno prilagode okolini. Zato je prirodno podesiti algoritam tako da boljim jedinkama pruži veće šanse da one

budu prenešene u novu populaciju. Kod ovog operatora važno je spomenuti elitizam – koncept koji automatski prenosi najbolju jedinku u novu populaciju. Selekcije se dijele na tri glavne vrste: selekcije ovisne o dobroti jedinke, selekcije koje koriste rangiranje te turnirske.

4.4. Operator križanja

Križanje (engl. crossover) proces je stvaranja novih jedinki čiji geni moraju biti slični svojim roditeljima. U evolucijskim algoritmima (konkretno, kod genetskog algoritma) to se ostvaruje na način da se odaberu dvije jedinke, često proporcionalno dobroti i proglasi ih se roditeljima. Proces križanja uzima dio gena prvog roditelja i kopira ih u dijete, a isto čini sa dijelom gena drugog roditelja. Operacija je uistinu inspirirana prirodom. Postoji više vrsta križanja, ovisno o strukturi genotipa. Nekad on može biti toliko složen da je vrlo kompliciran realizirati ovaj operator, što će biti slučaj kod neuro-evolucije rastućih topologija.

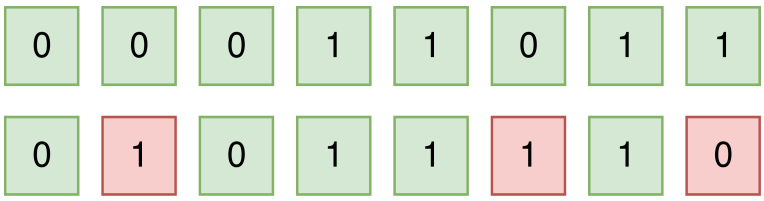


Slika 4.1: Operacija križanja s jednom točkom prijeloma

4.5. Operator mutacije

Mutacija je proces mijenjanja gena, tj. stvaranje detalja kod djeteta koji nisu viđeni na roditeljima. Ovo omogućuje stvaranje raznolikosti između jedinki, pogotovo onih koje imaju zajedničke roditelje. Generalno gledajući, mutacija mijenja gene u kromosomu za mali iznos (npr. u slučaju da je genotip vektor decimalnih brojeva, slučajno bi odabrali neku dimenziju i dodali joj također slučajni broj iz nekog intervala pazeći na granice vektorskog prostora te dimenzije).

Nekad mutacija i križanje mogu biti računalno zahtjevni pa se u svakoj epohi odabire samo jedan od tih dva operatora. Mutacijom se čak može ostvariti generiranje djece - u *hill-climbing* varijanti genetskog algoritma uzima se roditelj i mutacijom se stvaraju vrlo slične jedinke, a najbolja se onda prenosi u sljedeću generaciju i algoritam se vrti dok nije postignut uvjet zaustavljanja.



Slika 4.2: Mutacija

4.6. Problemi kod križanja, mutacije i selekcije

Vrlo je važno da algoritam ima prikladno postavljene parametre koji reguliraju učestalost križanja i mutacije. Učestalost križanja povezana je s preživljavanjem roditelja. Naime, ako postavimo premalu vjerojatnost za dotični parametar, roditeljski kromosomi će imati veću šansu za prijenos u novu populaciju, što potencijalno vodi ka postojanju mnogo sličnih rješenja. Stavljanje premale šanse za mutaciju uzrokovat će to da algoritam počne raditi u jednom uskom prostoru. Recimo da se nakon križanja u populaciji nađu jedinke koje su u području lokalnog minimuma. Mutacija neće doći do izražaja, a operator selekcije odabrat će roditelje proporcionalno njihovoj dobroti (pod hipotezom da je funkcija selekcije izvedena tako da ovisi o samoj dobroti pojedinih jedinki). Dopustimo li da jedinke previše mutiraju, djeca će manje sličiti roditeljima – rad genetskog algoritma će se svesti na nasumično pretraživanje vektorskog prostora funkcije čiji globalni minimum tražimo. Navedene probleme moguće je riješiti traženjem optimalnih parametara algoritma, što se također radi u domeni evolucijskog računanja. Važno je spomenuti kako loša realizacija selekcije također može uzrokovati zaglavljivanje u području lokalnog minimuma: primjerice, izvede li se odabir jedinki koje će preživjeti trenutnu epohu kao direktna selekcija najboljih, algoritam bi se mogao naći u stanju gdje su dvije najbolje jedinke upravo u neželjenom području - dakle, djeca bi se generirala upravo oko tog područja. Stoga se umjesto pohlepnog odabira uzimaju metode koje također lošijim jedinkama daju šanse za preživljavanje, ali proporcionalno dobroti, kao što je tzv. *roulette wheel* selekcija. [10]

4.7. Genetski algoritam

4.7.1. Općenito

Genetski algoritam vrsta je populacijskog algoritma iz domene evolucijskog računarstva čije se djelovanje temelji na stohastičkom modelu gdje se prirodom inspiriranim postupcima skup jedinki nastoji iz epohe u epohu poboljšati. Sličnost sa prirodnim procesima ovaj algoritam pronalazi u parenju i mutaciji - dakle, djeca će naslijediti osobine roditelja i pritom eventualno promijeniti neke značajke. Kao što vrijedi za većinu evolucijskih algoritama, šansa za preživljavanje direktno je vezana uz kvalitetu same jedinke.

4.7.2. Struktura kromosoma

Kromosomom je jedinka populacije čiju vjerojatnost preživljavanja (tj. prijenosa u sljedeću epohu) određuje funkcija dobrote. S obzirom da se genetski algoritmi koriste u problemima minimizacije, odnosno traženja globalnog minimuma funkcije, obično je implementacija takva da će jedinka s manjom dobrotom imati veću šansu za preživljavanje. Ona ima razne atribute koji se koriste u izračunavanju njezine vrijednosti dobrote, dakle, mogu biti predstavljeni cijelim ili decimalnim brojevima, a jedan može biti kodiran jednim ili više gena – nedjeljivih jedinica kromosoma. Atributi u implementaciji mogu biti kodirani na razne načine. Jedan od najčešće korištenih jest kodiranje poljem cijelih brojeva – kasnije će biti korišten naziv diskretni genotip. Dakle, kod je zapravo niz znamenki u nekoj bazi. Manja baza će značiti veći zapis kromosoma, budući da ona obuhvaća manji raspon cijelih brojeva, no jedinke će samim time biti raznolikije, što svakako može biti, a i najčešće je poželjno. Zapis jedinke (genotip) u programskoj izvedbi može biti i složeniji – npr. matrični bi bio zgodan kod rješavanja jedne pozicije u igri križić-kružić, dok bi stablo bilo pogodno kod simuliranja matematičke funkcije koja je u samom algoritmu prikazana varijablama, konstantama i operatorima (genetsko programiranje). Želimo da jedinke (rješenja) predstavljaju neki broj iz domene. Budući da se kao genotip koristi polje brojeva, trebamo imati metodu za konverziju istog u taj broj. Budući da u memoriju ne možemo pohraniti bilo koji realni broj, postojati će neka konačna rezolucija koja bude ograničila broj mogućih rješenja koja se mogu generirati algoritmom. To znači da ćemo, primjerice, želimo li koristiti polje cijelih brojeva duljine 10 u ternarnoj bazi, imati na raspolaganju ukupno 3^{10} brojeva. Ako potom ograničimo prostor pretrage rješenja sa -5 i 8, dolazimo do zaključka da će razmak između dva moguća susjedna rješenja biti $13 \cdot 3^{-10}$, što je

dovoljno velika preciznost. [3] [13]

4.7.3. Križanje kod vektorskog i matričnog genotipa

U ovom radu operator križanja izveden je na način da se odabere k točaka u kromosomu i označi točkama prijeloma. Nakon što se selekcijom odaberu dva roditelja, algoritam iterira od prvog do posljednjeg gena. U prvo dijete počne kopirati gene prvog, a u drugo gene drugog roditelja. Nakon što se dođe do točke prijeloma, postupak se nastavlja obrnutim umetanjem gena: sad drugo dijete prima gene prvog roditelja, a prvo gene drugog, sve dok se ne dostigne sljedeća prijelomna točka, itd. Križanje u matričnoj strukturi izvedeno na analogni način, samo što tada ulogu jednog gena (dimenzije kod vektora) preuzima redak, odnosno stupac.
Osim pristupa u kojem imamo direktno kopiranje gena, možemo ostvariti linearnu kombinaciju uzimajući zbroj umnožaka gena linearnim koeficijentom lambda koji se nalazi u intervalu $[0, 1]$. Prvi će gen biti pomnožen spomenutim koeficijentom, a drugi njemu komplementarnim, a djetetov gen bude zbroj tih dvaju umnožaka:

$$c1 = \lambda p_1 + (1 - \lambda)p_2$$

$$c2 = \lambda p_2 + (1 - \lambda)p_1$$

Na temelju izraza može se zaključiti kako će dijete biti slično nekom od roditelja ako je koeficijent λ blizu 0 ili 1, pa te dvije situacije predstavljaju izostanak operacije križanja. Formula se može zapisati kompaktnije, koristeći matrični zapis:

$$\begin{bmatrix} c_1 & c_2 \end{bmatrix} = \begin{bmatrix} \lambda & 1-\lambda \\ 1-\lambda & \lambda \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix}$$

4.7.4. Mutacija kod vektorskog i matričnog genotipa

Mutacija se u vektorskoj strukturi najčešće vrši komplementiranjem dimenzije: neki element vektora će nakon mutacije poprimiti vrijednost $B - x$, gdje je B baza koda kojim je kodiran genotip, a x sadašnja vrijednost. Time se onemogućuje da neka dimenzija poprimi vrijednost izvan granica. U matričnoj strukturi bira se jedan gen i dodaje mu se slučajni broj, generiran najčešće iz simetričnog intervala (matrični genotip je često memorijski zahtjevan pa se varijable ne kodiraju, već su same po sebi geni kromosoma).

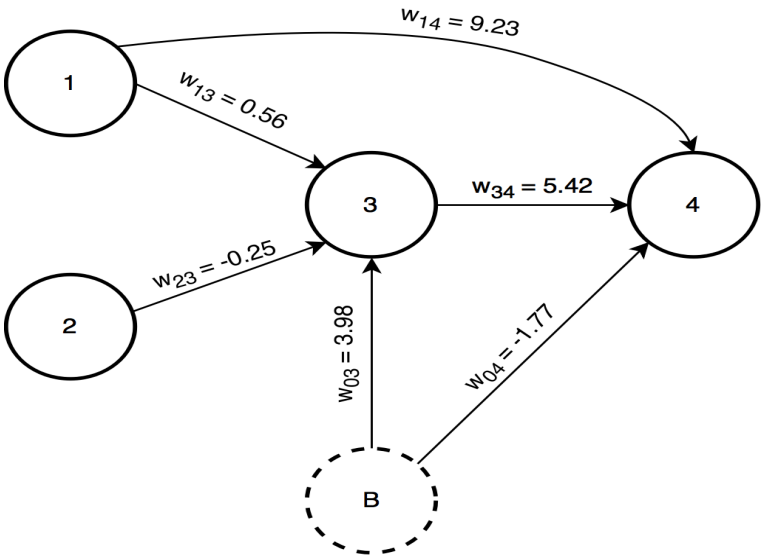
4.8. NEAT (Neuro-evolucija rastućih topologija)

4.8.1. Općenito

Kod rješavanja problema umjetnom neuronskom mrežom prirodno je zapitati se kakva će biti njezina struktura – koliko će mreža imati skrivenih slojeva, koliko će neurona imati svaki sloj te koji će neuron bit povezan s kojim (dakle, moguće je da na kraju završimo s mrežom koja nije potpuno povezana). Ako odaberemo mrežu čija je složenost premala, u kontekstu broja slojeva i neurona u njima, logično je da ona ne bude dovoljno moćna da nauči aproksimirati neku funkciju. S druge strane, preveliko povećanje broja neurona u mreži znači drastično povećanje broja težina koje se trebaju optimirati – dakle, algoritam učenja će presporo učiti mrežu zbog robusnog prostora rješenja kojeg bi trebao pretraživati. Znanstvenici Gomez i Mikkulainen izlažu u znanstvenom radu iz 1999.g. ideju o algoritmu neuro evolucije rastućih topologija (NEAT – neuro evolution of augmented topologies). Cilj algoritma jest generiranje optimalne strukture mreže i pronalaska idealnih vrijednosti težina, iako je moguća kombinacija sa numeričkim postupkom učenja težina. NEAT se temelji na genetskom algoritmu, a najjednostavnija varijanta koristi polje bitova kao genotip. Kod evolucije neuronske mreže, osim problema koji su bili navedeni u poglavlju o oblikovanju skupa težina genetskim algoritmom, javlja se još i sljedeće neželjeno svojstvo: dvije potpuno različite topologije mogu stvarati vrlo sličnu grešku (postojanje više minimuma koji su vrijednošću slični globalnom). Pseudokod algoritma nalazi se na kraju ovog potpoglavlja.

4.8.2. Struktura kromosoma

Kromosom je zapravo neuronska mreža. Sastoji se od genoma koji predstavljaju neurone i sinapse. Ima više načina za kodiranje neuronske mreže, najpoznatiji su vektorski(predstavljene su samo težine, pa je to nedostatak), matrični (dovoljno ekspresivan da predstavi bilo koju strukturu, čak i povratnu) prikaz stablima, rekurzivnim jednadžbama i grafovima, gdje genetsko programiranje može doći do izražaja.



Slika 4.3: Nepravilna arhitektura neuronske mreže

ID : 1	ID : 2	ID : 3	ID : 4
BIAS : 0	BIAS : 0	BIAS : 3.98	BIAS : -1.77
AKTIVACIJSKA FUNKCIJA : LIN	AKTIVACIJSKA FUNKCIJA : LIN	AKTIVACIJSKA FUNKCIJA : SIG	AKTIVACIJSKA FUNKCIJA : TH
PARAMETRI : **LIN(1.0)** SIG(1.0) TH(1.0)	PARAMETRI : **LIN(1.0)** SIG(1.0) TH(1.0)	PARAMETRI : LIN(1.64) **SIG(1.12)** TH(2.83)	PARAMETRI : LIN(0.97) SIG(0.54) **TH(0.91)**
AKTIVAN: DA	AKTIVAN: DA	AKTIVAN: DA	AKTIVAN: DA

INOVACIJSKI BROJ: 0	INOVACIJSKI BROJ: 1	INOVACIJSKI BROJ: 2	INOVACIJSKI BROJ: 3
TEŽINA: 0.56	TEŽINA: -0.25	TEŽINA: 5.42	TEŽINA: 9.23
IZVORIŠNI NEURON: 1	IZVORIŠNI NEURON: 2	IZVORIŠNI NEURON: 3	IZVORIŠNI NEURON: 1
ODREDIŠNI NEURON: 3	ODREDIŠNI NEURON: 3	ODREDIŠNI NEURON: 4	ODREDIŠNI NEURON: 4
AKTIVNA: DA	AKTIVNA: DA	AKTIVNA: DA	AKTIVNA: DA

Slika 4.4: Ekvivalentni kromosom mreži sa slike 4.3

Genotip

Kromosom se sastoji od neuronskih i sinapsnih genoma. To omogućuje manipulaciju nad aktivacijskim funkcijama te prenošenje cijelog skupa težina jednog neurona u drugu neuronsku mrežu.

Neuronski dio genotipa

Neuronski genom predstavlja jedan neuron u neuronskoj mreži. On sadrži indeks pomoću kojeg ga razlikujemo od ostalih neurona u mreži. Nadalje, u njemu je pohranjena

informacija o aktivacijskoj funkciji te njezinim hiperparametrima. Budući da želimo biti u mogućnosti isključiti neuron u svrhu simplifikacije, u neuronske genome je ugrađena zastavica koja to kontrolira. Na kraju, za neurone se može definirati njihova jakost, koja je zapravo razlika u izlazima mreže kad ona normalno funkcionira i kad je promatrani neuron isključen. U slučaju da se prati taj parametar, zgodno je mutaciju realizirati u skladu sa razdiobom koja omogućuje da slabiji neuroni imaju veću šansu za mutaciju.

Sinapsni dio genotipa

Sinapsni genomi se identificiraju na globalnoj razini - suprotno od neuronskih genoma, čiji indeks vrijedi na razini jednog kromosoma. Dakle, algoritam pamti globalni indeks, a kad mutacijom dodamo novu sinapsu, njezina će vrijednost biti globalni indeks uvećan za jedan. Nakon toga globalni indeks poprima vrijednost novonastalog genoma. U sinapsu je također pohranjena vrijednost, kao i informacija o njezinoj uključenosti. Dodatno, ona sadrži indekse neurona koje spaja.

4.8.3. Izvedba križanja

Križanje u algoritmu NEAT je vrlo intuitivno: dakle, operator djeluje tako da se u dijete prenesu neuroni, odnosno težine roditelja. Tu dolazimo do problema: koji geni iz prvog i drugog roditelja mogu istovremeno sudjelovati u operaciji? Drugim riječima, smije li operator birati između dva različita (različito u ovom kontekstu označuje nekorespodentnost, odnosno neurone koji se u različitim mrežama ne nalaze na istom mjestu - npr. nema smisla da operator križanja bira između ulaznog neurona u prvoj mreži i skrivenog u drugoj) neurona ili težine proslijediti ih u dijete? To, dakako, nema smisla. Zbog toga se križanje lakše može provesti na razini sinapsa. Globalni inovacijski broj koji indeksira sinapse može donekle pomoći tako da se sinapse iz prvog i drugog roditelja sortiraju i poredaju jedna iznad druge. Tada je vizualno jasno koje bismo sinapse smjeli tretirati kao korespodentne. Problem se javlja u tome što je nemoguće detektirati kad se dodala težina koja već ima svoj par u nekoj drugoj mreži. To se može pokušati rješiti prenošenjem svih težina nekog neurona i eventualnim deaktiviranjem onih koje ne bi bile adekvatno spojene zbog nedostatka nekog neurona u djetetu.

4.8.4. Izvedba mutacije

Kod algoritma NEAT imamo više vrsta operacija koje čine mutaciju. One mogu promijeniti strukturu mreže dodavanjem/isključenjem neurona i sinapsa, a promjene ne moraju nužno djelovati na fizičku strukturu mreže: mutacija može promijeniti vrijednosti težina i hiperparametara aktivacijskih funkcija, štoviše, te su operacije nužne iako početna evaluacija simpatizira dobrotu strukture mreže. Mutacija, kao i kod svakog drugog evolucijskog algoritma, ima vjerojatnost pojavljivanja. Odabir mikrooperacije može se odrediti proizvoljno, iako najviše smisla ima generirati slučajni broj i prema tome izabrati neku od mogućih. Neke varijante ovog algoritma čak odabiru jedan evolucijski operator u svakoj epohi (križanje/mutacija) i onda samo njega primjene, što nije slučaj kod genetskog algoritma. Problemi se dešavaju prilikom kompleksifikacije (povećanja broja gena u jedinki) - tada se rad algoritma značajno usporava. To je eventualno moguće riješiti definiranjem funkcije dobrote na globalnoj razini koja bi se koristila u evoluciji vjerojatnosti algoritma, a između ostalog bi ovisila i o trajanja jedne epohe.

Operacija *dodajNeuron()*

Dodavanje novog neurona u mrežu započinje odabirom sinapse. Dotična se uklanja, a njezino mjesto se postavlja neuron sa pretpostavljanim početnim vrijednostima. Recimo da je sinapsa spajala neurone A i B. Dobivene su dvije nove: jedna sada spaja neuron A sa novonastalim, a druga novonastali sa neuronom B. Vrijednost prve bit će jednaka 1, dok bude druga poprimila vrijednost stare.

Operacija *dodajSinapsu()*

Ova vrsta mutacije izabire dva neurona koja nisu povezana sinapsom i spaja ih. Inicijalna vrijednost može biti slučajni broj iz neke razdiobe (važno je napomenuti kako vrsta razdiobe može znatno utjecati na razvoj populacije), prosječna vrijednost svih prisutnih pomaknuta za malu slučajnu vrijednost itd. Nova težina imat će inovacijski broj jednak globalnom uvećanom za 1.

Operacija *isključiNeuron()*

Isključenje neurona je važno kad mreže porastu u smislu broja genoma. Tada je potrebno istu simplificirati ovom operacijom - odabire se neuron i njemu se spušta zastavica aktivnosti. Također, sinapse koje su povezane s ovim neuronom se deaktiviraju.

Dakle, izabrani neuron se ne izbacuje, već ostaje očuvan kako bi se kasnije mogao eventualno opet uključiti operacijom suprotnom od ove. Izlazni se neuron ne može isključiti, što je logično. Razumno je također onemogućiti operaciju da isključi ulaze, međutim, ako ona smije to učiniti, nastaje vrlo zanimljiva situacija: tada algoritam može odlučiti koji su ulazi uopće potrebni i direktno smanjiti prostor pretraživanja.

Operacija *isključiSinapsu()*

Simplifikacija se može učiniti deaktiviranjem sinapsi. Kao što je bio slučaj sa isključivanjem neurona, sinapse se ne izbacuju iz kromosoma, već se deaktivacija vrši zastavicom kako bi kasnije težina mogla opet postati aktivna.

Operacija *mutirajSinapsu()*

Ono što se genetskim algoritmom radi pomoću binarnog vektora, ovdje je realizirano kao dodavanje slučajne vrijednosti postojećoj - dobra ideja je koristiti brojeve iz simetričnog intervala kako vrijednost težine mutacijom ne bi isključivo rasla/padala.

Operacija *mutirajBiasNeuron()*

Ova je operacija jednaka gornjoj, budući da se također radi o težinama. Svaki neuron ima bias težinu čija vrijednost se može mijenjati mutacijom. Zbog jednostavnosti implementacije, svaki neuron ima podatak o toj težini, a ulazni neuroni je uopće ne sadrže, tj. vrijednost je fiksirana na 0.

Operacija *promijeniAktivaciju()*

Kako bi se postigla čim veća ekspresivnost, neuronski genomi imaju podatak o aktivacijskoj funkciji. Dakle, mutacijom se ona može promijeniti. Ulazni neuroni uvijek imaju identitet kao aktivacijsku funkciju.

Operacija *mutirajHiperparametre()*

Većina aktivacijskih funkcija ima neki parametar (npr. konstanta koja množi argument u sigmoidalnoj funkciji ili pomak kod linearne) i u tu svrhu je unijeta mogućnost da oni također evoluiraju, kao i same težine. Problem se događa kad prethodno navedenom mutacijom promijenimo aktivacijsku funkciju - u slučaju da opet generiramo

pretpostavljene vrijednosti za njezine parametre, izgubili bismo one koji su već eventualno dobro napredovali kroz epohe. Stoga neuroni trebaju pamtiti sve parametre za sve aktivacijske funkcije.

4.8.5. Izvedba selekcije/eliminacije

Selekcija je operator kojeg treba pažljivo implementirati jer se javlja problem gdje jedinke manje po broju genoma generiraju manju grešku nad skupom za validaciju - kad bi smo u obzir uzimali samo lokalnu dobrotu koja je zapravo rezultat računanja izraza, veće jedinke ne bi dobile šansu za napredak i tako bi algoritam zapeo u lokalnom minimumu. Zbog toga je potrebno formirati heuristiku koja će u obzir uzimati lokalnu dobrotu te kompatibilnost s ostatkom populacije. Također, nije dobro u svakoj epohi raditi eliminaciju: bolja je ideja dopustiti jedinkama da neko vrijeme napreduju pa ako se nakon zadanog broja iteracija ne vidi konkretan pomak u efikasnosti mreže, eliminirati taj kromosom populacije.[11]

Algoritam 4.1 Modifikacija Stanleyevog NEAT-a korištena u implementaciji

```
inicijaliziraj_populaciju()
while uvjet zaustavljanja nije zadovoljen do
    izracunaj_lokalnu_ dobrotu()
    odaberi_predstavnike_grupa()
    izracunaj_globalnu_dobrotu()
    elimininiraj_jedinke()
    krizaj()
    for ∀ kromosom do
        op ← odaberi_mikrooperaciju()
        op.mutiraj(kromosom)
    end for
end while
```

5. Evolucijski razvoj modela strojnog učenja

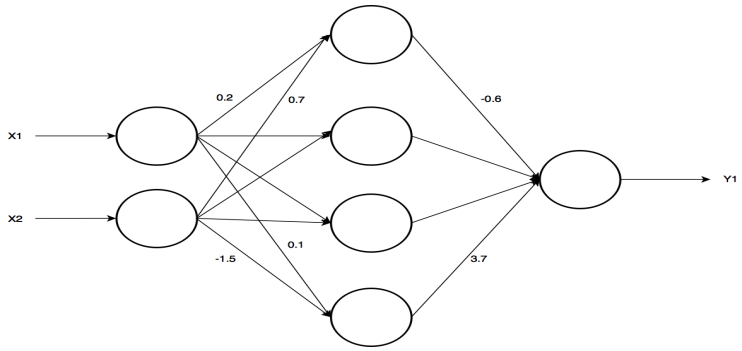
5.1. Evolucija neuronske mreže

Kod učenja neuronske mreže algoritmom *backpropagation* nailazimo na nekoliko problema. Osim već opisanog zaglavljivanja u lokalnim minimumima, backpropagation nam nameće par uvjeta. Prvi je da funkcija koja se želi oponašati mrežom bude neprekinuta. Sljedeći uvjet koji nam nameće ovaj algoritam jest činjenica da aktivacijska funkcija također treba biti derivabilna. Ovi uvjeti proizlaze iz same formule za propagaciju greške unatrag (u izrazu se mogu pronaći derivacije izlaza neurona po težinama i aktivacijske funkcije po težinskoj sumi pojedinih neurona). Genetski algoritam otporan je na ove pojave – ne koristi aktivacijsku funkciju ni njezinu derivaciju, ali oni mogu pronaći globalni minimum čak i da je riječ o funkciji s mnoštvom prekida. Doduše, mogu zapeti u lokalnom minimumu ili pretraživati prostor previše slučajno ako se loše izaberu vjerojatnosti križanja i mutacije.

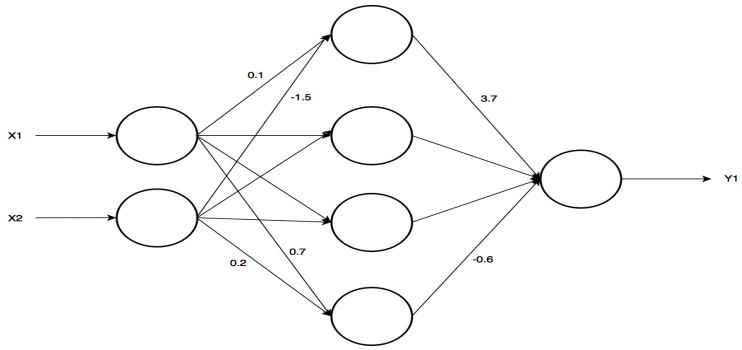
5.1.1. Evolucija težina

Kod učenja mreže gdje jedan kromosom predstavlja polje težina imamo više mogućnosti za evoluciju. Prvi pristup zasniva se na klasičnom genetskom algoritmu koji prima polje težina i zatim poboljšava populaciju kroz epohe algoritma. Kao funkciju za ocjenu dobrote najprikladnije je koristiti neku funkciju koja se inače koristi za ocjenu greške koju je mreža napravila. Najčešće je to srednje kvadratno odstupanje (*mean squared error*). Sam proces odvijao bi se na sljedeći način: prvi korak bila bi inicijalizacija kromosoma u skladu s nekom razdiobom. Zatim bi trebalo ocijeniti dobrotu svake jedinke. To bi ujedno bio računalno najzahtjevniji posao. Pretpostavka je da imamo mnogo uzoraka. Ovisno o broju kromosoma u populaciji trebali bi prilagoditi broj testnih uzoraka na kojima se kromosom ispituje da učenje ne traje dugo.

Konačno, epoha bi bila završena križanjem i mutacijom. Složenost ovog algoritma bila bi prilično velika. Međutim, postoje još neki problemi. Zamislimo da imamo dvije mreže kao što je prikazano na donjim slikama:



Slika 5.1: Neuronska mreža 1



Slika 5.2: Neuronska mreža 2

Na slikama je očito da neuronske mreže sadrže po dva neurona koji obavljaju potpuno identičan posao, uz hipotezu da su aktivacijske funkcije dotičnih jednake i da imaju jednake parametre. Dakle, ako se kromosomi križaju na razini neurona, u slučaju da se točka prijeloma nalazi na trećem neuronu skrivenog sloja, dijete će sadržati dupli neuron, što je vrlo nepogodno jer je i jedna i druga mreža potencijalno bolje rješavala problem nego dijete. Ukoliko je križanje izvedeno na razini težina, onda se javlja permutacijski problem: jednostavnim kombinatornim računom se dolazi do zaključka da jednakih mreža u ovom slučaju ima isto koliko permutacija neurona, dakle 24. Zato operator križanja nije baš popularan u evoluciji težina neuronskih mreža. [7]

5.1.2. Evolucija parametara algoritma backpropagation

Ono što ima smisla evoluirati genetskim algoritmom jesu tzv. hiperparametri neuronske mreže: stopa učenja, koja može ubrzati ili usporiti proces učenja, i inercija, koja određuje koliki udio u korekciji težine pripada prethodnoj korekciji (ako se greška

smanjuje, dobro je povećati stopu učenja kako bi se učenje ubrzalo, analogno vrijedi i za slučaj povećanja greške). Kod ovog pristupa, ali i općenito, zgodno je koristiti ideju unakrsne validacije opisanu u prethodnim poglavljima. Dakle, algoritam će koristiti diskretni binarni kromosom čije varijable želimo da pretražuje u intervalu $(0, 1)$ za inerciju, odnosno u intervalu $(0, \frac{2}{N})$ – dakle, domena pretraživanja za stopu učenja povećana je 4 puta u odnosu na standardnu vrijednost. Algoritam se inicijalizira sa veličinom populacije VP. Taj broj trebao bi se uskladiti sa složenošću računanja izlaza za zadane ulaze testnih, odnosno validacijskih primjeraka u slučaju da ne želimo da algoritam ne traje predugo. Zamisao je sljedeća: pokrene se algoritam učenja težina (npr. backpropagation), nakon završene epohe gdje su se koristili uzorci za učenje uzimamo uzorke za validaciju i, budući da svaki kromosom predstavlja uređeni par (stopa učenja, inercija), trebamo validaciju izvršiti koristeći svaki od tih parova. Dakle, funkcija dobrote će zapravo biti vrijednost greške pojedine mreže na skupu za validaciju gdje je mreža koristila upravo hiperparametre pohranjene u tom kromosomu.

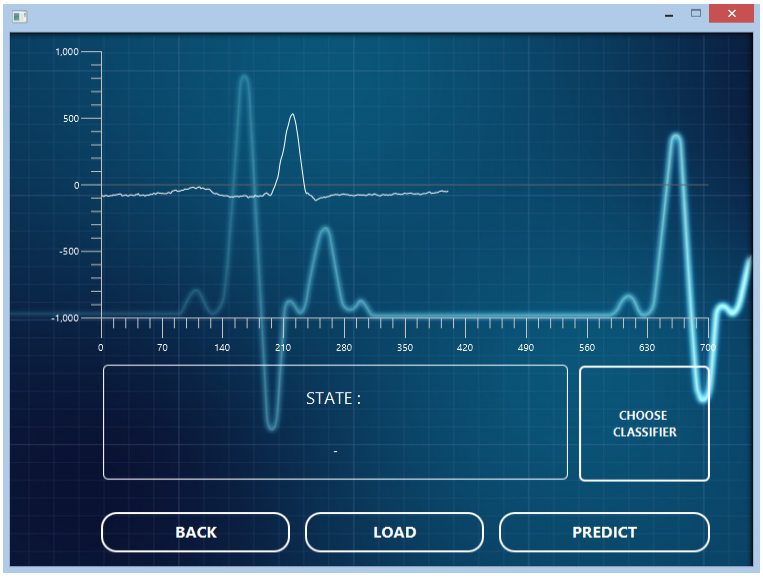
5.2. Evolucija parametara jezgre SVM-a

Ono što je vrlo zgodno mijenjati kod strojeva s potpornim vektorima jesu hiperparametri jezrenih funkcija. Svaka funkcija ima između jednog i tri parametra. Najčešće su to konstante koje množe skalarni produkt prije samog izračuna ili se nadodaju na njih. Evolucija se može napraviti npr. pomoću genetskog algoritma. Valja napomenuti kako ovdje drastično raste vrijeme izvođenja algoritma i ne isplati se koristiti konfiguraciju s mnogo kromosoma u populaciji ili mnogo iteracija. Funkcija dobrote je definirana kao omjer broja uzoraka korištenih u validacijskom skupu i broja ispravno klasificiranih uzoraka iz istog skupa. To je zato što ta funkcija treba za bolji rezultat dati manju vrijednost. Osim jednostavnih kernela, mogu se koristiti i ulančani, koji su u suštini kombinacija dvije ili više jezgrenih funkcija, pa tu broj parametara koji se mogu evoluirati raste. [12]

6. Implementacija programske potpore

Implementacija svih algoritama izvedena je u programskom jeziku Java. Za izgradnju grafičkog sučelja korištena je biblioteka JavaFX, odnosno program Scene Builder. Java se pokazala kao dobar odabir zbog jednostavnosti i brzine pisanja programa, kao i nepotrebnosti izravnog pristupanja memoriji. Međutim, upravo ta činjenica loše utječe na brzinu izvođenja pojedinih algoritama, posebice evolucijskih - populacija se sastoji od mnogo jedinki kroz koje sakupljač smeća treba iterirati i provjeravati treba li se određeni dio memorije osloboditi. Ovo se najviše odražava na algoritam neuro-evolucije rastućih topologija, gdje je svaka jedinka objekt, a ista ima mnogo neurona i sinapsa, koji su također modelirani klasama.

Slika 6.1: Izgled grafičkog korisničkog sučelja prilikom pokretanja



Slika 6.2: Izgled grafičkog korisničkog sučelja prilikom klasifikacije

6.1. Priprema uzoraka

Uzorci su preuzeti sa web stranice physionet.org. PhysioNet je skup baza podataka koje, osim EKG signala, sadrže i druge oblike zapisa podataka o fiziološkom stanju tijela, npr. elektromiografske signale. Sustav su 1999.g. utemeljili NIGMS (National Institute of General Medical Sciences) i NIBIB (National Institute of Biomedical Imaging and Bioengineering). Pohranjeni su zapisi zdravih osoba, kao i pacijenata sa raznim bolestima. Posebnu pažnju treba dati zapisima EKG signala osoba koje su iznenada umrle radi otkazivanja srca. Naime, ti signali su snimljeni holterom i spremljeni u bazu, a sadrže signal od početka trume pa do same smrti. Zapisi su rascjepkani na testne uzorke sa 401 vrijednošću, dakle to je broj ulaza korišten u modelima.

6.2. Implementacija

Implementacija je izvedena kao detektiranje eventualnih trauma uzimajući kao ulaz jedan otkucaj. Na raspolaganju su bili izmjereni EKG signali u trajanju od desetak minuta do nekoliko sati. Ono što loše utječe na efikasnost jest spoznaja da svi otkucaji ne predstavljaju jednu bolest, primjerice, EKG se mogao početi snimati kad još ništa ne pokazuje na eventualne pojave loših stanja i onda u nekom trenutku nastupi srčani udar. Podrazumijevano je da svi otkucaji pripadaju specifičnoj klasi. Za izdvajanje pojedinih otkucaja korišten je Pan-Tompkinsov algoritam preuzet sa GitHuba. Korišten je skup od otprilike 6000 uzoraka, od kojih je ukupno 400 korišteno za validaciju modela. U

skupu postoji približno jednak broj uzoraka za obje klase (normalno stanje i srčani udar). [8]

7. Rezultati

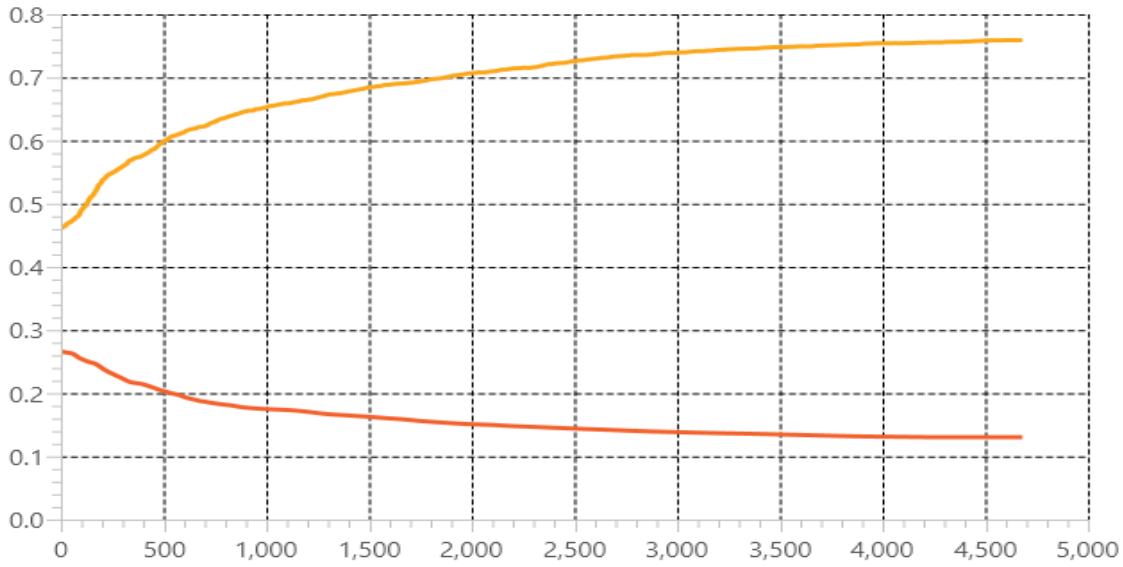
U ovom poglavlju su komentirani ishodi učenja neuronske mreže i stroja s potpornim vektorima raznim metodama. Objašnjeni su problemi kod loše postavljenih parametara kao i vrijeme izvođenja pojedinih algoritama.

7.1. Neuronska mreža učena algoritmom *backpropagation*

Neuronska mreža korištena u programu je unaprijedna slojevita sa 401 ulaznim neuronom, 200 skrivenih i 1 izlaznim. Težine su učene algoritmom propagacije pogreške unatrag(

$$E = \frac{1}{2N} \sum_{i=1}^{N} \sum_{j=1}^{O} (t_j - y_j)^2,$$

), stopa učenja je bila 9.6×10^{-5}, a inercija 0.2. U prvom pokušaju uzeta je u obzir mogućnost da korištenjem istih uzoraka u procesu učenja i validacije dođe do prenaučenosti. Također, bilježena je efikasnost mreže izračunata kao omjer broja točno klasificiranih uzoraka i ukupnog broja uzoraka. Slika 7.1 prikazuje vrijednost srednje kvadratne pogreške na validacijskom skupu i efikasnost na cijelom skupu uzoraka.

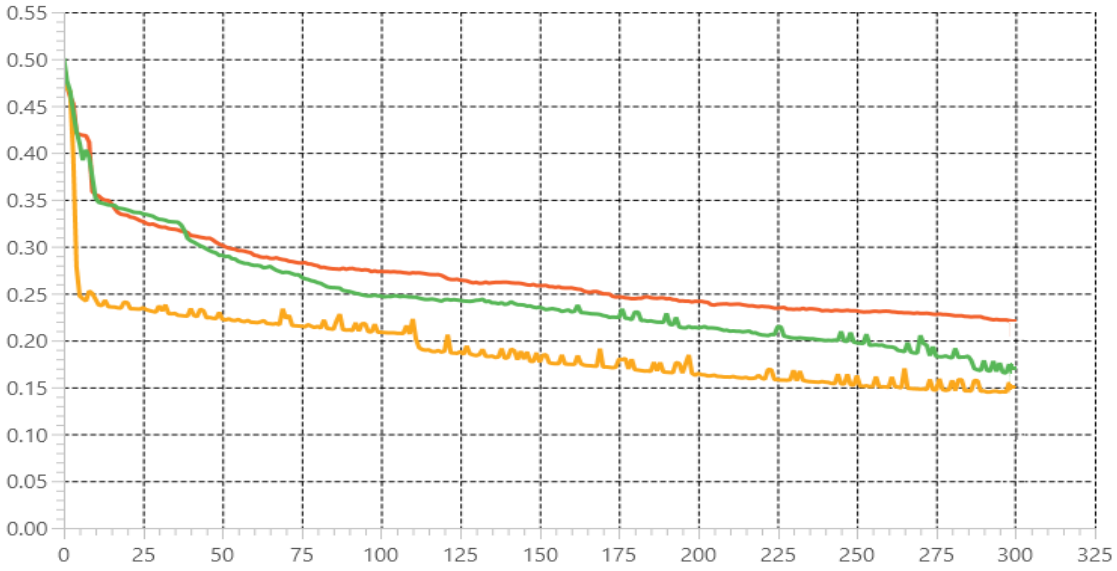


Slika 7.1: *Backpropagation*, efikasnost na cijelom skupu uzoraka i srednja kvadratna pogreška na skupu za validaciju

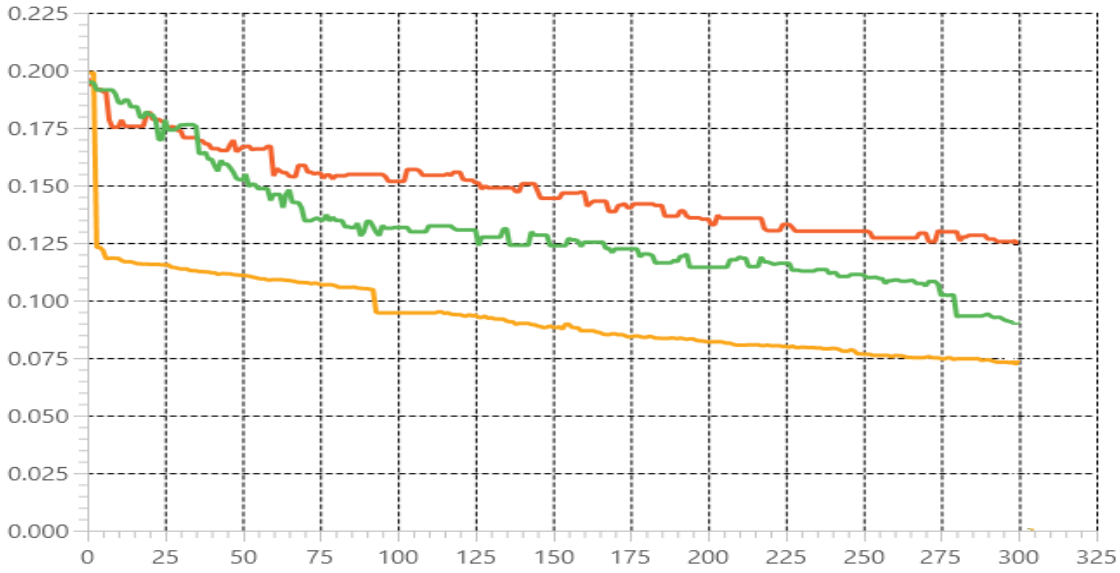
7.2. Neuronska mreža učena genetskim algoritmom

Genetski algoritam je iskorišten u svrhu učenja neuronske mreže jednake strukture kao što je bio slučaj i kod gradijentnog spusta, međutim parametri koji su evoluirani su, osim težina, bili i parametri aktivacijskih funkcija. Korištena je sigmoidalna aktivacijska funkcija, a parametar koji se optimirao za funkciju svakog neurona posebno bila je konstanta koja množi težinsku sumu. Dakle, u pitanju je funkcija $f(x) = \frac{1}{1+e^{-ax}}$. Ukupni broj varijabli u takvoj strukturi je 80802, stoga je korišteno direktno preslikavanje. Algoritam je pokretan sa više vrsta selekcija i križanja, dok je mutacija u svakoj varijanti bila dodavanje slučajnog broja iz Gaussove razdiobe trenutnoj vrijednosti varijable.

Slike u nastavku prikazuju vrijednosti srednje kvadratne pogreške sa vrijednostima vjerojatnosti za operaciju križanja: 60% (crvena linija), 40% (zelena linija) i 25% (žuta linija). Križanje je implementirano kao linearna kombinacija gena, kako je opisano u poglavlju o genetskim algoritmima. Vrijednost parametra λ bila je 0.15. Vjerojatnost mutacije bila je 0.8% za svako pokretanje algoritma. Slika 7.3 prikazuje prosječnu vrijednost srednje kvadratne pogreške populacije izračunate na skupu za učenje, a 7.4 također vrijednost greške, ali samo najbolje jedinke, i to na validacijskom skupu.



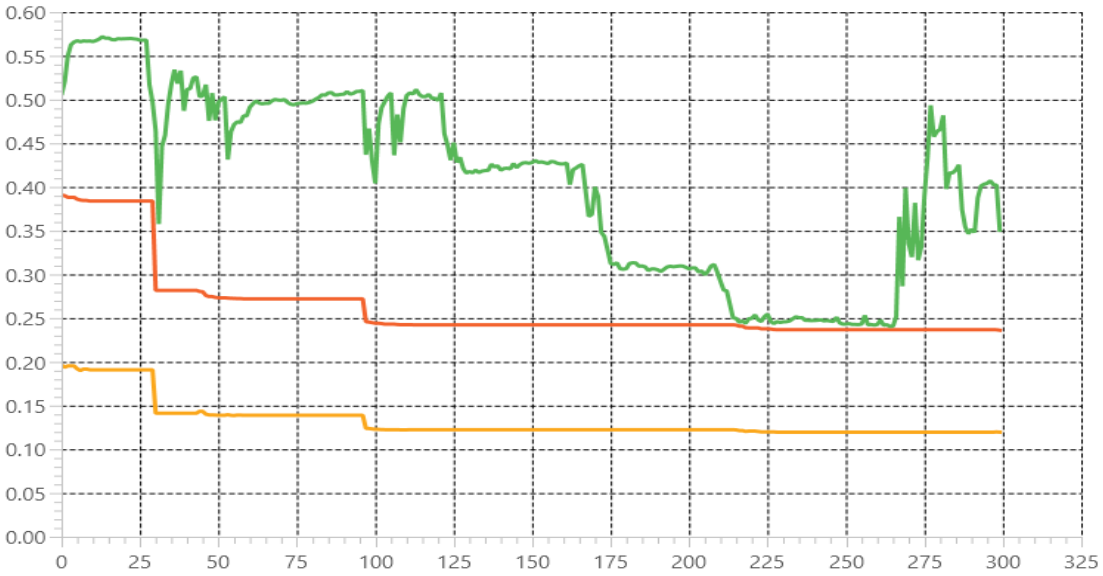
Slika 7.2: GA, greška na skupu za učenje



Slika 7.3: GA, greška na skupu za validaciju

Iz grafova je očito da križanje može štetno djelovati na populaciju. Također, u razmatranje je, osim pohlepne, uzeta varijanta 8-turnirske selekcije za odabir preživjelih jedinki sa vjerojatnošću križanja 50% i mutacije 0.5%. Zelena krivulja predstavlja prosječnu vrijednost dobrote cijele populacije, a crvena najbolju na skupu za učenje, dok je dobrota te iste najbolje jedinke na validacijskom skupu označena žutom linijom. Zanimljivo je uočiti da greška na validacijskom ima znatno veće anomalije od one na skupu za učenje. To je zbog toga što uzorci iz validacijskog skupa koji pripadaju prvoj

klasi imaju vrlo veliku sličnost sa onima iz skupa za učenje koji pripadaju drugoj klasi, a genetski algoritam koristi samo skup za učenje.



Slika 7.4: GA, 8-turnirska selekcija

7.3. Učenje težina neuronske mreže genetskim algoritmom u *hill-climbing* varijanti

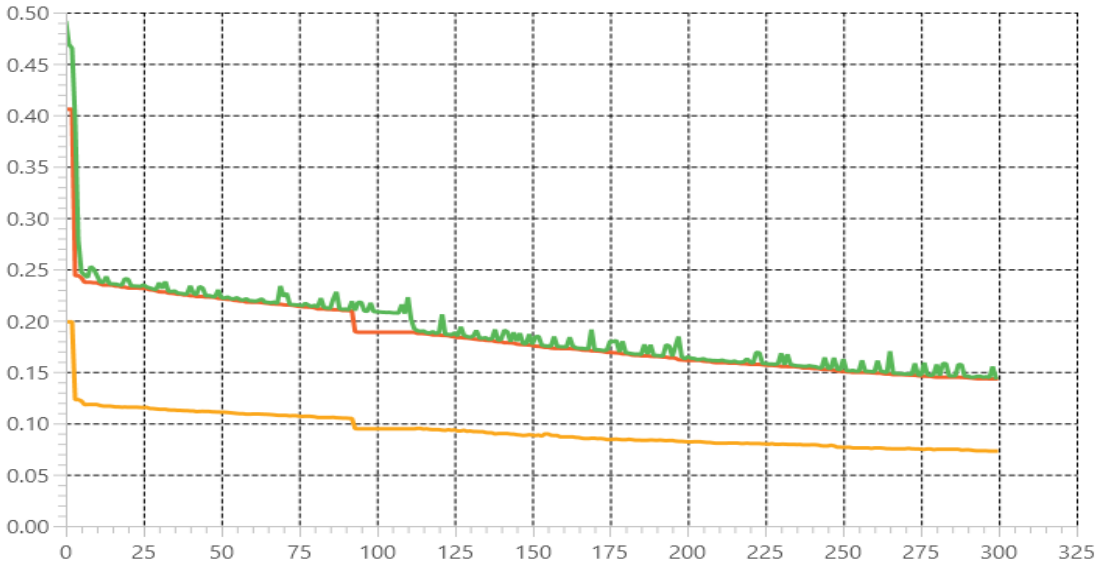
Hill-climbing genetski algoritam (evolucijska strategija) temelji se na ideji elitizma i jedinki koje su vrlo slične najboljoj. Nakon inicijalizacije, bira se najbolji kromosom i on jedini preživljava. Zatim se populacija popunjava kromosomima identičnih gena kao kod najbolje jedinke. Mutacija zatim mijenja gene i opet se vrši selekcija, a algoritam radi dok se ne postigne uvjet zaustavljanja. Ovdje je vrlo bitno istaknuti da će prevelika vjerojatnost za mutaciju pojedinog gena gotovo sigurno rezultirati neuspjehom, budući da je to ekvivalentno nasumičnom pretraživanju prostora problema. Adekvatna vjerojatnost može se odrediti preko količine gena koja se želi mutirati. Naime, izraz za očekivanje neke slučajne varijable diskretne razdiobe je:

$$E(x) = \sum_{i=1}^{N} x_i p(x_i)$$

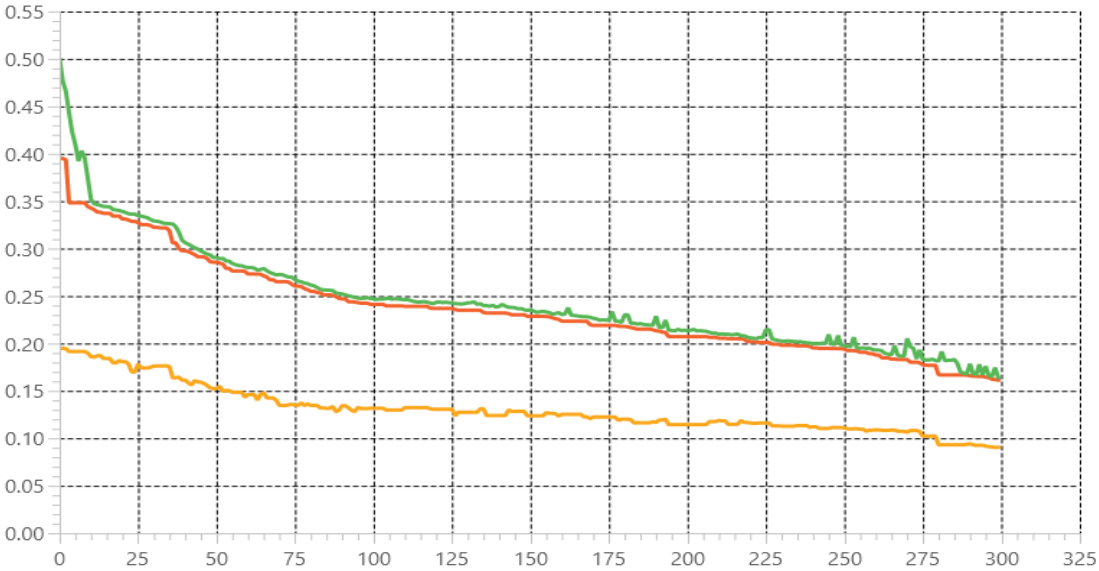
U implementaciji je korištena uniformna distribucija, pa oba događaja (0 - mutacija se nije dogodila, 1 - mutacija se dogodila) za svaku težinu imaju jednaku vjerojatnost, stoga možemo pisati:

$$E(x) = N \cdot 1 \cdot p(x_i) = Np(x_i),$$

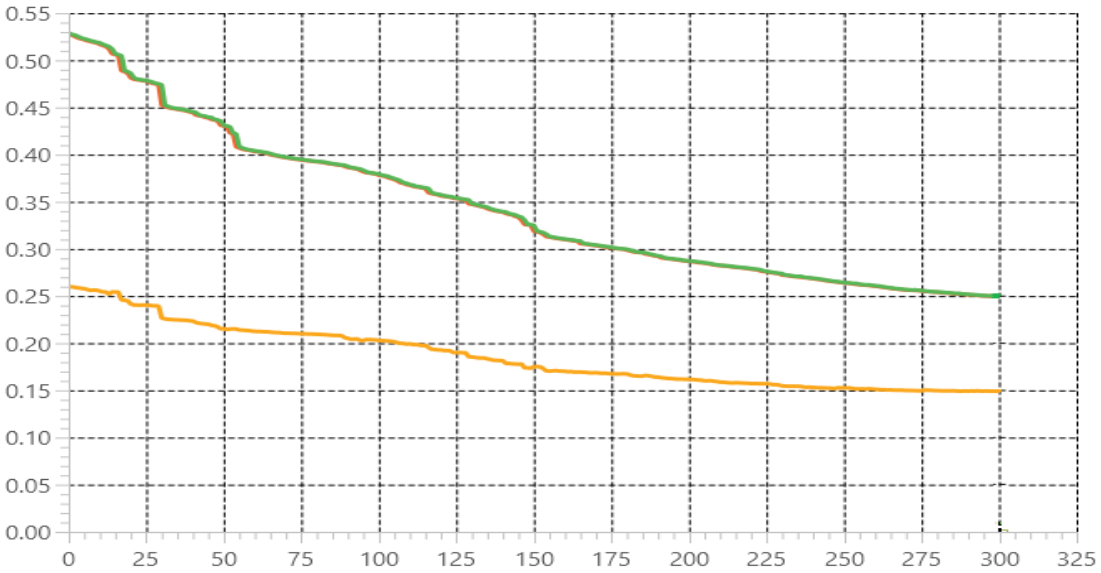
gdje je $E(x)$ očekivani broj mutacija, N broj težina, a $p(x_i)$ vjerojatnost jedne mutacije. Grafovi prikazuju rezultate algoritma s različitim vjerojatnostima mutacije: 0.1%, 0.5% i 0.04% odnosno kretanje prosječne i najmanje greške na skupu za učenje i najmanje greške na validacijskom skupu.



Slika 7.5: Evolucijska strategija, prvo pokretanje



Slika 7.6: Evolucijska strategija, drugo pokretanje

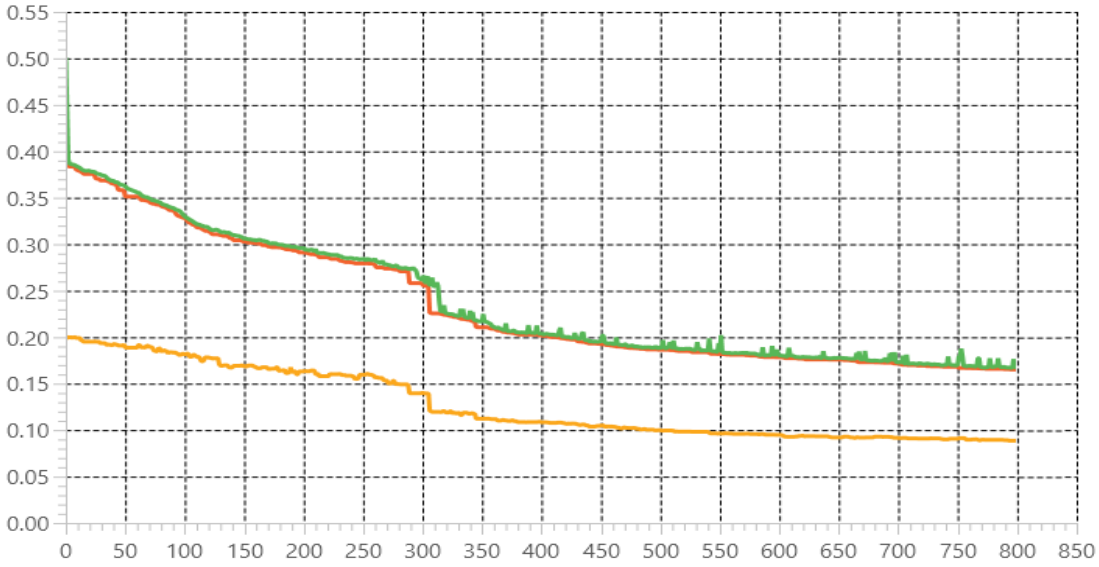


Slika 7.7: Evolucijska strategija, treće pokretanje

Vidi se kako u zadnjem grafu greška počinje stagnirati oko 0.15. To se događa zbog vrlo male vjerojatnosti mutacije kombinirane sa pohlepnom selekcijom.

7.4. Utjecaj smanjenja frekvencije uzorkovanja na efikasnost modela

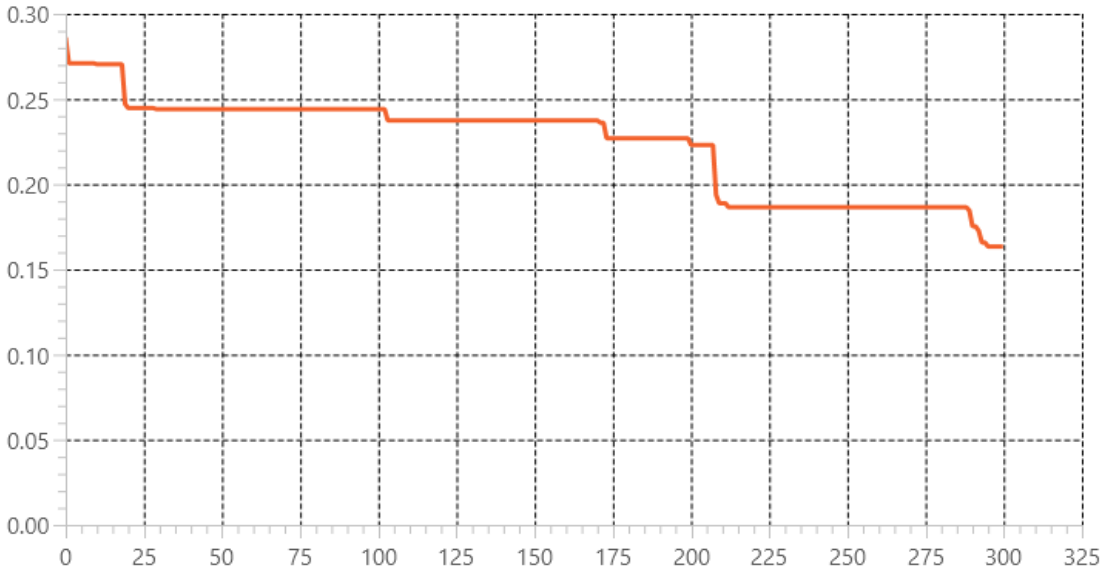
Budući da mreža sa čak 401 ulaza ima vrlo velik broj parametara koji se optimiraju, dobra je ideja da se frekvencija uzorkovanja EKG signala smanji uzimajući u obzir samo vrijednost očitanog signala u svakom p-tom dijelu cijelog uzorka. Zato je broj točaka sa 401 smanjen na 200 kako bi broj parametara mreže bio prihvatljiviji, a samim time i vrijeme učenja manje. Međutim, dobiveni rezultati nisu baš blizu prvotnoj arhitekturi. Sljedeći graf prikazuje dobivene srednje kvadratne greške, čak i na puno većem broju evaluacija. Crvena linija prikazuje najbolju dobrotu u svakoj epohi na skupu za učenje, a zelena prosječnu. Žuta linija predstavlja performansu najbolje jedinke na validacijskom skupu. Valja uočiti kako je u ovom pokretanju korišteno 30000 evaluacija više od ostalih, gdje se koristilo približno 18000 evaluacija.



Slika 7.8: Evolucijska strategija, pokretanje uz dvostruko manji broj ulaza

7.5. Neuro-evolucija rastućih topologija

Algoritam NEAT je bio izrazito zahtjevan kad je trebalo namjestiti parametre, a isto tako i najsporiji. Pokazalo se kako su ipak pravilne arhitekture neuronskih mreža pogodne za ovaj problem (unaprijedne slojevite). Rezultati dobiveni algoritmom NEAT podsjećaju na vrlo grubu pretragu prostora, pogotovo zato što za ovaj problem potencijalno postoji mnogo solidnih arhitektura, samo što je do njih teško doći jer je upitno kakve trebaju biti vjerojatnosti mutacije za pojedinu mikrooperaciju. Slika 7.10 prikazuje kretanje najbolje jedinke na skupu za učenje tijekom 300 epoha i znatno manjim brojem uzoraka.



Slika 7.9: NEAT

7.6. Stroj s potpornim vektorima

Kod stroja s potpornim vektorima korištene su razne jezgrene funkcije. Kao što je bio slučaj i kod aktivacijskih funkcija neurona, i ovdje su funkcije proširene raznim parametrima koji se mogu evoluirati. Taj je proces također proveden genetskim algoritmom, doduše uz vrlo malu populaciju, zbog računalne složenosti algoritma. U tablici 7.1 prikazani su rezultati nekih jezgara sa efikasnošću na validacijskom skupu.

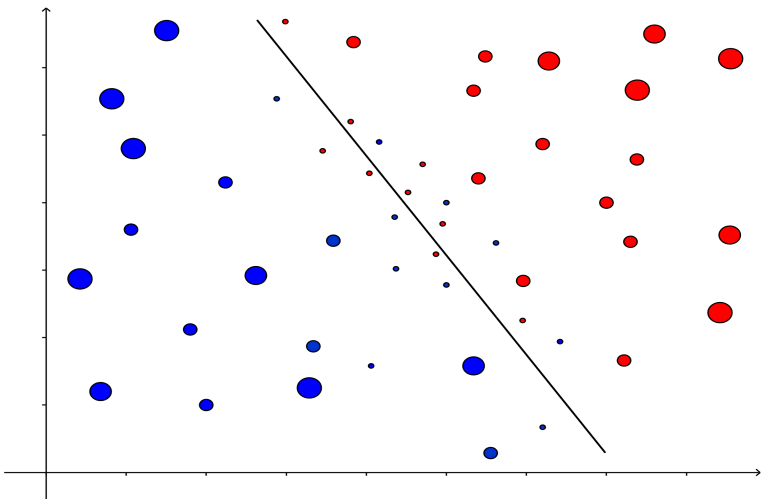
Rezultati evoluiranja hiperparametara jezgre

Jezgrena funkcija	Evoluirani parametri	Efikasnost
$\tanh\left(A\vec{\mathbf{x}}\cdot\vec{\mathbf{y}}+B\right)$	$A=0.016$, $B=1.145$	87.50%
$A\vec{\mathbf{x}}\cdot\vec{\mathbf{y}}+B$	$A=-1.776$, $B=2.953$	72.46%
$-\log\left(1+\|\vec{\mathbf{x}}-\vec{\mathbf{y}}\|^{A}\right)$	$A=2.744$	90.54%
$\sqrt{\|\vec{\mathbf{x}}-\vec{\mathbf{y}}\|^2+A^2}$	$A=3.196$	73.63%
$e^{-\frac{\|\vec{\mathbf{x}}-\vec{\mathbf{y}}\|^2}{2A^2}}$	$A=1.661$	91.04%
$1-3\frac{\|\vec{\mathbf{x}}-\vec{\mathbf{y}}\|+\|\vec{\mathbf{x}}-\vec{\mathbf{y}}\|^3}{2A}$	$A=2.628$	78.43%
$e^{-\frac{\|\vec{\mathbf{x}}-\vec{\mathbf{y}}\|}{2A^2}}$	$A=1.138$	77.68%

Tablica 7.1: Tablica efikasnosti različitih jezgrenih funkcija na skupu za validaciju

8. Moguća proširenja

Od mogućih proširenja valja spomenuti implementaciju i nekih drugih modela, posebice onih koji u obzir uzimaju redoslijed ulaznih podataka (npr. konvolucijske i povratne neuronske mreže), kao i evolucijskih algoritama za učenje njihovih parametara. Također, ukoliko više modela pokaže približno jednaku efikasnost, svakako se isplati izgraditi ansambl raznih modela - to je koncept koji kombinira više modela strojnog učenja u jaki klasifikator - oni naprosto imaju veću sigurnost kod korištenja dajući na svojim izlazima velik postotak za smještanje u neku klasu. Također, testni uzorci mogu itekako stvarati probleme (npr. može doći do grubih, slučajnih ili sustavnih pogrešaka u postupcima mjerenja) i u tu svrhu bi se uzorci trebali vrednovati po važnosti što bi osiguralo da model postane manje osjetljiv na iste. To je prikazano dijagramom ispod. U svrhu ubrzanja algoritma može se iskoristiti paralelizacija i izvođenje nekih operacija na grafičkom procesoru (GPGPU tehnologija).



Slika 8.1: Testni uzorci uz dodjelu važnosti

9. Zaključak

Analiza EKG signala vrlo je složen zadatak kojemu treba pristupiti pažljivim odabirom modela, kao i algoritma koji će optimirati težine istog. Posebnu pozornost treba obratiti na oblikovanje ulaza modela kao i korištenje uzoraka za učenje i testiranje. Učenje može biti vrlo dugotrajan proces zbog brojnih parametara pa treba napraviti kompromis između numeričkih i evolucijskih postupaka. Stroj sa potpornim vektorima se pokazao kao superioran model nad neuronskom mrežom u problemu klasifikacije signala temeljem njegovih izmjerenih vrijednosti.

Kod evolucijskih algoritama treba paziti u slučaju kad se oni koriste u svrhu evoluiranja mreže: operacija križanja pokazala se vrlo destruktivnom, dok varijanta koja koristi samo mutaciju ima smisla. Također, genetski algoritam je otporan na funkcije koje sadrže prekide, dok algoritmi učenja bazirani na derivaciji zahtijevaju da funkcija čiji se globalni minimum traži bude derivabilna, a takva treba biti i aktivacijska funkcija, dok njezina derivacija treba biti definirana u svakoj točki.

Vrlo je važno podesiti vjerojatnost mutacije tako da ona ne uzrokuje nasumično pretraživanje, što se može detektirati promatranjem najbolje jedinke, ni zaglavljenje u lokalnom minimumu.

Na kraju, bitno je odabrati ispravan način ocjene efikasnosti modela - postupak treba sadržavati unakrsnu validaciju jer je glavni cilj zapravo generalizirani stroj s potpornim vektorima ili neuronska mreža, a ne greška na skupu na učenje, budući da će eventualni komercijalni proizvod biti korišten na nepoznatim uzorcima.

LITERATURA

[1] MOHD FAISAL AMIN. What happens during a Heart Attack. https://www.youtube.com/watch?v=sR4cVqnWV9Q/, 2013. [].

[2] Cristopher M. Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[3] Marin Golub. *Genetski algoritam*. Fakultet elektrotehnike i računarstva, 2004.

[4] Simon Haykin. *Neural networks, a comprehensive foundation*. Prentice Hall International, Inc., 1999.

[5] Bojana Dalbelo Bašić Jan Šnajder. *Strojno učenje*. 2014.

[6] Alan Jović i Nikola Bogunović. Electrocardiogram analysis using a combination of statistical, geometric, and nonlinear heart rate variability features. *Artificial intelligence in medicine*, 51(3):175–186, 2011.

[7] Marin Golub Marko Čupić, Bojana Dalbelo Bašić. *Neizrazito, evoucijsko i neuroračunarstvo*. 2013.

[8] Blake Milner. Real time QRS detection. https://github.com/blakeMilner/real_time_QRS_detection/, 2013. [].

[9] MIT OpenCourseWare. 16. Learning: Support Vector Machines. https://www.youtube.com/watch?v=_PwhiWxHK8o/, 2014. [].

[10] John R.Koza. *A field guide to genetic programming*. 2008.

[11] Kenneth O Stanley i Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

[12] Sven F. Crone Stefan Lessmann, Robert Stahlbock. Genetic algorithms for support vector machine model selection. stranice 3063–3069, 2006.

[13] Marko Čupić. *Prirodom inspirirani optimizacijski algoritmi. Metaheuristike.* 2013.

[14] Wikipedia. Elektrokardiogram. https://hr.wikipedia.org/wiki/Elektrokardiogram/, 2015. [].

[15] Wikipedia. Support vector machine. https://en.wikipedia.org/wiki/Support_vector_machine/, 2017. [].

Klasifikacija srčanih bolesti na temelju EKG signala uz pomoć strojnog učenja podržanog evolucijskim računarstvom

Sažetak

Cilj ovog završnog rada je upoznati se s najčešćim načinima za analizu signala u medicinskoj dijagnostici u svrhu prepoznavanja mogućih tegoba srca. Korišteni su modeli iz strojnog učenja, a isti su učeni numeričkim i stohastičkim metodama. Za klasifikaciju su odabrani stroj s potpornim vektorima i neuronska mreža, a učenje njihovih parametara je izvedeno algoritmima evolucijskog računanja: genetski algoritam, evolucijska strategija, kao i neuro-evolucija rastućih topologija. Opisani su problemi u korištenju pojedinih algoritama sa loše odabranim parametrima.

Ključne riječi: EKG, klasifikacija, strojno učenje, SVM, jezgrene metode, neuronske mreže, NEAT, evolucijska strategija, genetski algoritam, evolucijsko računarstvo

Classification of heart diseases based on ECG signal using machine learning supported by evolutionary computation

Abstract

Goal of this project is to get familiar with methods for biomedical signal analysis for purpouse of detecting heart wave abnormalities. The machine learning techniques were used and models were trained using numerical, as well as stohastic algorithms. The classification is implemented using support vector machine and feed forward neural network. Their parameters were trained using evolutionary computation algorithms: genetic algorithm, evolution strategy, as well as neuro-evolution of augmented topologies(NEAT). Problems with which one can encounter because of badly selected parameters are described.

Keywords: ECG, classification, machine learning, SVM, kernel methods, neural networks, NEAT, genetic algorithm, evolution strategy, evolutionary computation