Surogatni model za funkciju Rosenbrock u višedimenzijskom prostoru

Gilja, Borna

Master's thesis / Diplomski rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Rijeka, Faculty of Engineering / Sveučilište u Rijeci, Tehnički fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:190:632027

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Diplomski studij računarstva

Diplomski rad

Surogatni model za funkciju Rosenbrock u višedimenzijskom prostoru

Rijeka, srpanj 2022.

Borna Gilja
0069078391

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Diplomski studij računarstva

Diplomski rad

Surogatni model za funkciju Rosenbrock u višedimenzijskom prostoru

Mentor: doc. dr. sc. Goran Mauša

Rijeka, srpanj 2022.

Borna Gilja
0069078391

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA DIPLOMSKE ISPITE

Rijeka, 14. ožujka 2022.

Zavod: **Zavod za računarstvo**
Predmet: **Inženjerstvo kompleksnih programskih sustava**
Grana: **2.09.04 umjetna inteligencija**

ZADATAK ZA DIPLOMSKI RAD

Pristupnik: **Borna Gilja (0069078391)**
Studij: Diplomski sveučilišni studij računarstva
Modul: Programsko inženjerstvo

Zadatak: **Surogatni model za funkciju Rosenbrock u višedimenzijskom prostoru / Surrogate model for Rosenbrock function in multidimensional space**

Opis zadatka:

Istražiti primjenu surogatnih modela u optimizaciji koja iziskuje računski zahtjevne funkcije dobrote. Odabrati odgovarajuću vrstu surogatnog modela za aproksimaciju funkcije Rosenbrock u višedimenzijskom prostoru i obrazložiti njezinu prikladnost za rješavanje zadanog problema. Implementirati genetski algoritam koji će koristiti odabrani surogatni model s ciljem traženja globalnog minimuma odabrane funkcije. Analizirati učinkovitost odabranog rješenja u vidu računalnih resursa i energije.

Rad mora biti napisan prema Uputama za pisanje diplomskih / završnih radova koje su objavljene na mrežnim stranicama studija.

Borna Gilja

Zadatak uručen pristupniku: 14. ožujka 2022.

Mentor:

Doc. Goran Mauša, dipl. ing.

Predsjednik povjerenstva za diplomski ispit:

Prof. dr. sc. Kristijan Lenac

Izjava o samostalnoj izradi rada

Izjavljujem da sam ovaj završni rad izradio samostalno na temelju znanja stečenog na Tehničkom fakultetu u Rijeci te koristeći navedenu literaturu.

Rijeka, srpanj 2022.

_____

Ime Prezime

Zahvala

Zahvaljujem doc. dr. sc. Goranu Mauši na podršci tijekom pisanja ovoga rada, korisnim raspravama i savjetima te dodatnoj literaturi i materijalima.

Također, zahvaljujem svojoj obitelji i prijateljima na konstantnoj podršci, kako kroz studij, tako i kroz cjelokupno obrazovanje.

Sadržaj

Popis slika

Poglavlje 1

UVOD

Posljednjih nekoliko godina došlo je do značajnog povećanja sposobnosti inženjera u izgradnji modela za simulaciju rada složenih proizvoda. Na primjer, za potrebe automobilske industrije danas možemo simulirati razinu ozljede putnika u prometnoj nesreći. Također, znatno se povećala i sposobnost izmjene takvih modela kako bi se održao korak s neprestanim promjenama u dizajnu. Temeljem toga se potencijal za korištenje optimizacije, odnosno za poboljšanje inženjerskog dizajna, znatno povećao te je sada veći nego ikad prije. Međutim, najveća prepreka u korištenju optimizacije je dugo vremensko izvođenje simulacija [1]. Stoga je izravna primjena gotovo svakog optimizacijskog algoritma na simulaciju izrazito spora [1].

Osnovna ideja iza surogatnih modela je izbjegavanje dodatnih povećanja resursa utrošenih za računalnu snagu (*eng. computing power*). Umjesto toga, cilj je razvijanje brzih matematičkih aproksimacija [1] kojim bi se zamijenile dugotrajne i skupe simulacije. Surogatni model je inženjerska metoda koja se koristi kada se ishod interesa ne može lako izmjeriti ili izračunati, pa se umjesto njega koristi model ishoda. Korištenjem surogatnih modela mogu se postaviti i odgovoriti mnoga pitanja koja bi pomogla u donošenju odluka. Nakon boljeg razumijevanja problema osoba se može vratiti na izvođenje simulacija s ciljem testiranja novih ideja te prema potrebi ažurirati aproksimacije. Takav proces se iterira dok se ne pronađe zadovoljavajuće rješenje.

Ovaj diplomski rad izrađen je u sklopu Erasmus+ projekta pod naslovom "Pro-

moting Sustainability as a Fundamental Driver in Software development Training and Education" s oznakom 2020-1-PT01-KA203-078646. Njime se nastoji istražiti primjenu surogatnih modela u optimizaciji i analizirati učinkovitost odabranog rješenja u vidu utrošenih računalnih resursa, vremena i energije. Takva primjena biti će istražena korištenjem Rosenbrock funkcije, u višedimenzijskom prostoru za funkciju dobrote s ciljem traženja globalnog minimuma. Osim što optimizacija Rosenbrock funkcije spada u probleme neograničene optimizacije, njezin globalni minimum se nalazi na dnu glatke i uske doline paraboličnog oblika. Općim algoritmima optimizacije je teško razlikovati smjer pretraživanja zbog male količine informacija o pretraživanju zbog čega svoje pretraživanje često završavaju u područjima lokalnih minimuma [2]. Rosenbrock funkcija je odabrana kao studija slučaja (*eng. case study*) u kojoj predstavlja računalno "skupu" funkciju. Za rješavanje optimizacijskog problema odabran je genetski algoritam koji koristi surogatni model u obliku neuronske mreže za izradu potencijalnih rješenja koja će naknadno biti vrednovana Rosenbrock funkcijom.

Poglavlje 2

OPIS PROBLEMA I OSNOVNIH POJMOVA

Unutar ovog poglavlja biti će objašnjeni osnovni pojmovi poput evolucijskog računarstva i optimizacije. Osim toga, biti će objašnjeno značenje surogatnih modela, motivacija za njihovo korištenje, a zatim prednosti koje njihovo korištenje donosi.

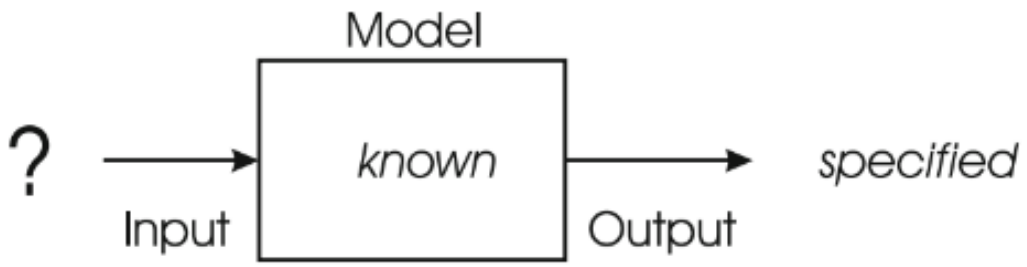
2.1 Motivacija

Glavna motivacija za izradu ovog rada je primjena surogatnog modela prilikom rješavanja problema koji se može klasificirati kao problem crne kutije (*eng. black box problems*). U računarstvu, pojam crne kutije označava uređaj, sustav ili program koji daje uvid u ulazne i izlazne podatke [1], ali ne daje uvid u interne procese. Takav sustav u početku čeka na unos podataka koje zatim obrađuje nekim računalnim modelom, čiji detalji nisu poznati krajnjem korisniku, te na kraju vraća izlazni rezultat. Izlazni rezultat može biti prikaz nekakvih poruka na ekranu, spremanje vrijednosti u izlaznu datoteku i sl. U biti, svaki sustav crne kutije se sastoji od tri komponente, ulaz, model i izlaz. Ovisno o tome koja je komponenta nepoznata, može se napraviti podjela problema crne kutije na probleme:

1. optimizacije,

2. modeliranja sustava,
3. simulacije.

Kod optimizacijskih problema poznati su model i izlazni rezultat (ili rezultati ako ih je više) te je cilj traženje ulaza kojim se dobiva željeni izlazni rezultat [1].



Slika 2.1 Prikaz optimizacijskog problema (preuzeto iz [1])

Jedan od najpoznatijih optimizacijskih problema je problem trgovačkog putnika (*eng. travelling salesman problem*) u kojem je potrebno pronaći najkraći put za obilazak svih gradova pod uvjetom da se niti jedan grad ne obiđe dva ili više puta. Ovaj problem se ubraja u klasu NP teških problema i ima više varijacija. Stoga ne čudi to što ima široku primjenu u područjima poput planiranja, logistike pa čak i u proizvodnji mikročipova.

Osim optimizacijskih problema, u probleme crne kutije ubrajaju se još i problemi modeliranja sustava kao i simulacijski problemi. Kod problema modeliranja, tj. identifikacije sustava poznati su ulazni i izlazni podaci te je cilj traženje modela sustava [1]. Prilikom rada s takvim problemima nastoji se napraviti model koji će što bolje mapirati ulazne podatke u izlazne. Kod simulacijskih problema, poznati su ulazni podaci i model pomoću kojih se žele otkriti izlazni rezultati [1]. Korištenje simulacije je ponekad puno isplatljivije od provođenja stvarnog eksperimenta stoga su takvi problemi često zastupljeni u svakodnevnom životu.

U ovom radu se radilo s dva takva problema. Jedan se može klasificirati kao problem modeliranja u kojem je cilj bio funkciju dobrote opisati surogatnim modelom,

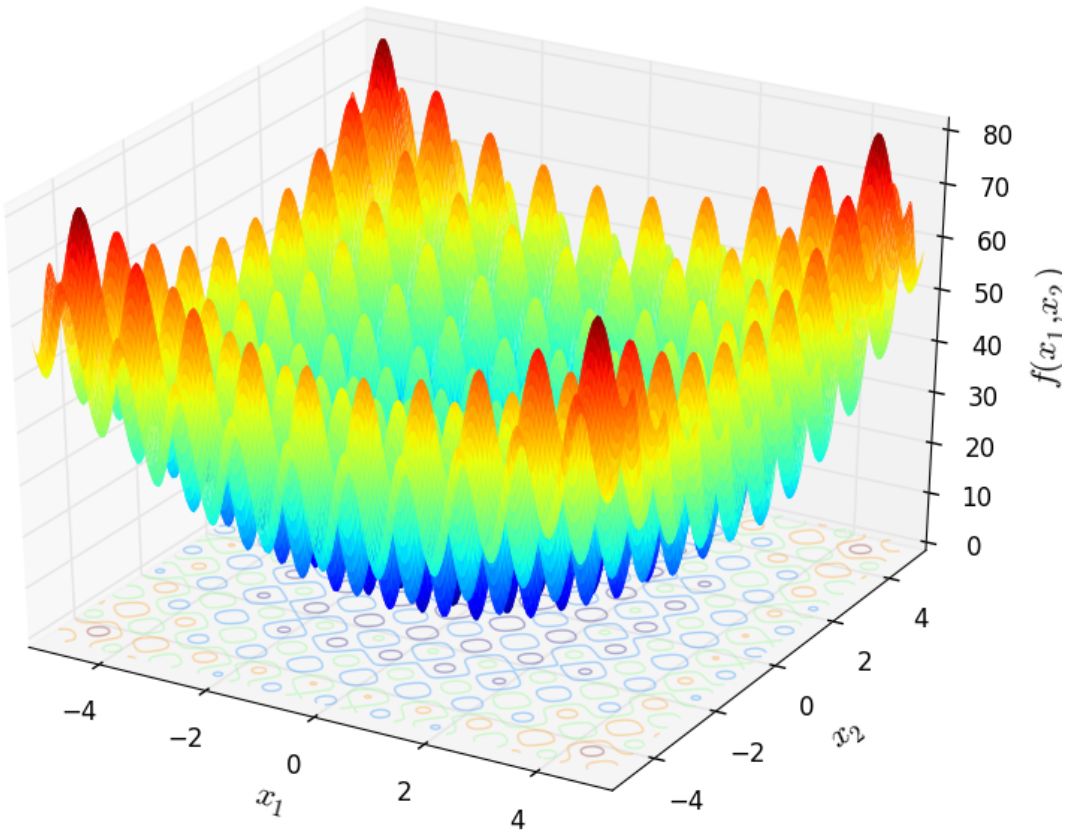
dok se drugi ubraja u optimizacijske probleme i odnosi se na pronalazak optimalnog rješenja.

2.2 Testiranje optimizacijskih algoritama

Testiranje optimizacijskih algoritama, tj. njihovih karakteristika, izvodi se pomoću posebnih testnih funkcija. Takve funkcije se koriste kao dokaz učinkovitosti algoritma te su njihovi rezultati često priloženi prilikom predstavljanja novog algoritma [3]. Ne postoji standardni popis funkcija koje istraživači moraju koristiti prilikom testiranja nego samostalno odabiru skup testnih funkcija [3] koje se mogu razlikovati po svojstvima kao što su broj ciljeva (razlikujemo jednociljne od višeciljnih funkcija) i ograničenost prostora pretraživanja. Neke od najpoznatijih funkcija za testiranje optimizacijskih algoritama su:

- Rastrigin - $f(\mathbf{x}) = 10n + \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i)]$,
- Sphere - $f(\boldsymbol{x}) = \sum_{i=1}^{n} x_i^2$,
- Rosenbrock - $f(\boldsymbol{x}) = \sum_{i=1}^{n-1} \left[100\left(x_{i+1} - x_i^2\right)^2 + \left(1 - x_i\right)^2\right]$,
- Beale - $f(x, y) = (1.5 - x + xy)^2 + (2.25 - x + xy^2)^2 + (2.625 - x + xy^3)^2$.

Na slici 2.2 prikazana je Rastrigin funkcija. Rastrigin funkcija je tipičan primjer nelinearne višemodalne funkcije i karakterizira ju velik broj lokalnih minimuma [4]. Funkcija je definirana u višedimenzijskom prostoru, a njezin globalni minimum iznosi 0 i postiže se kada su sve ulazne vrijednosti jednake nuli. Pronalazak globalnog minimuma je težak zbog velikog prostora pretraživanja.



Slika 2.2 3D prikaz Rastrigin funkcije za dva ulazna parametra. Na slici x_1 i x_2 označavaju vrijednosti parametara dok $f(x_1, x_2)$ označava vrijednost Rastrigin funkcije za odabrane parametre (preuzeto iz [5])

Funkcije za testiranje optimizacijskih algoritama su u osnovi optimizacijski problemi prikazani u obliku numeričkih matematičkih funkcija [3]. Njihovo optimalno rješenje je poznato, ali je skriveno u mnoštvu suboptimalnih rješenja kako bi se otežalo njegovo traženje optimizacijskim algoritmom. Bilo koji algoritam za optimizaciju nastoji pronaći optimalno rješenje uz što manji utrošak vremena i ostalih resursa, ali pronalazak takvog rješenja nije zajamčen. Na primjer, kod Rastrigin funkcije (slika 2.2) se može dogoditi da algoritam pronađe lokalni optimum koji ga odvede u krivom smjeru pa zatim više ne može pronaći optimalno rješenje. Stoga se učinkovitost optimizacijskog algoritma mjeri s obzirom na njegovu globalnu pretraživost i sposobnost lokalne konvergencije [3].

Unutar ovog rada korištena je Rosenbrock funkcija koju je predstavio Howard Harry Rosenbrock 1960. godine. Globalni minimum nalazi se unutar dugačke, uske, ravne doline paraboličnog oblika [5] zbog kojega se ova funkcija često naziva Rosenbrockova dolina.

Iako je traženje doline trivijalno, isto se ne može reći za globalni minimum čiji je pronalazak znatno teži. Funkcija je definirana dvama koeficijentima a i b čije su vrijednosti najčešće postavljene na 1 i 100 [2]. Funkcija je definirana u višedimenzijskom prostoru te njezina formula za dva ulazna parametra glasi:

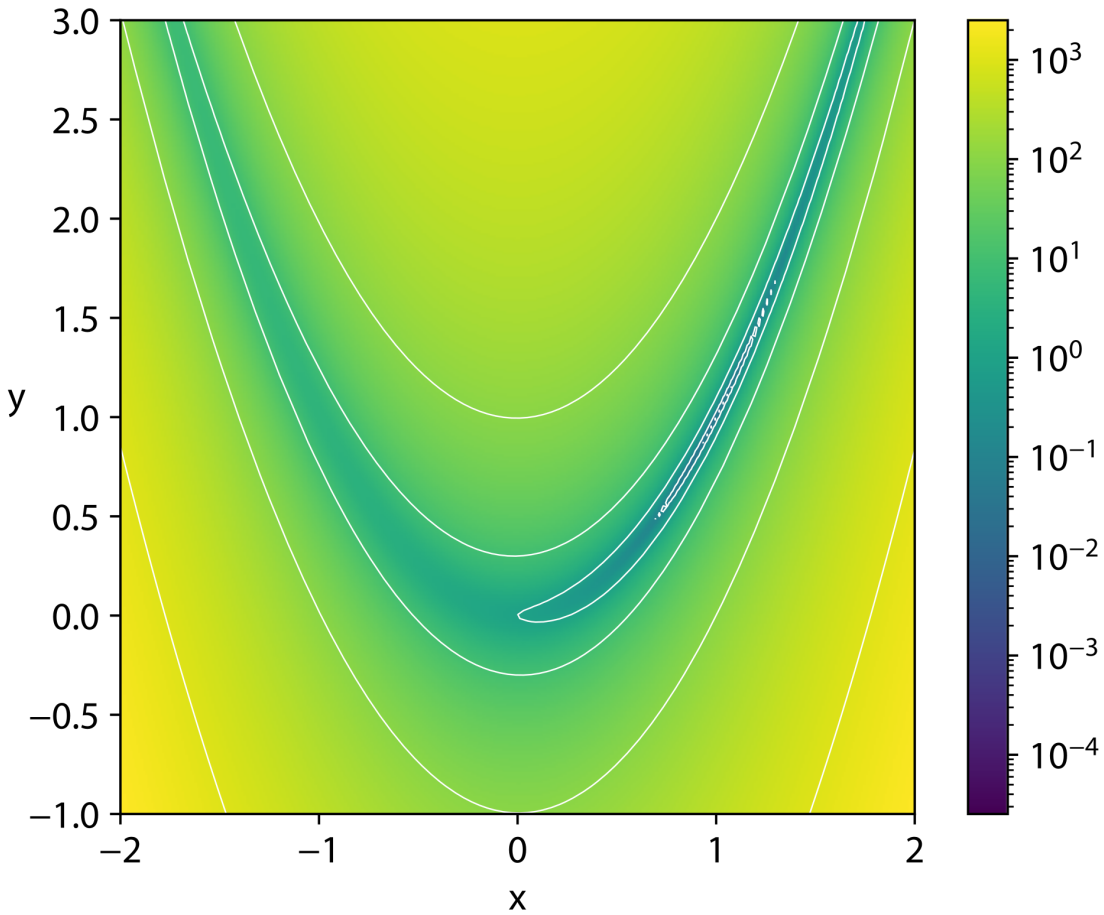
$$f(x, y) = (a - x)^2 + b(y - x^2)^2 \tag{2.1}$$

Formula 2.1 Formula Rosenbrock funkcije za dva ulazna parametra gdje su a i b koeficijenti funkcije

Na slici 2.3 je prikazana Rosenbrock funkcija za dva parametra. Već spomenuta dolina prikazana je na slici plavom bojom, a optimalno rješenje se postiže za $x = 1$ i $y = 1$ i iznosi $f(x, y) = 0$. Rosenbrock funkcija ima nekoliko različitih varijanti, a opća formula najpoznatije varijante glasi:

$$f(x) = \sum_{i=1}^{N-1} [(a - x_i)^2 + b(x_{i+1} - x_i^2)^2], \text{ gdje je } x = (x_1, ..., x_N) \in \mathbb{R}^N \tag{2.2}$$

Formula 2.2 Opća formula Rosenbrock funkcije gdje je x_i vrijednost i-tog parametra, a i b koeficijenti funkcije, a N broj parametara



Slika 2.3 Projekcija 3D prikaza Rosenbrock funkcije za dva ulazna parametra na 2D plohu (preuzeto iz [5])

2.3 Evolucijsko računarstvo

Evolucijsko računarstvo je istraživačko područje unutar računalne znanosti koje svoju inspiraciju crpi iz procesa prirodne evolucije [1]. Polazna točka u evolucijskom računarstvu je okolina koja može biti domaćin samo ograničenom broju jedinki čiji je osnovni instinkt razmnožavanje. Zbog toga, ako populacija ne može rasti eksponencijalno, prirodna selekcija postaje neizbježna i ona favorizira jedinke koje se najučinkovitije natječu za dostupne resurse [1]. Ovaj fenomen je poznat kao preživ-

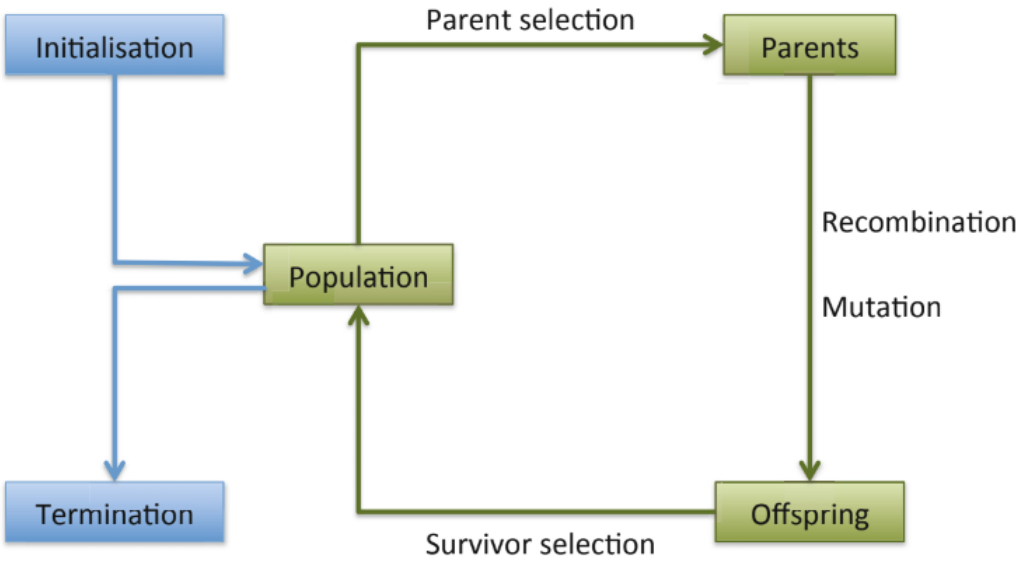
ljavanje najjačih te je odabir temeljen na natjecanju jedan od temelja evolucijskog napretka [1].

U zadnje vrijeme evolucijski algoritmi (EA) su dobili puno pozornosti zbog visokog potencijala za rješavanje složenih problema optimizacije. Pretraživanje korištenjem populacije kandidata rješenja je poprilično robusno s obzirom na umjereni šum i multimodalnost optimizirane funkcije za razliku od nekih klasičnih rješenja [6]. Kao što je već navedeno, glavno ograničenje svih optimizacijskih algoritama pa tako i evolucijskih algoritama jest veliki broj potrebnih procjena mogućih rješenja što sprječava korištenje takvih algoritama nad problemima čije vrednovanje rješenja uključuje poziv nekakve računalno skupe funkcije. Najvažnije komponente evolucijskog algoritma su:

1. predstavljanje, tj. definiranje jedinke,
2. funkcija dobrote,
3. populacija,
4. mehanizam odabira roditelja,
5. operatori varijacije poput križanja i mutacije,
6. mehanizam odabira jedinki radi preživljavanja (prijenos u iduću iteraciju).

Kako bi se stvorio potpun i izvodljiv algoritam, nužno je odrediti svaku komponentu i definirati postupak inicijalizacije [1]. U većini evolucijskih algoritama postupak incijalizacije je jednostavan i služi za generiranje inicijalne populacije.

Također, ako se želi omogućiti ranije zaustavljanje algoritma u nekoj fazi, potrebno je definirati i uvjet prekida. Uvjet prekida se najčešće koristi u radu s problemom za koji je već prethodno poznat optimalan rezultat. U takvom slučaju optimizacijom se nastoji pronaći ulazne parametre čijim se uvrštavanjem u funkciju dobrote dobiva optimalan rezultat. Pri tome, naravno, treba računati da zbog toga što je evolucijski algoritam stohastički proces, ne postoji garancija pronalaska optimalnog rješenja.



Slika 2.4 Prikaz evolucijskog algoritma (preuzeto iz [1])

2.4 Surogatni modeli

Numeričke simulacije su svakodnevno korištene u inženjerskom dizajnu i optimizaciji u svrhu proučavanja ili imitiranja stvarnih fizičkih funkcija [7]. Zbog povećanja razvoja simulacijskog softvera, simulacijski modeli postaju sve složeniji i mogu održavati sve više detalja stvarnih sustava. Stoga provođenje dugih simulacija oduzima dugo vremena i često postaje neisplativo. Učinkovit način za smanjenje vremena pretraživanja, pa tako i trajanja simulacije, je korištenje zamjenskih modela. Takav model djeluje kao model modela i može zamijeniti skupi simulacijski model aproksimacijom njegovih ulazno-izlaznih vrijednosti [7].

Važno je napomenuti da kvaliteta surogatnog modela ima značajan utjecaj na računalne troškove i karakteristike konvergencije optimizacije dizajna temeljene na

surogatnom modelu dok i sama točnost modela može varirati za različite probleme [7]. U svrhu prevencije dobivanja surogatnog modela loše kvalitete, uobičajena je praksa izgraditi više modela i odabrati onaj najbolje kvalitete. Takav pristup može zahtijevati dodatne resurse iako ne iskorištava u potpunosti prednosti izgrađivanja više modela. Alternativna metoda za prevladavanje takvih nedostataka kao i za poboljšanje točnosti aproksimacije je formiranje skupa kombiniranjem pojedinačnih modela pri čemu je svakom dodijeljena određena težina, tj. težinski parametar. Takav skup modela može iskoristiti prednosti svakog pojedinačnog modela smanjujući pogrešku uzrokovanu nestabilnošću jednog surogatnog modela. Zbog želje da cjelokupno programsko rješenje ostane jednostavno, ovaj rad koristi neuronsku mrežu kao surogatni model za aproksimaciju ulazno-izlaznih parametara Rosenbrock funkcije.

Od nekih sličnih radova može se izdvojiti znanstveni članak *On the Selection of Surrogate Models in Evolutionary Optimization Algorithms (Alan D´ıaz-Manr´ıquez, Gregorio Toscano-Pulido and Wilfrido G´omez-Flores)* u kojem autori uspoređuju učinkovitost primjene najpoznatjih surogatnih modela poput Kriginga za rješavanje problema određenih testnih funkcija. Jedna od odabranih testnih funkcija bila je i Rosenbrock funkcija. Istraživanje je pokazalo da se problem Rosenbrock funkcije najučinkovitije rješava ako se za model koristi mreža s radijalnom funkcijom (*eng. radial basis function, RBF*). Takav model je u prosjeku preciznije opisivao testne funkcije (s bržom izradom predviđanja) od preostalih modela (Kriging, polinomni aproksimacijski modeli, regresija algoritmom stroja potpornih vektora). Naime, ovim istraživanjem nisu obuhvaćene neuronske mreže čiju bi primjenu u vidu surogatnih modela trebalo detaljnije istražiti.

Poglavlje 3

METODOLOGIJA

U sklopu programskog rješenja implementirano je nekoliko skripti za obradu podataka, treniranje i testiranje surogatnog modela te optimizaciju parametara višedimenzijske Rosenbrock funkcije.

3.1 Struktura projektnog rješenja

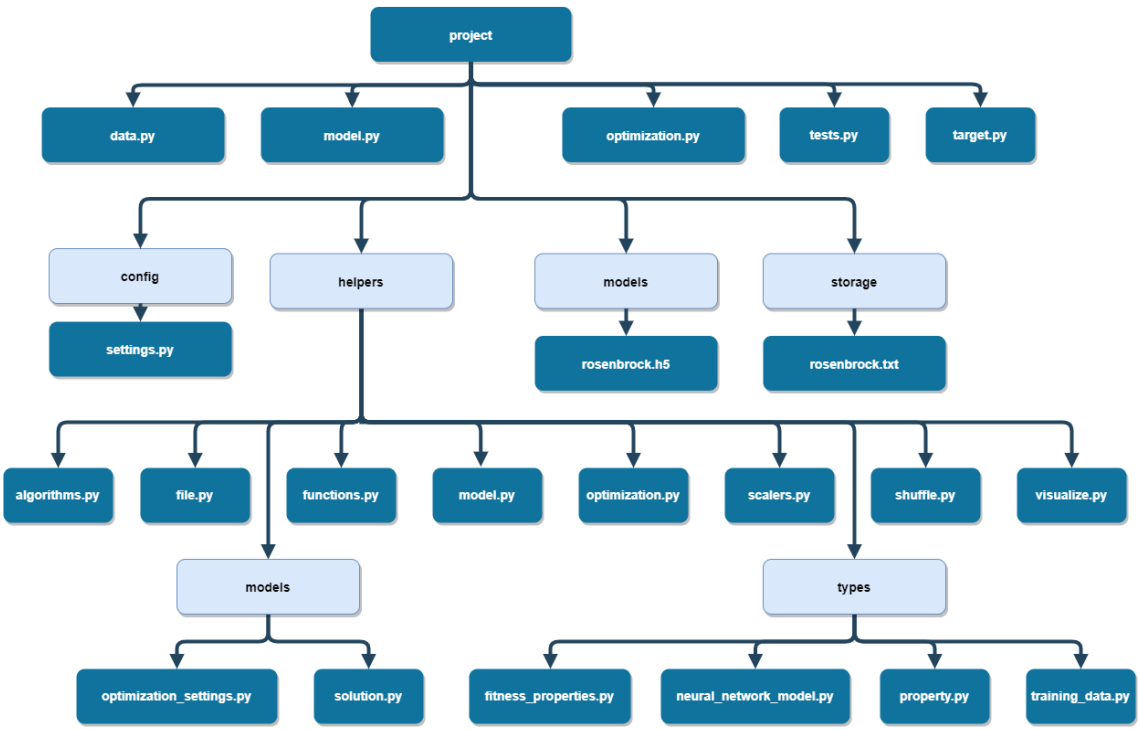
Razvoj tog programskog rješenja obavljen je korištenjem programskog jezika Python te nekolicine dodatnih modula i knjižnica koje je moguće instalirati pomoću Pythonovog sustava za upravljanje softverskim paketima *pip*. Glavni razlog za odabir programskog jezika Python je zbog toga što je Python jedan od najpristupačnijih programskih jezika što mu omogućava široku primjenu. Njegova pristupačnost proizlazi iz pojednostavljene sintakse. Također, Python je najvažniji jezik u podatkovnoj znanosti [8] te je najpopularniji programski jezik na području strojnog učenja (*eng. machine learning*). S obzirom kako je za surogatni model odabrana neuronska mreža, a Python sam po sebi ne omogućava rad s neuronskim mrežama, potrebno je instalirati Tensorflow. Tensorflow je platforma otvorenog koda za strojno učenje razvijena od strane Google Brain tima [9] koja pomoću aplikacijskog programskog sučelja (*eng. application programming interface, API*) Keras omogućava jednostavnu izgradnju i treniranje modela neuronske mreže [10].

Programsko rješenje razdvojeno je na pet Python skripti koje se mogu samostalno

izvršavati. Svaka od tih skripti obavlja specifičan korak u rješavanju cjelokupnog problema. One se nalaze unutar *project* direktorija te su njihova imena:

1. data.py - za generiranje inicijalnog skupa podataka,
2. model.py - za treniranje modela neuronske mreže,
3. optimization.py - pretraživanje optimizacijskim algoritmom pomoću surogatnog modela s ciljem traženja optimalnog rješenja,
4. target.py - pretraživanje genetskim algoritmom dok se ne pronađe rješenje dobiveno jednako ili bolje od onog dobivenim optimizacijskim algoritom (koristi se za procjenu učinkovitosti algoritma),
5. tests.py - za testiranje modela neuronske mreže.

Osim navedenih skripti, bilo je potrebno napisati i desetak pomoćnih skripti koje su odvojene u zasebne direktorije prema funkciji koju obavljaju. Cjelokupnu strukuru programskog rješenja moguće je vidjeti na slici 3.1.



Slika 3.1 Prikaz strukture projektnog rješenja

Glavni razlog razdvajanja programskog rješenja na više skripti je stvaranje mogućnosti pokretanja samo određenog dijela rješavanja problemskog zadatka. To je važno ponajviše zbog toga što je svaki od koraka rješavanja problema složen pa i samim time može imati veće vremensko izvršavanje. Nakon svakog koraka, spremaju se svi podaci potrebni za uspješno izvršavanje nadolazećeg koraka. Primjerice, iako je za Rosenbrock funkciju generiranje inicijalnog skupa podataka vrlo lagano i brzo isto tako ga nije poželjno generirati prilikom svakog treniranja neuronske mreže nego je puno bolja praksa spremanje već generiranog skupa podataka u vanjsku datoteku kako bi bio brzo i lagano dostupan u idućim koracima koji se tek trebaju izvršiti.

3.2 Inicijalni skup podataka

Izrada inicijalnog skupa podataka je prvi korak u rješavanju optimizacijkog problema korištenjem surogatnog modela. To je baza rješenja i pripadnih vrijednosti dobrote koja proizlaze iz nasumičnog pretraživanja i uporabe skupe funkcije dobrote. Njezina važnost proizlazi iz toga što se koristi za treniranje modela neuronske mreže kao i za ugađanje težinskih parametara modela (*eng. fine tuning*) tijekom optimizacije što će biti naknadno objašnjeno. Generiranje inicijalnog skupa podataka izvodi se pokretanjem skripte *data.py* za čije je izvođenje potrebno prethodno odrediti broj dimenzija (tj. parametara) funkcije Rosenbrock kao i veličinu ukupnog skupa podataka. Na kraju izvršavanja ovog koraka, podaci se spremaju u vanjsku datoteku. Primjer takve datoteke može se vidjeti na slici 3.2 gdje su informacije o parametrima kao i završnom rezultatu Rosenbrock funkcije spremljeni unutar tekstualne datoteke. Ta se datoteka može pronaći unutar *storage* direktorija pod imenom *rosenbrock.txt*.

rosenbrock.txt

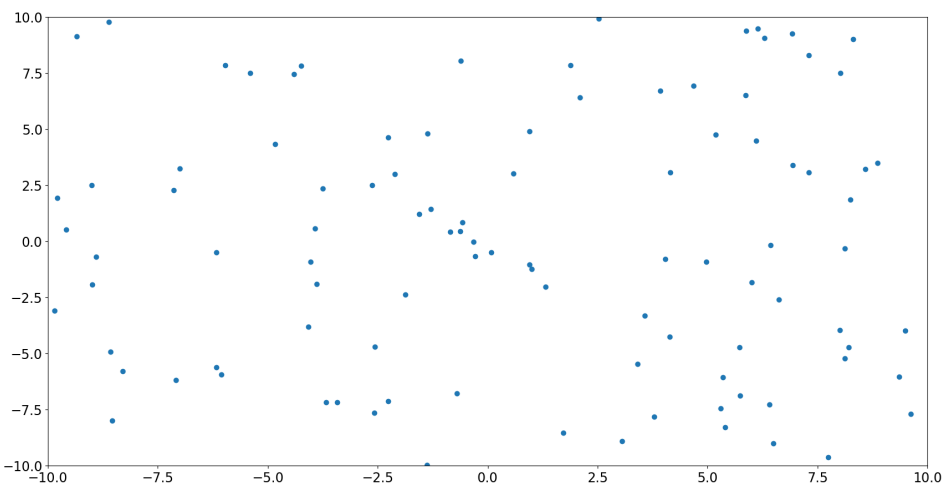
```
2.765494 4.337839 6.25332 -0.751941 3.56292 -6.02595 4.671523 -0.590472 -2.32132 -1.503172 367612.96322
3.819679 4.1544 -2.315655 -3.282702 -3.571495 3.222658 -2.309534 1.161372 -6.828565 -6.39126 392161.272666
2.956956 0.011039 -1.2241 5.06143 -7.418785 -0.3943 -2.962201 -3.946657 -2.416089 5.474392 475157.719341
-3.334638 6.766903 -3.99748 2.027628 -6.433478 0.521558 -4.160202 -2.172405 4.417214 -3.079247 538526.230436
-4.459339 7.318465 7.480496 -3.110457 -3.227886 -1.677763 3.23853 -2.917452 -2.3268 -5.309658 649733.648985
1.507187 -2.375929 6.730336 1.836473 5.508062 0.023955 -6.597779 -6.467707 3.497566 6.791311 687949.97527
-6.456475 -4.374359 -7.069199 -5.441757 4.413465 2.982349 0.380317 -1.469737 -2.674959 7.154325 688695.467093
-5.716299 6.988353 5.049179 -2.670312 0.836292 -1.323409 4.397663 -7.464216 4.250164 7.203844 690824.82998
3.039459 -5.493951 1.722507 8.533106 -1.326951 -0.623591 -6.393926 -0.528865 3.372937 -3.683538 855961.144318
-1.066874 8.099172 3.457815 -5.793227 2.9886 2.562976 3.658082 5.444373 -6.966004 0.730155 889795.757085
6.267821 -2.458858 3.01771 6.38211 3.089169 -7.99941 6.670743 3.168265 -3.358288 -6.866087 898844.289068
-4.669393 7.142821 2.856296 6.97277 8.461239 0.42504 0.065803 -0.064491 -2.106485 -6.765923 934532.070374
-2.835382 7.216416 -7.764998 0.796781 -0.401747 3.727014 -2.438816 -0.108515 -6.888268 -0.680916 980505.563991
-6.8808 4.900986 -0.840493 -4.058902 -4.946591 -8.051447 -1.460793 1.257382 6.118346 -3.056272 1001600.192883
7.074475 6.235595 -7.623156 1.512962 -0.137315 -2.455958 -2.487925 -5.771099 -6.085486 1.325491 1034279.743904
-4.67967 -5.470927 4.15518 4.528355 -7.73201 2.758589 -5.700765 -6.574749 -5.772036 6.760984 1043831.578598
5.195568 -8.809798 -3.783876 -5.938149 -4.232478 0.278087 -1.193761 5.265212 1.57475 4.448162 1089468.786247
-5.514463 4.380492 5.218197 -0.103975 -4.731039 9.038875 5.54912 -6.646464 -0.071705 -8.598389 1105552.412705
-2.092124 9.088498 6.98353 5.055677 -5.378778 -1.768893 6.083199 5.348436 4.022492 6.316978 1126558.838888
7.062192 1.962266 -8.300328 4.886967 -5.722971 -3.022032 -6.668738 -3.934762 2.544222 7.65856 1146117.400516
4.169003 5.350589 -4.111537 7.040988 -1.518955 9.114463 2.323279 -5.965762 1.169049 8.213546 1185448.575231
-4.258676 -2.690041 -6.210632 -3.498702 8.048859 3.218514 5.380805 -8.140903 5.968648 9.491126 1191583.331765
```

Slika 3.2 Prikaz spremljenog inicijalnog skupa podataka

S obzirom kako se optimizacija surogatnim modelima najčešće koristi na problemima za koje je vrednovanje jednog rješenja skup i dugotrajan proces, baza inicijalnih rješenja je bila malena. Najčešće je ukupan broj rješenja bio manji od 1000 dok se dobri rezultati mogu postići i za puno manje skupove podataka (npr. 100 rješenja).

Veličina inicijalnog skupa podataka ovisi broju parametara Rosenbrock funkcije kao i veličini domene iz koje ti parametri mogu poprimiti vrijednosti. Zbog toga što Rosenbrock funkcija brzo raste, tj. vrijednosti se jako razlikuju za malu promjenu parametara, broj parametara Rosenbrock funkcije prilikom provođenja istraživanja nije prelazio 100 dok su vrijednosti parametara nasumično uzimane iz intervala od -10 do 10 uniformnom razdiobom. Za tako postavljene vrijednosti parametara, vrijednost Rosenbrock funkcije na nasumičnim vrijednostima iznosi nekoliko milijardi što označava velik prostor pretraživanja, a samim time i optimizacije. Sustav nije ograničen na te vrijednosti, ali povećavanjem tih vrijednosti uvelike se povećava ukupno trajanje procesa kao i računalna snaga potrebna za rad sustava. Njihovim povećanjem potrebno je povećati broj skrivenih neurona neuronske mreže, produljiti vrijeme treniranja modela povećanjem broja epoha kao i produljiti vrijeme trajanja optimizacije povećanjem broja iteracija genetskog algoritma.

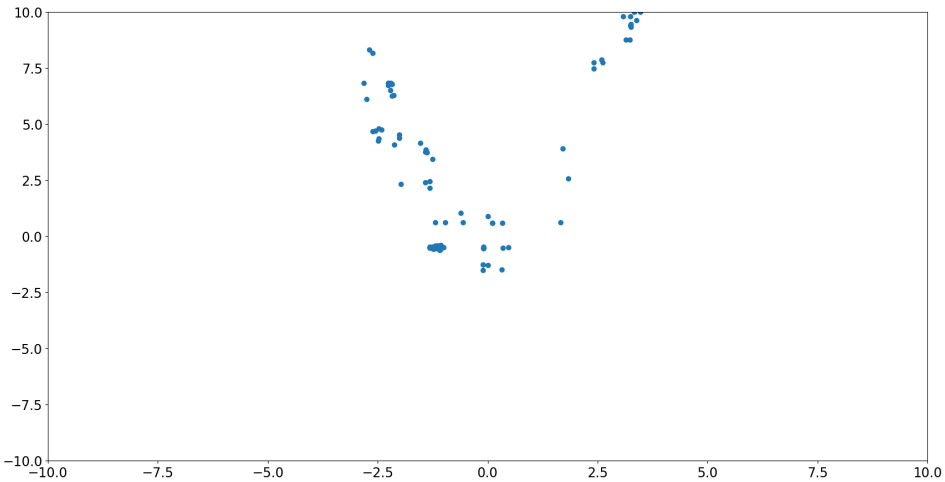
Generiranje inicijalnog skupa podataka izvodi se na način da se se nasumično odabiru vrijednosti parametara za koje se kasnije izračuna stvarna vrijednost Rosenbrock funkcije. Nakon završetka generiranja, podaci se spremaju u već spomenutu izlaznu datoteku na način da svaki redak odgovara jednom rješenju Rosenbrock funkcije. Zadnja vrijednost u redu odgovara vrijednosti Rosenbrock funkcije za odgovarajuće parametre dok sve vrijednosti prije opisuju parametre Rosenbrock funkcije. S obzirom kako se rješenja Rosenbrock funkcije koriste u genetskom algoritmu, sasvim slobodno ih se može nazivati jedinkama pri čemu treba pripaziti da jedinke u genetskom algoritmu koriste drugačiju funkciju dobrote pa samim time imaju i drugačiji krajnji rezultat. Na slici 3.3 prikazana je distribucija nasumično generiranih vrijednosti parametara Rosenbrock funkcije u dvije dimenzije.



Slika 3.3 Prikaz podataka dobivenih nasumičnim odabirom parametara

Još jedan pristup generiranju inicijalnog skupa podataka je i onaj korištenjem genetskog algoritma. Takav pristup moguć je prilagodbom genetskog algoritma korištenog za optimizaciju te bi u tom slučaju algoritam započeo s nasumično generiranim podacima nad kojima bi provodio tehnike evolucijskog računarstva. Ideja algoritma je da se nakon svake iteracije genetskog algoritma izdvoji nekoliko jedinki. Korištenjem takvog algoritma rezultiralo bi skupom podataka u kojem bi se rješenja

postupno mijenjala od lošijih prema boljima. U praksi takav pristup nije rezultirao boljim modelom te je naprotiv, model često bio lošiji nego onaj dobiven iz skupa podataka generiranim nasumičnim odabirom. Pretpostavka je da skup podataka generiran ovim pristupom ima lošiju raznolikost gena (*eng. gene diversity*) pa je zbog toga i dobiveni model lošiji iako je u njegovo cjelokupno treniranje utrošeno više resursa. Na slici 3.4 prikazana je distribucija vrijednosti parametara Rosenbrock funkcije u dvije dimenzije dobivenih korištenjem genetskog algoritma.



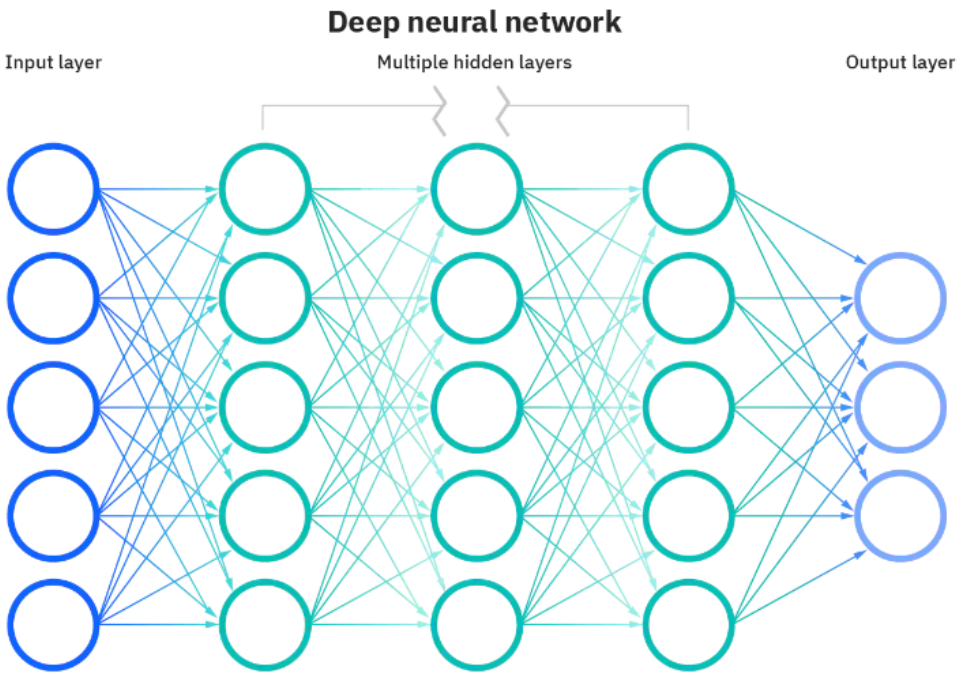
Slika 3.4 Prikaz podataka dobivenih genetskim algoritmom nakon 100 iteracija odabirom jedne najbolje jedinke iz svake iteracije

Prilikom izvođenja ovog koraka za neki problem iz stvarnog svijeta, preporučeno je ručno odabrati ulazne parametre funkcije dobrote (u ovom slučaju Rosenbrock funkcije) kako bi se ostvarila što bolja pokrivenost prostora mogućih rješenja te kako bi surogatni model što bolje opisivao takvu funkciju. S obzirom kako je to poprilično dugotrajan i kompliciran proces, tijekom izrade ovog programskog rješenja korišten je način nasumičnog odabira vrijednosti parametara funkcije dobrote što uvelike pridonosi jednostavnosti cjelokupnog istraživanja.

3.3 Surogatni model

Kao što je već prethodno navedeno, ovo programsko rješenje kao surogatni model koristi neuronsku mrežu. Glavni razlog za odabir neuronske mreže proizlazi iz karakteristika strojnog učenja koje omogućavaju treniranom modelu dobro opisivanje nepoznate funkcije. U ovom slučaju, radi se o poznatoj Rosenbrock funkciji, ali je svejedno moguće primijeniti ovo programsko rješenje na neki problem iz stvarnog svijeta poput izrade cjevovoda nekog grada. U takvom slučaju, potrebno je zamijeniti Rosenbrock funkciju nekom drugačijom funkcijom dobrote i naravno prilagoditi konfiguracijske postavke za rad cjelokupnog programskog rješenja. Konfiguracijske postavke nalaze se u datoteci *settings.py* koja je smještena u *config* direktoriju te se u njoj nalaze razni konfiguracijski parametri poput broja iteracija genetskog algoritma, veličini populacije i broju skrivenih neurona u skrivenom sloju duboke neuronske mreže.

Neuronske mreže, također poznate kao umjetne neuronske mreže, podskup su strojnog učenja i najvažniji dio algoritama dubokog učenja [11]. Sastoje se od slojeva čvorova koji sadrže ulazni sloj, jedan ili više skrivenih slojeva i izlazni sloj. Neuronske mreže oponašaju ponašanje ljudskog mozga, omogućujući računalnim programima da prepoznaju obrasce i riješe uobičajene probleme u područjima umjetne inteligencije, strojnog učenja i dubokog učenja [11]. Duboka neuronska mreža je svaka neuronska mreža koja ima više od tri sloja (uključujuči ulazni i izlazni sloj) dok riječ duboko u dubokom učenju proizlazi iz dubokih neuronskih mreža. Stuktura duboke neuronske mreže prikazana je na slici 3.5.



Slika 3.5 Prikaz duboke neuronske mreže (preuzeto iz [11])

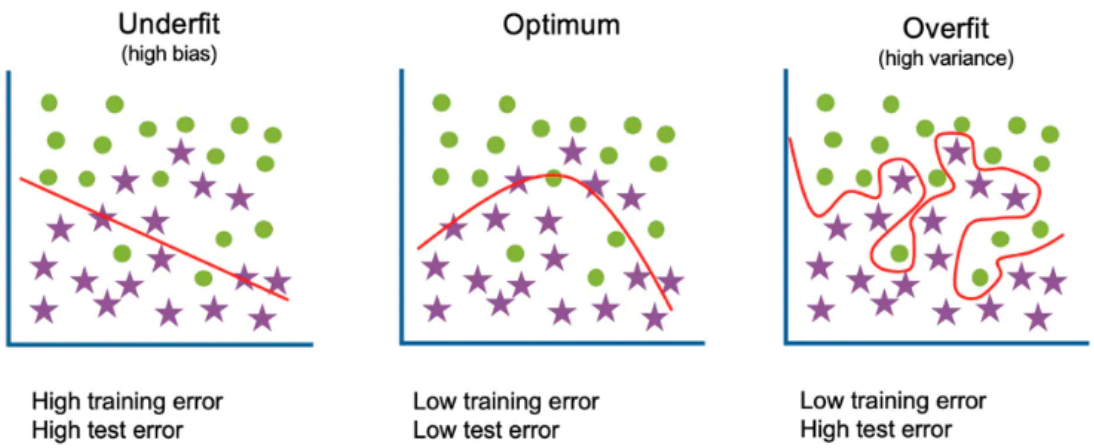
3.3.1 Izgradnja modela

Korak izgradnje i treniranja modela neuronske mreže izvodi se pokretanjem skripte *model.py* za čije izvršavanje mora postojati skup podataka u prethodno definiranom obliku. Izgradnju modela neuronske mreže uvelike olakšava programska knjižnica *Kears*, koja kao što je već navedeno, omogućava jednostavnu izgradnju i treniranje modela neuronske mreže. Na kraju ovog koraka, trenirani model se sprema u direktorij *models* u obliku *h5* datoteke jer je na taj način moguće pohraniti težine i konfiguraciju modela u jednu datoteku.

Prije treniranja modela, izgrađen je model duboke neuronske mreže s pet slojeva. Model se sastoji od jednog ulaznog sloja, tri skrivena sloja i jednog izlaznog sloja. Veličina tj. broj neurona ulaznog sloja ovisi o broju parametara Rosenbrock funkcije dok je veličina izlaznog sloja uvijek 1 i označava predviđenu vrijednost funkcije za ulazne parametre. Broj neurona skrivenih slojeva nije prethodno određen nego se

može konfigurirati prije treniranja te su odabrane vrijednosti iznosile između 64 i 1024 neurona po jednom sloju. Model je izgrađen korištenjem klase *Sequential* koja omogućava dodavanje slojeva neuronske mreže u zadanom poretku. Prema tipu, prethodno navedeni slojevi neuronske mreže su potpuno povezani slojevi koji osim običnih značajki koriste i *L1* regularizaciju. U ovom slučaju, korištenjem tehnika regularizacije nastoji se spriječiti pojavu pretreniranosti (*eng. overfitting*) modela prilikom koje trenirani model postiže dobre rezultate na treniranom skupu podataka dok na drugim podacima ostvaruje lošije rezultate.

Takva pojava je suprotna pojavi podtreniranosti (*eng. underfitting*) modela koja je rezultat prijevremenog završetka treniranja modela neuronske mreže. Neslužbeno, kapacitet modela je njegova sposobnost da odgovara velikom broju funkcija [12]. Modeli s malim kapacitetom mogu imati problema s učenjem nad skupom podataka za treniranje, dok modeli s velikim kapacitetom mogu zapamtiti značajke iz skupa podataka za treniranje koje im ne služe dobro na skupu podataka za provjeru. Pojave podtreniranosti i pretreniranosti modela prikazane su na slici 3.6.

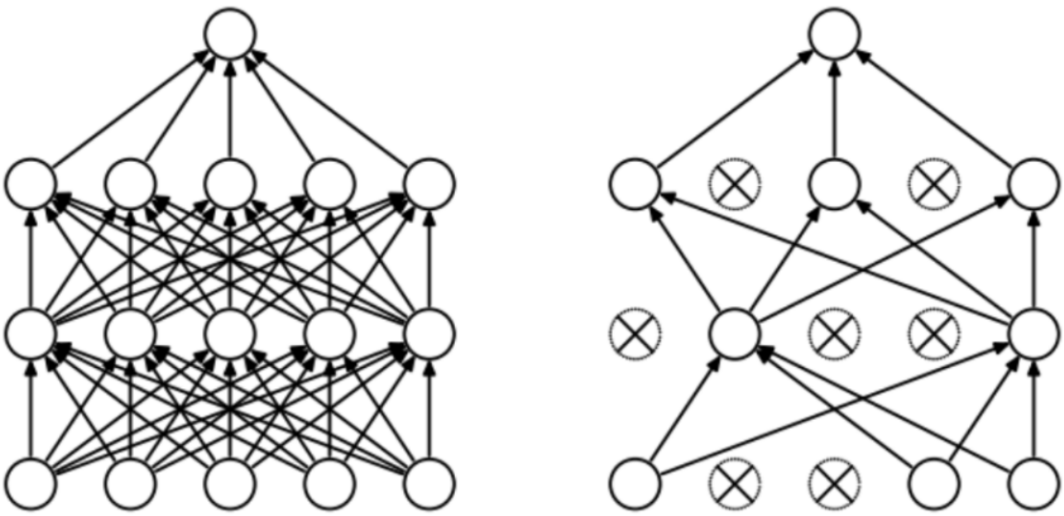


Slika 3.6 Prikaz podtreniranosti i pretreniranosti modela (preuzeto iz [13])

Treniranjem neuronske mreže nad malim skupom podataka, poput onog koji se koristi u ovom programskom rješenju, povećava se rizik od pretreniranosti modela. Stoga je važno odabrati ispravnu tehniku regularizacije. U ovom slučaju odabrana

je *L1* regularizacija kod koje je dodani trošak proporcionalan apsolutnoj vrijednosti težinskih koeficijenata [14].

Osim L1 regularizacije, za sprječavanje pretreniranosti modela neuronske mreže, prilikom izgradnje i testiranja modela korištena je i metoda regularizacije s nasumično izbačenim neuronima (*eng. dropout*) prikazana na slici 3.7. To je tehnika u kojoj se nasumično odabrani neuroni zanemaruju tijekom treninga, tj. oni su nasumično "izbačeni". To znači da se njihov doprinos aktivaciji ostalih neurona na naprijed uklanja dok se bilo kakvo ažuriranje težine ne primjenjuje na neuron prilikom prolaska unatrag. Učinak je da mreža postaje manje osjetljiva na specifične težine neurona. Kao rezultat toga, mreža je sposobna za bolju generalizaciju te se smanjuje vjerojatnost pretreniranosti modela neuronske mreže.



Slika 3.7 Primjena tehnike s nasumičnim izbacivanjem neurona na potpuno povezanu neuronsku mrežu (lijevo) rezultirati će mrežom koja ima veću sposobnost generaliziranja (desno). Izbacivanjem neurona prisiljava se cijeli sustav na učenje više značajki [15]

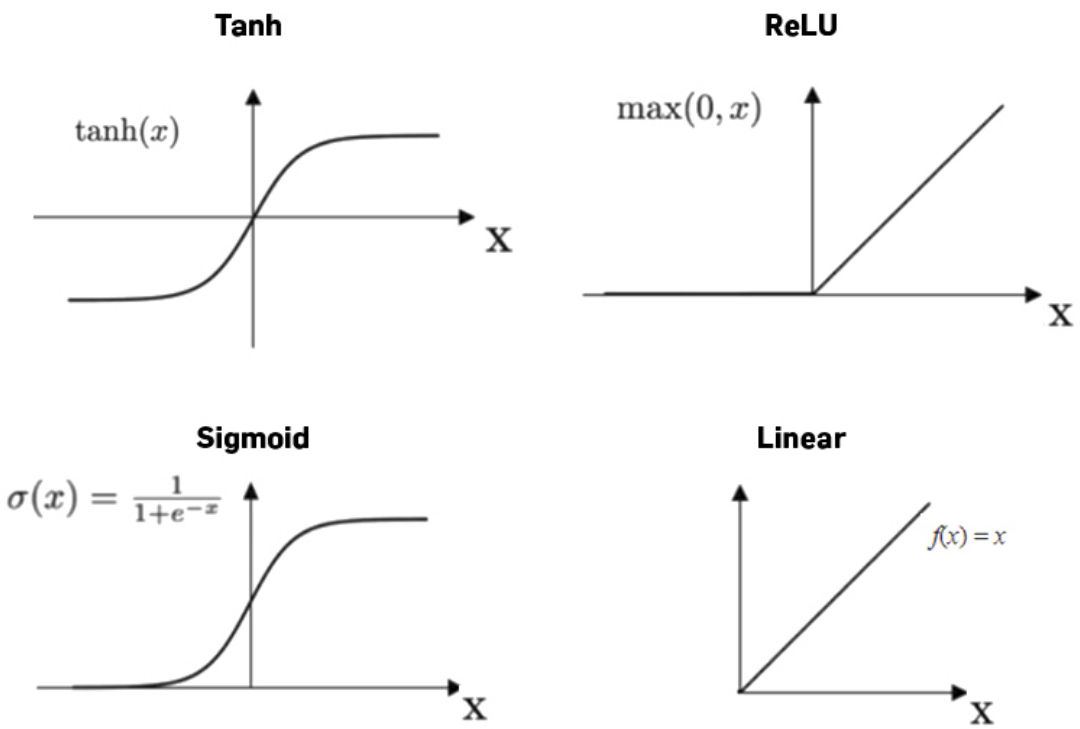
Implementacija tehnika regularizacije je poprilično jednostavna korištenjem programske knjižnice *Keras*. Prilikom izgradnje modela neuronske mreže *L1* regularizacija dodaje se na svaki sloj dok je izbacivanje neurona ostvareno dodavanjem zasebnog sloja nakon sloja u kojem se želi koristiti ta regularizacijska tehnika. Osim regularizacije, potrebno je odrediti i aktivacijsku funkciju (*eng. activation function*)

za svaki sloj. Aktivacijska funkcija odlučuje hoće li se neki neuron aktivirati ili ne te je za potrebe ovog programskog rješenja odabrana ReLU (*eng. Rectified Linear Unit*) aktivacijska funkcija čija jednadžba glasi:

$$f(x) = x^+ = max(0, x) \tag{3.1}$$

Formula 3.1 ReLU funkcija

Glavni razlog za korištenje ReLU aktivacijske funkcije je taj što je jednostavna i brza kao i zbog toga što empirijski ostvaruje dobre rezultate. Empirijski, trening duboke neuronske mreže s ReLU aktivacijskom funkcijom imao je tendenciju bržeg i pouzdanijeg konvergiranja od treninga duboke neuronske mreže sa sigmoidnom aktivacijom. ReLU je samo jedna od nekolicine aktivacijskih funkcija (prikazanih na slici 3.8), ali zbog navedenih prednosti ne čudi njezin odabir.

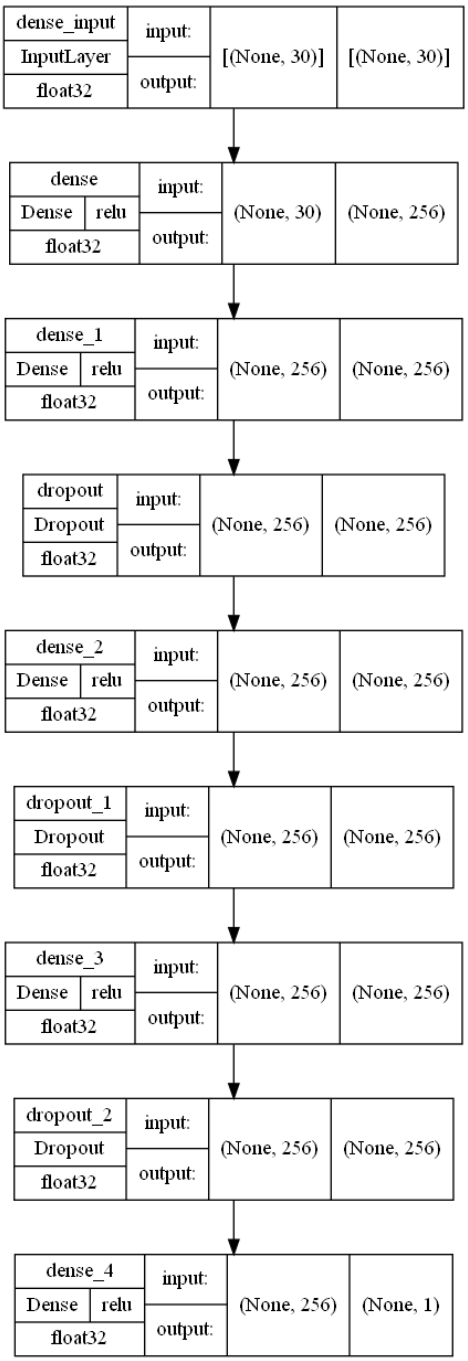


Slika 3.8 Najpoznatije aktivacijske funkcije i njihove formule

Keras pruža način sažimanja modela koji pruža tekstualni zapis modela. Taj zapis uključuje:

1. slojeve modela i njihov redoslijed,
2. izlazni oblik svakog sloja,
3. broj težinskih parametara u svakom sloju,
4. ukupan broj težinskih parametara u modelu.

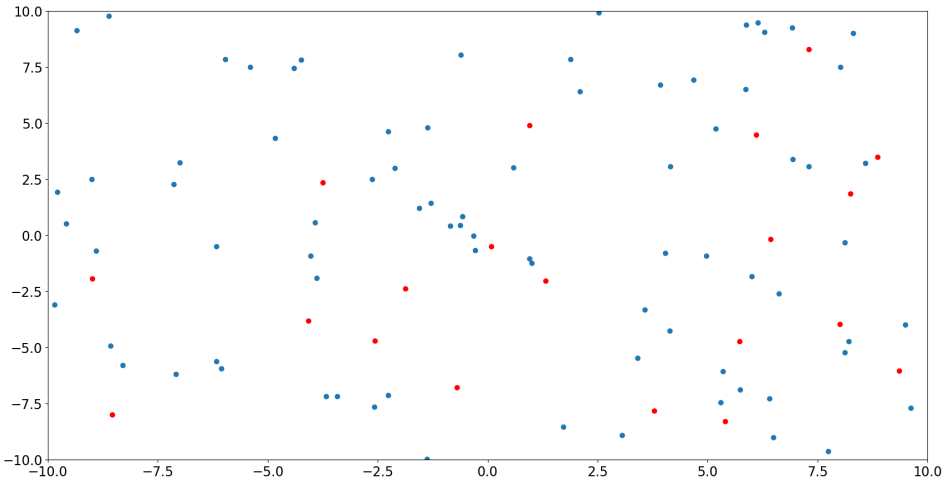
Sažetak se može stvoriti pozivanjem funkcije *summary* na izgrađenom modelu prije ili poslije treniranja. Sažetak je koristan za jednostavne modele, ali može biti zbunjujući za modele koji imaju više ulaza ili izlaza. Keras također pruža funkciju za stvaranje dijagrama mrežnog grafa neuronske mreže koji može olakšati razumijevanje složenijih modela. Korištenjem takve funkcije može se dobiti slika poput one na slici 3.9 koja daje dobar dojam kako izgleda izgrađeni model. Jedna od informacija koje nisu navedene na grafičkom prikazu je ukupan broj težinskih parametara koje model posjeduje, a model prikazan na slici ima ih 205,569.



Slika 3.9 Grafički prikaz izgrađenog modela

3.3.2 Treniranje modela

Prije početka treniranja potrebno je izgraditi model neuronske mreže i učitati prethodno generirani skup podataka. Učitani podaci razdvajaju se na ulazne (vrijednosti parametara Rosenbrock funkcije) i izlazne (vrijednost Rosenbrock funkcije za odabrane parametre) podatke. Prije samog razdvajanja podataka potrebno je promiješati generirani skup podataka. Razlog tome je dodatna podjela skupa podataka (prikazana na slici 3.10 za Rosenbrock funkciju s dva parametra) na skup podataka za treniranje i skup podataka za provjeravanje (*eng. validation*) modela prilikom čega je važno napomenuti kako se skup podataka za provjeru ne koristi za ažuriranje težina neuronske mreže nego prikazuje kako bi se trenirani model ponašao nad nekim podacima s kojima se još nije susreo. Ta dodatna podjela odvija se prema omjeru 80:20 što znači da se 80% podataka koristi za treniranje modela dok se preostalih 20% koristi za provjeru. U određenom slučaju, takva provjera može rezultirati ranijim završetkom treniranja modela. Radi se o metodi ranijeg zaustavljanja koja će zbog svoje važnosti biti malo detaljnije opisana u nastavku.



Slika 3.10 Podjela podataka na skup za treniranje i skup za provjeru, plavom bojom su označena rješenja koja pripadaju skupu za treniranje dok su crvenom bojom označena rješenja iz skupa za provjeravanje modela

Omjer podjele podataka može se dodatno konfigurirati promjenom vrijednosti varijable *TRAINING_ VALIDATION_ SPLIT* koja podržava vrijednosti između 0 i 1 te se nalazi unutar konfiguracijske datoteke *settings.py*. S obzirom da se za treniranje modela koristi poprilično malen skup podataka (do 1000 rješenja), potrebno je ispravno odrediti omjer podjele podataka kako bi skup podataka za treniranje bio što veći, ali i kako bi skup podataka za provjeru dobro generalizirao značajke optimizacijske funkcije, u ovom slučaju Rosenbrock funkcije.

Metrička funkcija (*eng. metric function*) je funkcija koja se koristi za ocjenjivanje izvedbe treniranog modela neuronske mreže [16] dok se funkcija gubitka (*eng. loss function*) koristi za računanje vrijednosti koju bi model trebao minimizirati [17]. Funkcija gubitka u neuronskoj mreži kvantificira razliku između očekivanog ishoda i ishoda proizvedenog modelom te se pomoću nje mogu izvesti gradijenti koji se koriste za ažuriranje težinskih parametara, a metričke funkcije slične su funkcijama gubitka, osim što se njihovi rezultati ne koriste pri učenju modela. Bilo koja funkcija gubitka se može koristiti kao metrička funkcija.

Regresijska analiza je statistička metoda koja pomaže analizirati i razumjeti odnos između dvije ili više interesnih varijabli pa se i sam problem opisivanja Rosenbrock funkcije može klasificirati kao problem koji spada u domenu regresijske analize. S obzirom kako je rješavanje regresijskih problema jedan od najčešćih načina primjene modela strojnog učenja tako i sama programska knjižnica *Keras* nudi nekolicinu metričkih funkcija koje se mogu koristiti prilikom rada s regresijskim problemom. Stoga su korištene sljedeće metričke funkcije:

1. Mean squared error (MSE) - računa srednju vrijednost kvadrata pogrešaka između stvarnih i predviđenih vrijednosti,
2. Mean absolute error (MAE) - računa srednju apsolutnu pogrešku između stvarnih i predviđenih vrijednosti,
3. Mean absolute percentage error (MAPE) - računa srednju apsolutnu postotnu pogrešku između stvarnih i predviđenih vrijednosti.

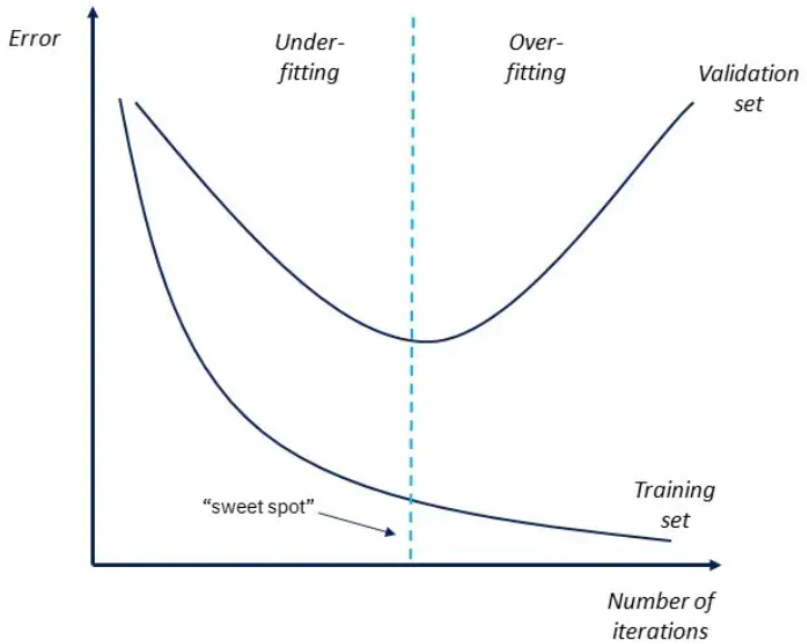
Za funkciju gubitka odabrana je *mean squared error (MSE)* funkcija koja je empirijski pokazala jako dobre rezultate u radu s regresijskim problemima [18] i njezina

formula glasi:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{3.2}$$

Formula 3.2 računanje *MSE* gdje je n broj podataka, y_i stvarna vrijednost i-tog podatka i $\hat{y}_i$ predviđena vrijednost i-tog podatka

Osim navedene konfiguracijske varijable potrebno je odrediti i broj epoha treniranja modela. Epoha je izraz koji se koristi u strojnom učenju i označava broj prolaza cijelog skupa podataka za obuku koji je algoritam strojnog učenja završio. Skupovi podataka obično se grupiraju u skupine (*eng. batches*). Određivanje broja epoha koje neki model treba izvoditi prilikom treniranja često nije moguć bez dobrog razumijevanja podataka. Previše epoha može dovesti do pretreniranosti modela skupu podataka dok premalo epoha može rezultirati podtreniranošću modela. Rano zaustavljanje je metoda koja omogućuje treniranje modela nad velikim brojem epoha uz ranije zaustavljanje treniranja modela nakon što se točnost modela prestane povećavati nad skupom podataka za provjeru. Metoda ranijeg zaustavljanja je još jedna tehnika regularizacije kojom se nastoji spriječiti pretreniranost modela skupu podataka za treniranje te je njezino djelovanje prikazano na slici 3.11.



Slika 3.11 Metoda ranijeg zaustavljanja (preuzeto iz [13])

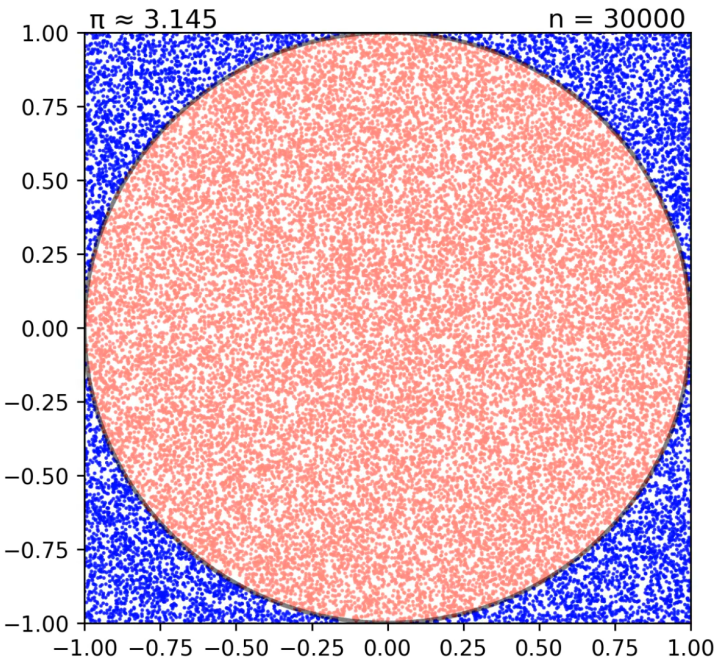
Prilikom izvođenja ovog programskog rješenja broj epoha bio je postavljen na 5000 dok se ranije zaustavljanje izvodilo nakon 100 uzastopnih epoha nad kojima model nije popravio točnost nad skupom podataka za provjeru. To je moguće ponajviše zbog toga što je koeficijent učenja (*eng. learning rate*) modela neuronske mreže postavljen na nisku vrijednost na samom početku te se još dodatno smanjuje nakon svakih nekoliko prolazaka kroz skup podataka korištenjem povratne metode *ExponentialDecay*. Ukoliko stvarno dođe do ranijeg zaustavljanja, tada će povratna metoda vratiti težinske parametre na vrijednosti s kojima je model postigao najbolji rezultat.

3.3.3 Vrednovanje modela

Vrednovanje modela je jedan kratak korak koji se može ali i ne mora izvoditi prilkom rješavanja problema optimizacije korištenjem surogatnih modela. Štoviše, takav

korak zapravo neće biti moguće izvesti prilikom rješavanja stvarnog problemskog zadatka nego unutar ovog programskog rješenja služi samo kao provjera modela prije korištenja istog za optimizaciju. Razlog tome je u visokoj cijeni izvođenja funkcije dobrote (npr. simulacija) što onemogućava generiranje velikog broja rješenja korištenih za vrednovanje modela. Zbog toga što je Rosenbrock funkcija jedna od funkcija za testiranje optimizacijskih algoritama, a i njezino izvođenje je poprilično jeftino, napravljena je skripta koja generira 100 000 nasumičnih rješenja pomoću kojih testira model računanjem pogreške za svako individualno rješenje. Navedeni test je implementiran u obliku Monte Carlo simulacije dok korak testiranja modela započinje pokretanjem skripte *tests.py*.

Monte Carlo simulacija, također poznata kao Monte Carlo metoda ili simulacija višestruke vjerojatnosti, matematička je tehnika koja se koristi za procjenu mogućih ishoda neizvjesnog događaja [19]. Za razliku od normalnog modela predviđanja, Monte Carlo simulacija predviđa skup ishoda na temelju procijenjenog raspona vrijednosti naspram skupa fiksnih ulaznih vrijednosti. Monte Carlo simulacije se također koriste za dugoročna predviđanja zbog svoje točnosti. Kako se broj unosa povećava, raste i broj predviđanja, što omogućava projekciju rezultata dalje u vremenu s većom točnošću [19]. Kada je Monte Carlo simulacija dovršena, ona daje niz mogućih ishoda s vjerojatnošću da se svaki rezultat dogodi. Jedan jednostavan primjer Monte Carlo simulacije je razmatranje izračunavanja vjerojatnosti bacanja dvije standardne kocke. Postoji 36 kombinacija bacanja kockica. Na temelju toga je moguće ručno izračunati vjerojatnost određenog ishoda i zatim se korištenjem Monte Carlo simulacije može simulirati bacanje kocki (npr. 10 000 puta, veći broj bacanja će rezultirati boljim rezultatima) te je na kraju potrebno usporediti ručno izračunate, tj. očekivane rezultate i rezultate dobivene simulacijom. Na slici 3.12 prikazano je računanje π vrijednosti korištenjem Monte Carlo metode.



Slika 3.12 Računanje vrijednosti broja π korištenjem Monte Carlo metode na način da se točke nasumično "ispaljuju" na 2D plohu. Vrijednost broja π se zatim može izračunati pomoću omjera broja točaka unutar kružnice i broja točaka unutar kvadrata, tj. ukupnog broja točaka

Tijek izvršavanja ovog koraka odvija se na način da se generira 100 000 nasumičnih rješenja Rosenbrock funkcije koja se zatim vrednuju korištenjem treniranog modela neuronske mreže. Nakon vrednovanja svih rješenja Rosenbrock funkcije, računa se pogreška. Za potrebe računanja pogreške napisana je jednostavna metrična funkcija koja će podijeliti predviđenu vrijednost individualnog rješenja Rosenbrock funkcije, dobivenu uvrštavanjem njezinih parametara, sa stvarnom vrijednošću Rosenbrock funkcije za iste parametre. Izvođenje ovog koraka nije nužno (niti zapravo moguće za stvarne probleme zbog ograničenog broja vrednovanja rješenja), ali je dobar pokazatelj spremnosti modela neuronske mreže za korištenje unutar optimizacijskog algoritma.

3.4 Optimizacijski proces

Posljednji korak unutar ovog programskog rješenja je korak optimizacije. On započinje pokretanjem skripte *optimization.py* te zapravo predstavlja cijeli optimizacijski proces primjenom surogatnog modela unutar optimizacijskog algoritma. Za optimizaciju se koristi iterativan algoritam koji unutar svake iteracije korištenjem genetskog algoritma pretražuje surogatni model s ciljem traženja optimalnog rješenja. U ovom slučaju, algoritam traži parametre Rosenbrock funkcije za koje model daje najmanju vrijednost. Nakon završetka genetskog algoritma, dobivene vrijednosti parametara Rosenbrock funkcije se vrednuju te se postojeći surogatni model nadopunjuje dodatnim treniranjem korištenjem starih i novih podataka, pri čemu se stariji podaci odnose na inicijalni skup podataka dok noviji podaci predstavljaju parametre Rosenbrock funkcije i stvarnu vrijednost Rosenbrock funkcije dobivene iz prijašnjih iteracija algoritma.

Prilikom rada s problemom optimizacije korištenjem surogatnih modela važno je koristiti takav iterativni pristup zbog toga što model može predvidjeti neke parametre Rosenbrock funkcije kao jako dobre dok se njihovim uvrštavanjem u stvarnu Rosenbrock funkciju može dobiti neko totalno drugačije rješenje. U takvom slučaju, genetski algoritam vraća parametre Rosenbrock funkcije za koje predviđa jako nisku vrijednost dok je stvarna vrijednost značajno veća. Kod iterativnog pristupa, svaka iteracija služi kao kontrolna točka (*eng. checkpoint*) u kojoj se vrednuju najbolja rješenja dobivena genetskim algoritmom (u ovom slučaju samo jedno) te se sprječava dolazak u situaciju u kojoj je potrošeno puno resursa za dobivanje osrednjeg rezultata.

Jedan od načina poboljšavanja učinkovitosti optimizacijskog algoritma je prenošenje genetskog sadržaja između iteracija. To je ostvareno na način da se određeni broj (u ovom slučaju samo jedan) jedinki iz jedne iteracije prenosi u iduću. Zbog toga što je model dotreniran pomoću tih jedinki, one mogu imati veliku važnost unutar daljnjeg optimizacijskog procesa.

Implementirani genetski algoritam se sastoji od nekoliko manjih, ali važnih koraka. Takav genetski algoritam se također izvodi iterativnim pristupom pa se tako nakon generiranja početnog skupa jedinki izvodi iterativni proces. Generiranje jedinki

se izvodi nasumičnim odabirom parametara Rosenbrock funkcije i njihovim vrednovanjem surogatnim modelom dok se unutar svake iteracije genetskog algoritma izvodi određeni broj križanja (*eng. recombination*) čime se stvaraju nove jedinke. Jedinka je definirana listom vrijednosti parametara Rosenbrock funkcije i konačnim rezultatom i kao takva predstavlja jedno rješenje. Cjelokupna Python implementacija jedinke s pomoćnim funkcijama nalazi se unutar klase *Solution*.

Križanje se izvodi nasumičnim odabirom roditelja te nasumičnim odabirom presjeka. Radi jednostavnosti implementacije, algoritam izabire samo dva roditelja te broj presjeka, zbog toga jer ovisi o broju roditelja, iznosi jedan. U ovisnosi o tom presjeku parametri Rosenbrock funkcije nove jedinke se sastoje od vrijednosti parametara prvog roditelja do presjeka i parametara drugog roditelja od presjeka.

Nakon dobivanja nove jedinke križanjem, nad tom jedinkom se provodi mutacija. Implementacija mutacije je vrlo jednostavna i uključuje nasumični odabir jednog parametra Rosenbrock funkcije čijoj će se vrijednosti dodati nasumično odabrana vrijednost. Ta nasumično odabrana vrijednost se izabire iz intervala koji je značajno manji od cjelokupnog intervala vrijednosti koje parametar može poprimiti. Razlog tome je implementacija s ciljem postupnog mijenjanja apsolutne vrijednosti parametara Rosenbrock funkcije kako bi algoritam i u kasnijim stadijima mogao sigurnije napredovati prema boljim rješenjima. Također, prilikom mutacije može doći do promjene predznaka vrijednosti parametra Rosenbrock funkcije kako bi utjecaj mutacije bio značajniji. Takva pojava promjene predznaka vrijednosti parametra se zbog svoje značajnosti rijetko događa.

Na samom kraju iteracije genetskog algoritma izvodi se proces odabira, tj. preživljavanje jedinki. Proces odabira služi kako bi se očuvala veličina populacije te će takva odluka dati prednost onim jedinkama koje imaju bolju vrijednost funkcije dobrote. S obzirom kako se kod Rosenbrock funkcije traži globalni minimum, proces odabira će očuvati jedinke koje imaju najmanju vrijednost, tj. za čije vrijednosti parametara Rosenbrock funkcije surogatni model predviđa najmanju vrijednost. Kod takvog pristupa preživljavanja često dolazi do pojave elitizma (*eng. elitism*) prilikom koje određeni broj jedinki preživljava i prolazi u iduću iteraciju bez križanja i mutacije (npr. roditelji koji se propagiraju kroz iteracije). Elitizam sprječava nasumično uništavanje jedinki s dobrom genetikom dodatnim križanjem ili mutacijom.

Algoritam optimizacije, kao i genetski algoritam može se dodatno konfigurirati. Konfiguracija optimizacijskog algoritma odvija se namještanjem konfiguracijskih parametara unutar konfiguracijske datoteke dok se genetski algoritam konfigurira unutar optimizacijskog algoritma. Konfiguracijski parametri uključuju vrijednosti kao što su broj iteracija optimizacijskog algoritma, broj iteracija genetskog algoritma, broj križanja unutar jedne iteracije genetskog algoritma i njima sličnih parametara. Nakon završetka optimizacijskog algoritma, model se sprema na mjesto prijašnjeg modela te je poželjno izvršiti još jednom korak testiranja i usporediti dobivene vrijednosti pogrešaka prije i poslije optimizacijskog procesa. Nakon optimizacijskog procesa, ukupna greška modela bi trebala biti manja, što je očekivano zbog dodatnog treniranja modela nakon svake iteracije, tj. zbog treniranja modela nad podacima dobivenim optimizacijskim procesom.

Poglavlje 4

REZULTATI

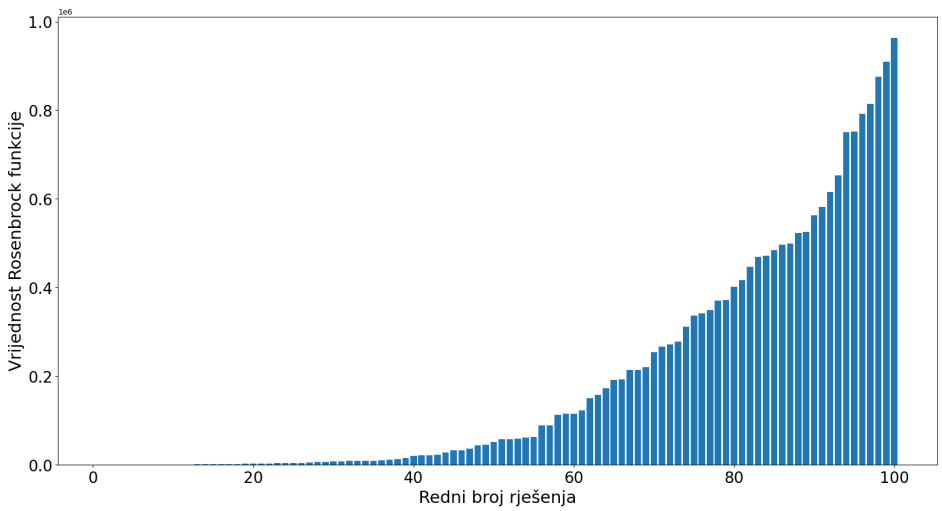
U sklopu ovog rada provedeno je i istraživanje s ciljem određivanja kako se izgrađeni model ponaša u ovisnosti o broju parametara Rosenbrock funkcije. Istražena su 3 slučaja s manjim, srednjim i velikim brojem parametara Rosenbrock funkcije dok su preostale konfiguracijske vrijednosti bile iste za svaki slučaj pa je tako, između ostalog, veličina inicijalnog skupa populacije iznosila 100 dok su vrijednosti parametara Rosenbrock funkcije uzimane na intervalu između -10 i 10.

Promatrane vrijednosti broja parametara Rosenbrock funkcije iznosile su 2, 10 i 100 te je na kraju ovog poglavlja napravljena kratka usporedba ponašanja cjelokupnog optimizacijskog algoritma kao i isplativosti korištenog pristupa. Prije samih rezultata, važno je prisjetiti se cilja, a to je pronalazak najboljeg rješenja s malim brojem izvođenja stvarne Rosenbrock funkcije. Razlog tome jest simulacija okruženja u kojem je ograničen broj pristupanja nekoj skupoj funkciji dobrote tako da se ne može isprobati veliki broj rješenja nego se ta rješenja moraju inteligentno izabirati.

U ovom istraživanju je iskorišteno 100 izvođenja za generiranje inicijalnog skupa podataka dok je dodatnih 20 iskorišteno za optimizaciju provjeravanjem najboljih rješenja dobivenih pomoću surogatnog modela.

4.1 Rezultati za 2 parametra

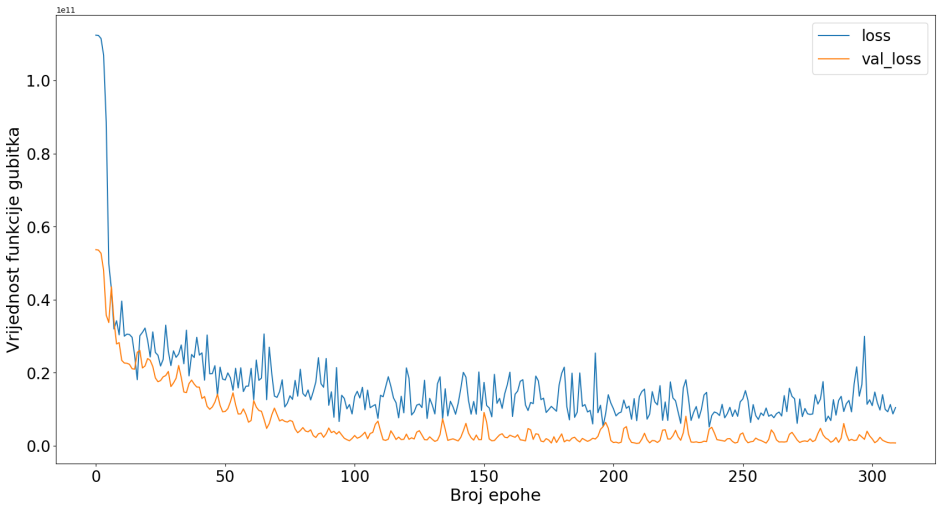
Prije treniranja modela neuronske mreže, potrebno je generirati inicijalni skup podataka. Tako je i u slučaju, u kojem se traži globalni minimum Rosenbrock funkcije za 2 parametra, bilo potrebno generirati inicijalni skup podataka. Generirani skup podataka prikazan je na slici 4.1 i najmanja vrijednost u tom skupu iznosi oko 24.91 dok najveća iznosi malo manje od milijun. Najmanja vrijednost je važna zbog toga jer se kasnije koristi za mjerenje učinkovitosti optimizacije.



Slika 4.1 Prikaz vrijednosti svih rješenja inicijalnog skupa podataka poredana od najmanjeg prema najvećem (Rosenbrock funkcija s 2 parametra)

Pomoću tog skupa podataka trenira se model neuronske mreže. Prije samog treniranja izgrađen je model sa 198,401 težinskih parametara (zbog razlike u veličini ulaznog sloja i broj težinskih parametara varira između različitih slučajeva). Treniranje je završeno metodom ranijeg zaustavljanja nakon 310 epoha. Za funkciju gubitka je korištena već spomenuta MSE metrika te su rezultati treniranja prikazani na slici 4.2. Na slici su prikazane dvije krivulje. Plava krivulja se odnosi na vrijednost funkcije gubitka u određenoj epohi na skupu podataka za treniranje dok žuta krivulja prikazuje vrijednosti funkcije gubitka u određenoj epohi na skupu podataka

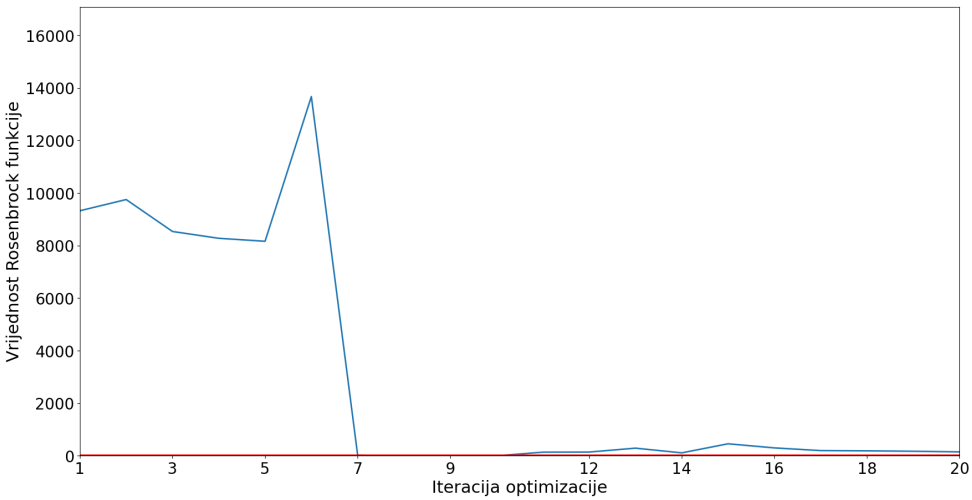
za provjeru modela. Također, iz rezultata treniranja se vidi kako su vrijednosti tih dviju krivulja podjednake. Time se može zaključiti kako je model dobro treniran te bi u suprotnom slučaju trebalo razmisliti o promjeni konfiguracijskih parametara i ponovnom treniranju modela.



Slika 4.2 Rezultati treniranja (Rosenbrock funkcija s 2 parametra)

Proces optimizacije odvija se u 20 iteracija s prenošenjem najbolje jedinke kroz iteracije i dodatnim treniranjem modela između iteracija. Optimizacijskim algoritmom je vrednovano preko milijun jedinki surogantim modelom te su pronađena 3 rješenja koja su bolja od najboljeg rješenja iz inicijalnog skupa podataka. Algoritam je u pravilu svako rješenje precijenio, tj. predvidio mu vrijednost veću od stvarne što može uzrokovati probleme prilikom optimizacije. U ovom slučaju, čini se, to nije slučaj. Sa strane optimizacijskog algoritma najbitnije je da model zna izabrati bolje rješenje između 2 (ili više njih) na temelju njegovih parametara te u slučaju u kojem model precijeni sva rješenja podjednako, algoritam konvergira jednako dobro kao i u radu s pravom Rosenbrock funkcijom. Algoritam je pronašao rješenje čija je stvarna vrijednost Rosenbrock funkcije 1.28 dok je surogatni model za iste parametre predvidio vrijednost od 3059.78. Tijek optimizacije prikazan je na slici 4.3 i na njoj plava krivulja označava promjenu najboljih vrijednosti kroz iteracije dok crvena

linija prikazuje najbolje rješenje inicijalnog skupa podataka.



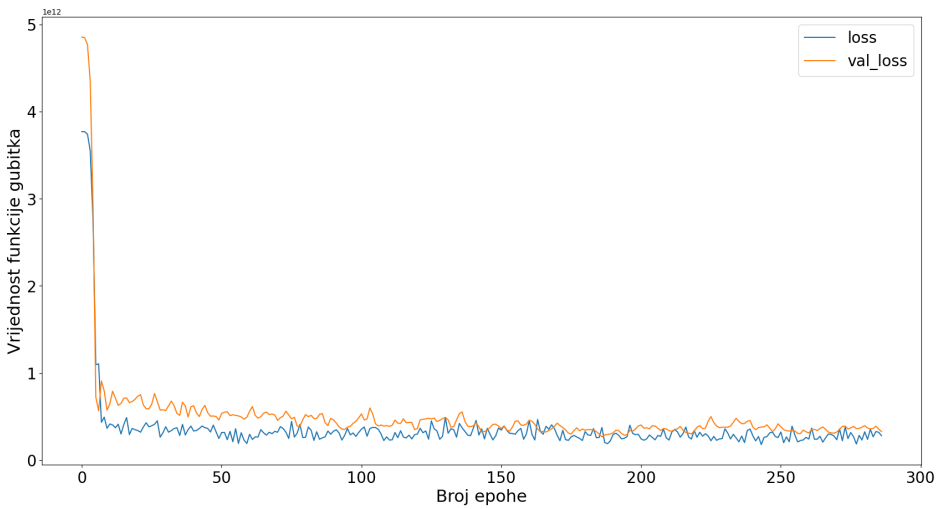
Slika 4.3 Tijek optimizacije (Rosenbrock funkcija s 2 parametra)

Usporedbom najboljeg rješenja dobivenog optimizacijskim algoritmom i najboljeg rješenja iz inicijalnog skupa podataka dolazi se do zaključka da je optimizacijski algoritam pronašao rješenje skoro 20 puta bolje od najboljeg rješenja inicijalnog skupa podataka. Nadalje, korištenjem običnog genetskog algoritma koji kao funkciju dobrote koristi pravu Rosenbrock funkciju bilo je potrebno oko 2200 vrednovanja rješenja te u slučaju u kojem bi se Rosenbrock funkcija zamijenila nekom vremenski skupom funkcijom dolazi do značajne uštede vremena korištenjem optimizacije pomoću surogatnog modela.

4.2 Rezultati za 10 parametara

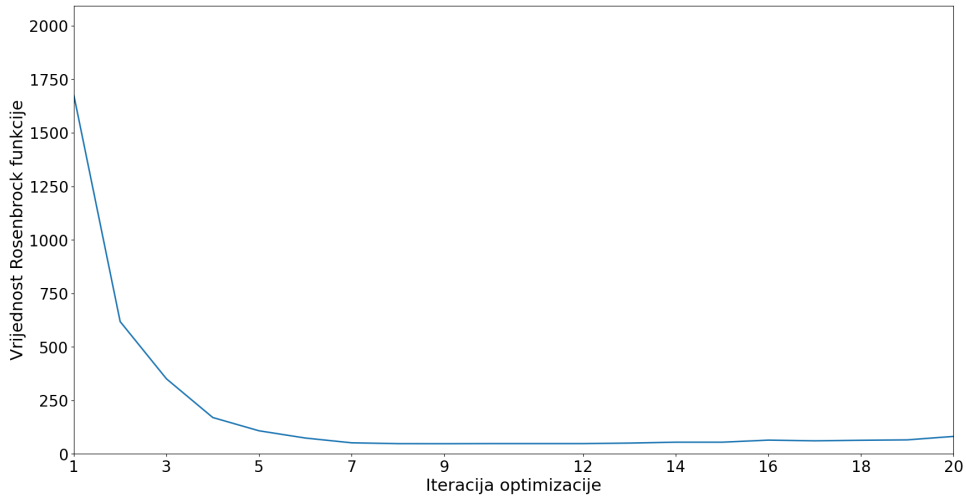
U ovom slučaju se odvija proces isti kao u prošlom. Generiranjem inicijalnog skupa podataka pronalazi se rješenje za koji vrijednost Rosenbrock funkcije iznosi 275,595.06 te se treniranjem modela nad ovakvim skupom podataka dobiva model koji se sastoji od 200,449 težinska parametra. Tijek treniranja vidi se na slici 4.4. Također, i

u ovom slučaju su vrijednosti funkcije gubitka bile podjednake za obje krivulje što ukazuje na dobro izgrađen i treniran model neuronske mreže.



Slika 4.4 Rezultati treniranja (Rosenbrock funkcija s 10 parametara)

U ovom slučaju rezultati optimizacije su iznenađujuće dobri. Sva rješenja dobivena algoritmom optimizacije su višestruko bolja od najboljeg rješenja inicijalnog skupa podataka. Tijek optimizacije je prikazan na slici 4.5. Najbolje rješenje pronađeno je u devetoj iteraciji algoritma i sva rješenja nakon tog su imala sličnu vrijednost. U tom trenutku je algoritam najvjerojatnije zapeo u lokalnom minimumu iz kojeg više nije mogao izaći. Unatoč tome, sva pronađena rješenja su toliko dobra da niti najbolje rješenje iz inicijalnog skupa podataka nije prikazano na slici.

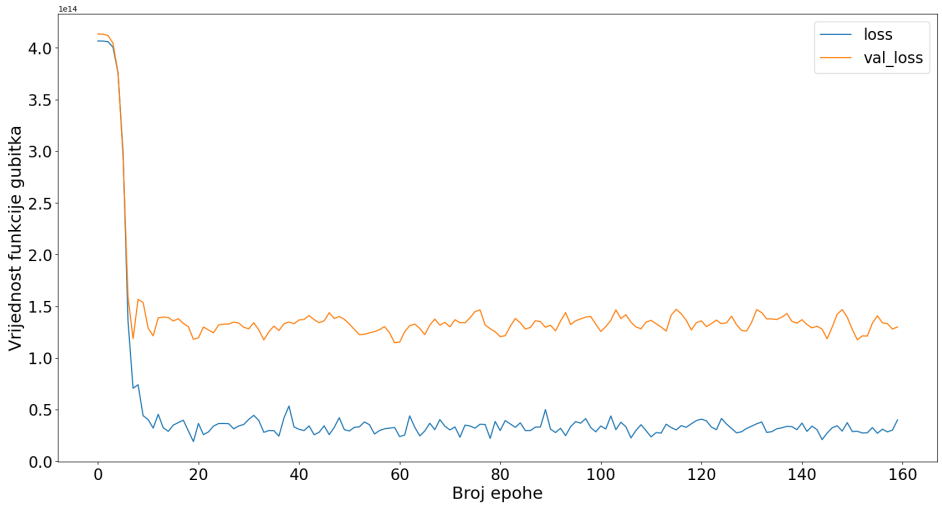


Slika 4.5 Tijek optimizacije (Rosenbrock funkcija s 10 parametara)

Za usporedbu, istom genetskom algoritmu koji za funkciju dobrote koristi pravu Rosenbrock funkciju za traženje istog ili boljeg rezultata treba najmanje 70000 izvođenja Rosenbrock funkcije.

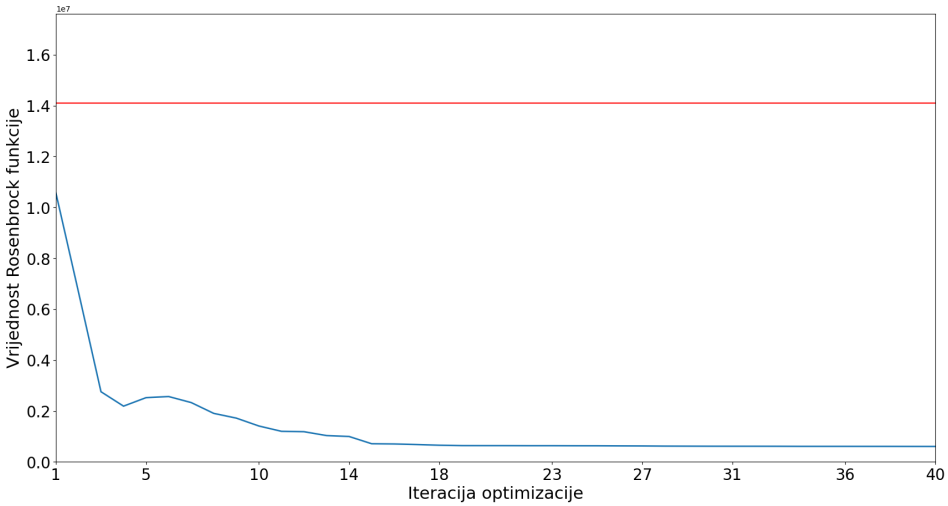
4.3 Rezultati za 100 parametara

I u ovom slučaju je bilo potrebno generirati inicijalni set podataka, te nakon generiranja, najbolja rješenje ima vrijednost od 14,091,467.96. Trenirani model sastoji se od 223.489 težinskih parametara te je sam tijek treniranja prikazan na slici 4.6. Na slici se vidi čak poprilična razlika između dviju krivulja. Razlog tome je najvjerojatnije veličina skupa podataka za treniranje zbog toga što model pokušava naučiti značajke za 100 parametara iz samo 80 rješenja (preostalih 20 se koristi za vrednovanje modela).



Slika 4.6 Rezultati treniranja (Rosenbrock funkcija sa 100 parametara)

S obzirom na prijašnje slučajeve, rezultati optimizacije su razočaravajući. Pronađeno rješenje je skoro tri puta bolje od najboljeg rješenja iz inicijalnog skupa podataka i iznosi 5,226,245.27. Tijek optimizacije vidljiv je na slici 4.7. Traženje sličnog rješenja običnim genetskim algoritmom zahtijeva preko 7000 izvođenja Rosenbrock funkcije. Iako rješenje nije zadovoljavajuće i dalje se vidi veliki doprinos korištenja surogatnih modela s obzirom za broj izvođenja Rosenbrock funkcije (120 naprema 7000).



Slika 4.7 Tijek optimizacije (Rosenbrock funkcija sa 100 parametara)

Zbog lošijih rezultata odlučeno je da se eksperiment provede još jednom uz jednu izmjenu, a to je broj iteracija optimizacijskog algoritma. U slučaju u kojem se optimizacijski algoritam izvodi u 40 iteracija umjesto u 20, pronalazi se rješenje koje je 20 puta bolje od najboljeg iz inicijalnog skupa podataka. Vrijednost takvog rješenja iznosi 608,669.52 i traženje takvog rješenja običnim genetskim algoritmom zahtijeva približno 100,000 izvođenja prave Rosenbrock funkcije. U slučaju u kojem se Rosenbrock funkcija zamijeni nekom drugačijom, ali vremenski zahtjevnijom funkcijom, sama mogućnost optimizacije postaje upitna zbog ukupne vremenske složenosti.

4.4 Usporedba rezultata

U svakom od tri provedena eksperimenta algoritam je pronašao rješenje višestruko bolje od najboljeg rješenja iz inicijalnog skupa stoga se može zaključiti kako je istraživanje uspješno provedeno. U većini slučajeva pronađeno rješenje je bilo zadovoljavajuće. S druge strane, optimizacijski algoritam je ponekad nailazio na probleme i jednostavno nije uspijevao redovno pronaći rješenja bolja od onih iz inicijalnog skupa podataka. To se najbolje vidi na slučaju s velikim brojem parametara u ko-

jem je algoritam tek nakon dodatnih izmjena u konfiguraciji počeo pronalaziti dobra rješenja. Glavni zaključak istraživanja jest da će se vjerojatnost pronalaska zadovoljavajućeg rješenja povećati sukladno povećanju ukupnog vremena optimizacije povećanjem broja iteracija optimizacijskog algoritma pri čemu je važno redovno zanemariti genetsko prenošenje kako bi se izbjegao dolazak u lokalni optimum. Iako se takav zaključak ponajviše temelji na slučaju s velikim brojem parametara, njega se može primijeniti uvijek.

S obzirom da je cilj istraživanja bilo traženje što boljeg rješenja uz što manji broj pozivanja originalne Rosenbrock funkcije, konačno dobivena rješenja su i dalje jako dobra. Optimizacijski algoritam je za funkciju s 10 parametara pronašao rješenje za koje bi običnom genetskom algoritmu s originalnom Rosenbrock funkcijom trebalo dodatnih 6000 izvođenja Rosenbrock funkcije. Možda se to u ovom trenutku ne čini toliko impresivno, ali za kompanije poput Forda, kojima za izvođenje jedne simulacije prometne nesreće treba između 36 i 160 sati [20], takva ušteda jako puno znači.

Poglavlje 5

OPTIMIZACIJA HIPERPARAMETARA

Prije uspoređivanja surogatnog modela iz ovog rada s ostalim surogatnim modelima važno je napraviti podešavanje hiperparametara (*eng. hyperparameter tuning*). Podešavanje, tj. ugađanje hiperparametara je jedna od najvažnijih stvari u kontroli ponašanja modela strojnog učenja. S obzirom kako ovaj rad koristi surogatni model u obliku neuronske mreže, podešavanje hiperparametara može značajno promijeniti ponašanje cjelokupnog optimizacijskog algoritma. Neispravno podešeni hiperparametri će u ovom slučaju rezultirati lošijim modelom neuronske mreže, što znači da će trenirani model lošije opisivati funkciju dobrote.

Sa strane strojnog učenja važno je razlikovati parametre od hiperparametara. Parametri se odnose na vrijednosti koje algoritam strojnog učenja procjenjuje za dani skup podataka te se ažuriranjem tih parametara odvija učenje modela. Hiperparametri su, s druge strane, specifični za sam algoritam što znači da se njihove vrijednosti ne mogu izračunati iz podataka. Oni se odnose na vrijednosti poput broja skivenih slojeva ili čak broja neurona određenog sloja. Hiperparametri kontroliraju ponašanje modela te se njihovo podešavanje sastoji od pronalaženja optimalnih vrijednosti hiperparametara za koje se maksimizira izvedba modela.

Osim podešavanja hiperparametara, potrebno je također podesiti i parametre evolucijskog algoritma. Takvi parametri opisuju ponašanje evolucijskog (u ovom

slučaju genetskog) algoritma. Oni opisuju vrijednosti poput veličine populacije, vjerojatnosti mutacije ili broj iteracija. Iako se procesi podešavanja hiperparametara algoritma strojnog učenja i podešavanja parametara evolucijskog algoritma mogu obaviti jednim procesom (može se nazvati optimizacija hiperparametara) to nažalost nije pokriveno u ovom radu. Naime, implementacija takvog procesa može se ostvariti na primjer korištenjem još jednog evolucijskog algoritma, ali takav pristup osim složenije implementacije zahtijeva dodatne računalne resurse i postavlja dodatna pitanja poput onog o načinu vrednovanja pojedinog rješenja algoritma za optimizaciju hiperparametara. Radi jednostavnosti implementacije, donešena je konačna odluka protiv implementacije dodatnog optimizacijskog algoritma unatoč svim prednostima koje ona donosi.

Poglavlje 6

ZAKLJUČAK

Potreba za rješavanjem problema optimizacije danas je veća nego ikada prije što u kombinaciji s povećanom sposobnosti inženjera u izgradnji složenih simulacijskih modela znatno otežava proces optimizacije. Ovaj diplomski rad izrađen je u sklopu Erasmus+ projekta pod naslovom "Promoting Sustainability as a Fundamental Driver in Software development Training and Education" s oznakom 2020-1-PT01-KA203-078646 i njegov cilj bio je istražiti primjenu surogatnog modela u obliku neuronske mreže i analizirati njegovu učinkovitost na primjeru optimizacije Rosenbrock funkcije.

Implementiran je sustav koji genetskim algoritmom pretražuje prostor mogućih rješenja s ciljem traženja globalnog minimuma. Sva rješenja dobivena genetskim algoritmom vrednuju se surogatnim modelom koji služi kao aproksimacijski model Rosenbrock funkcije, dok se najbolja rješenja vrednuju i originalnom Rosenbrock funkcijom u svrhu dobivanja informacija o tijeku optimizacije kao i dopunjavanja modela novim značajkama. Nakon samo 120 poziva originalne Rosenbrock funkcije, genetski algoritam koji koristi surogatni model pronašao je rješenje za čiji pronalazak istom takvom genetskom algoritmu bez korištenja surogatnog modela treba najmanje 70,000 poziva originalne Rosenbrock funkcije. Korištenjem takvog pristupa optimizacije nad vremenski skupim simulacijama, poput onakvih kakve kompanija Ford koristi za simuliranje prometnih nesreća, može se ostvariti ušteda od preko 11 milijuna sati. Dodatnim doradama može se osigurati pouzdaniji rad optimizacijskog algoritma te su takve dorade, poput optimizacije hiperparametara, navedene u samom radu.

Bibliografija

[1] J. S. A.E. Eiben, *Introduction to Evolutionary Computing*. Springer, 2015.

[2] J. Ma and H. Li, "Research on rosenbrock function optimization problem based on improved differential evolution algorithm," *Journal of Computer and Communications*, vol. 07, pp. 107–120, 01 2019.

[3] K. Hussain, M. Salleh, S. Cheng, and R. Naseem, "Common benchmark functions for metaheuristic evaluation: A review," *International Journal on Informatics Visualization*, vol. 1, pp. 218–223, 11 2017.

[4] Rastrigin function. , s Interneta, https://en.wikipedia.org/wiki/Rastrigin_function , lipanj 2022.

[5] Rosenbrock function. , s Interneta, https://en.wikipedia.org/wiki/Rosenbrock_function , lipanj 2022.

[6] I. Loshchilov, "Surrogate-assisted evolutionary algorithms," 01 2013.

[7] P. Jiang, Q. Zhou, and X. Shao, *Surrogate Model-Based Engineering Design and Optimization*, 01 2020.

[8] Kaggle: Python. , s Interneta, https://www.kaggle.com/learn/python , lipanj 2022.

[9] Tensorflow (github). , s Interneta, https://github.com/tensorflow/tensorflow , lipanj 2022.

[10] Tensorflow. , s Interneta, https://www.tensorflow.org/about , lipanj 2022.

[11] Neural networks. , s Interneta, https://www.ibm.com/cloud/learn/neural-networks , lipanj 2022.

[12] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[13] Overfitting. , s Interneta, https://www.ibm.com/cloud/learn/overfitting , lipanj 2022.

[14] Overfit and underfit. , s Interneta, https://www.tensorflow.org/tutorials/keras/overfit_and_underfit , lipanj 2022.

[15] C. F. G. dos Santos and J. P. Papa, "Avoiding overfitting: A survey on regularization methods for convolutional neural networks," *ACM Computing Surveys*, jan 2022. , s Interneta, https://doi.org/10.1145%2F3510413

[16] Metrics. , s Interneta, https://keras.io/api/metrics/ , lipanj 2022.

[17] Losses. , s Interneta, https://keras.io/api/losses/ , lipanj 2022.

[18] J. Brownlee, *Better Deep Learning: Train Faster, Reduce Overfitting, and Make Better Predictions*. Machine Learning Mastery, 2018.

[19] Monte carlo simulation. , s Interneta, https://www.ibm.com/cloud/learn/monte-carlo-simulation , lipanj 2022.

[20] X. Cai, L. Gao, and X. Li, "Efficient generalized surrogate-assisted evolutionary algorithm for high-dimensional expensive problems," *IEEE Transactions on Evolutionary Computation*, vol. PP, pp. 1–1, 05 2019.

Sažetak

U radu je opisan optimizacijski problem traženja globalnog minimuma Rosenbrock funkcije korištenjem surogatnog modela. Kao surogatni model korištena je neuronska mreža kojom se napravila aproksimacija Rosenbrock funkcije. Razvijeno je kompletno rješenje koje nudi izradu inicijalnog skupa podataka, izgradnju i treniranje modela neuronske mreže kao i optimizaciju korištenjem evolucijskog algoritma. Implementiran je evolucijski algoritam koji vrednuje rješenja pomoću surogatnog modela. U odnosu na evolucijski algoritam koji vrednuje rješenja korištenjem Rosenbrock funkcije, implementirani algoritam je pronašao jednako dobro rješenje uz uštedu od 70,000 izvršavanja Rosenbrock funkcije. Zbog uštede u ukupnom broju izvršavanja funkcije dobrote, korištenje surogatnih modela omogućava rješavanje problema optimizacije u slučaju u kojem vrednovanje jednog rješenja zahtjeva izvršavanje vremenski skupe funkcije.

Ključne riječi **— surogatni model, optimizacija, evolucijski algoritam, duboko učenje**

Abstract

The paper describes the optimization problem of searching for the global minimum of the Rosenbrock function using a surrogate model. As a surrogate model, a neural network was used to approximate the Rosenbrock function. A complete solution has been developed that offers creation of an initial data set, construction and training of a neural network model, as well as optimization using an evolutionary algorithm. An evolutionary algorithm was implemented that evaluates solutions using a surrogate model. Compared to the evolutionary algorithm that evaluates solutions using the Rosenbrock function, the implemented algorithm found an equally good solution with a saving of 70,000 executions of the Rosenbrock function. Due to the savings in the total number of executions of the fitness function, the use of surrogate models enables solving the optimization problem in the case where the evaluation of one

solution requires the execution of a time-consuming function.

Keywords **— surrogate model, optimization, evolutionary algorithm, deep learning**