ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

FAKULTA DOPRAVNÍ

Tereza Panská

VYUŽITÍ EVOLUČNÍCH TECHNIK PŘI UČENÍ UMĚLÝCH NEURONOVÝCH SÍTÍ

Bakalářská práce

2022

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta dopravní
děkan
Konviktská 20, 110 00 Praha 1

K614..... **Ústav aplikované informatiky v dopravě**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE
(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení studenta (včetně titulů):

Tereza Panská

Studijní program (obor/specializace) studenta:

bakalářský – ITS – Inteligentní dopravní systémy

Název tématu (česky): **Využití evolučních technik při učení umělých neuronových sítí**

Název tématu (anglicky): Using evolutionary techniques for ANN learning

Zásady pro vypracování

Při zpracování bakalářské práce se řiďte následujícími pokyny:

- Prostuduje principy genetických algoritmů a ostatních evolučních technik
- Prostudujte existující aplikace evolučních technik na učení neuronových sítí a zhodnoťte je
- Prostudujte existující knihovny implementující neuronové sítě a evoluční techniky ve vybraných programovacích jazycích
- Demostrujte využití popisovaných knihoven

ČVUT
FD

Rozsah grafických prací: dle pokynů vedoucího práce

Rozsah průvodní zprávy: minimálně 35 stran textu (včetně obrázků, grafů a tabulek, které jsou součástí průvodní zprávy)

Seznam odborné literatury: Ivan Nagy: Základu Bayesovského odhadování a řízení, 2003

Mařík a kol.: Umělá inteligence 1 - 3

Zelinka a kol.: Evoluční techniky

Vedoucí bakalářské práce: **doc. Ing. Vít Fábera, Ph.D.**

Datum zadání bakalářské práce: **8. října 2021**
(datum prvního zadání této práce, které musí být nejpozději 10 měsíců před datem prvního předpokládaného odevzdání této práce vyplývajícího ze standardní doby studia)

Datum odevzdání bakalářské práce: **8. srpna 2022**
a) datum prvního předpokládaného odevzdání práce vyplývající ze standardní doby studia a z doporučeného časového plánu studia
b) v případě odkladu odevzdání práce následující datum odevzdání práce vyplývající z doporučeného časového plánu studia

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE Fakulta dopravní

L. S.

Fábera V.

doc. Ing. Vít Fábera, Ph.D.
vedoucí
Ústavu aplikované informatiky v dopravě

-2-

doc. Ing. Pavel Hrubeš, Ph.D.
děkan fakulty

Potvrzuji převzetí zadání bakalářské práce.

Panská

Tereza Panská
jméno a podpis studenta

V Praze dne ..... 8 října 2022

Poděkování

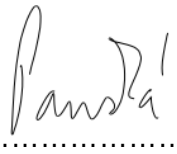
Děkuji svému vedoucímu doc. Ing, Vítu Fáberovi, Ph.D. za vedení bakalářské práce, užitečné rady a připomínky a vstřícnost při konzultacích týkajících se bakalářské práce.

Prohlášení

„Nemám závažný důvod proti užívání tohoto školního díla ve smyslu § 60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).“

„Prohlašuji, že jsem předloženou práci vypracovala samostatně a že jsem uvedla veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.“

V Praze dne 7. 8. 2022



……

podpis

Abstrakt

Předmětem bakalářské práce „Využití evolučních technik při učení neuronových sítí“ je popis jednotlivých technik a jejich existujících aplikací při trénování neuronových sítí. Popis je zaměřen především na genetické algoritmy a genetické operátory (selekce, křížení, mutace), které jsou používány v genetických algoritmech. Dále je zde přiblížen problém při učení neuronových sítí a jsou představena možná řešení pomocí evolučních technik. Poslední část se zabývá knihovnami implementujícími evoluční techniky a neuronové sítě a demonstrací jejich využití.

Abstract

The subject of the bachelor‘s thesis “Using evolutionary techniques for ANN learning” is a description of particular techniques and their existing applications in the neural network learning. The description is mainly focused on genetic algorithms and genetic operators (selection, crossover, mutation) that are used in genetic algorithms. Next, the learning problem of neural networks is presented alongside possible solutions using evolutionary techniques. The final section covers libraries that implement evolutionary techniques and neural networks and a demonstration of their use.

Klíčová slova

evoluční techniky, fitness funkce, genetické algoritmy, jedinec, křížení, mutace, neuronové sítě, selekce

Keywords

evolution algorithms, fitness function, genetic algorithm, individual, crossover, mutation, neural networks, selection

Obsah

Seznam použitých zkratek

ADAM	Adaptive Moment Estimation metoda
BP	backpropagation
CNN	konvoluční neuronová síť
DE	diferenciální evoluce
DS	deterministické vzorkování
ES	evoluční strategie
ET	evoluční techniky
FPS	fitness proporcionální selekce
GA	genetické algoritmy
GP	genetické programování
LSTM	long short-term memory síť
NS	neuronové sítě
PSO	optimalizace hejnem částic
RNN	rekurentní neuronová síť
SRS	stochastické zbytkové vzorkování
SUS	stochastické univerzální vzorkování

1 Úvod

Umělé neuronové sítě jsou výpočetní systémy umělé inteligence, které jsou inspirované fungováním biologických neuronových sítí a mozku. Tyto systémy jsou schopny provádět několik dílčích operací najednou a jsou navrženy tak, aby se neustále učily a zlepšovaly. Jakmile je systém naučen, může produkovat výstupy bez potřeby úplných vstupů. Tudíž neuronové sítě dokáží řešit komplikované úlohy, jako je zpracování přirozeného jazyka, rozpoznávání řeči a obrazů. Díky svému rostoucímu využití, neuronové sítě přitahují čím dál více pozornosti, avšak nové náročnější problémy vyžadují také vývoj učících technik, aby neuronové sítě byly schopny produkovat vyhovující výsledky dostatečně rychle.

Jedním takovým řešením se ukazují být evoluční techniky. Evoluční techniky jsou heuristické optimalizační algoritmy inspirované mechanismy biologické evoluce, jako například reprodukce, mutace, selekce. V evolučních technikách je generována a iterativně aktualizována sada kandidátských řešení, která se postupem času vyvíjí v naději, že vzniknou lepší řešení. Různé druhy evolučních technik (genetické algoritmy, genetické programování, evoluční strategie, diferenciální evoluce) začínají být běžnou technikou pro řešení obtížných problémů reálného světa. Používají se pro složité optimalizační problémy založené na vyhledávání, které je obtížné a časově náročné pro jiné algoritmy. Jejich hlavní výhodou je totiž rychlost nalezení řešení, robustnost a dobrá schopnost globálního vyhledávání.

Evoluční techniky lze uplatnit při trénování umělých neuronových sítí. Bylo již navrženo několik postupů, jak toho dosáhnout. Nejčastěji jsou používány genetické algoritmy pro učení umělých neuronových sítí, avšak výzkumy z posledních let naznačují, že využití i ostatních druhů evolučních technik pro nejen učení neuronových sítí, ale i hledání jejich ideální topologie se jeví jako lepší varianta oproti klasicky používaným algoritmům. Proto zájem o jejich spojení a použití roste. Nicméně je zde stále velký prostor pro další prozkoumání a nové poznatky.

Cílem této práce je seznámení s teoretickým základem genetických algoritmů a dalších metod evolučních technik a neuronových sítí. Dále pak představit existující aplikace evolučních technik na učení neuronových sítí a popsat existující knihovny ve vybraných programovacích jazycích a demonstrovat jejich využití.

2 Evoluční techniky

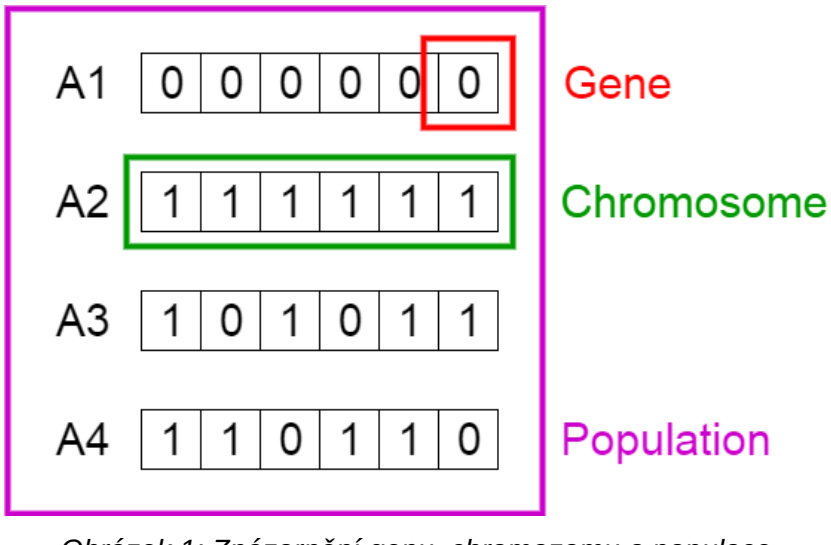
2.1 Genetické algoritmy

Genetické algoritmy jsou optimalizační stochastické algoritmy založeny na principech genetiky a přirozeného výběru. Myšlenka použít evoluci jako optimalizační nástroj se objevila již během 50. let a 60. let 20. století, kdy několik počítačových vědců nezávisle na sobě studovalo evoluční systémy. Jejich společným cílem bylo použití mutace a selekce – základních konceptů neodarwinistické evoluční teorie – k vyvinutí populace řešení daného problému.

Termín genetické algoritmy byl poprvé použit Johnem Hollandem [1] v roce 1975. Holland se inspiroval adaptací, která se vyskytuje v přírodě, a snažil se vyvinout způsoby, jak tuto přirozenou adaptaci přenést do počítačových systémů. Na rozdíl od předchozích evolučních algoritmů, které používaly převážně jen mutaci a selekci [2], Holland přidal další složku – křížení.

2.1.1 Základní principy

Genetické algoritmy prohledávají stavový prostor, jenž je tvořen potenciálními řešeními. Na počátku je inicializována skupina jedinců, kteří představují možné řešení problému a nazývají se populace. Řešení z jedné populace se vezmou a použijí k vytvoření nové populace. O populacích v takovýchto jednotlivých krocích mluvíme jako o generacích. Každý jedinec v jednotlivé generaci je charakterizován sadou proměnných označených jako geny. Geny jsou spojeny do řetězce a utvářejí chromozom. Zobrazení populace se čtyřmi chromozomy, kde každý chromozom obsahuje šest genů, je možno vidět na obrázku 1.



Obrázek 1: Znázornění genu, chromozomu a populace

Genetické algoritmy mohou být binární nebo spojité, to záleží na řešeném problému. V binárních GA jsou jednotlivé geny reprezentovány pomocí bitového pole (0 a 1), ve spojitých GA jsou reprezentovány pomocí řetězců (string).

Jedinci v každé generaci jsou ohodnoceny funkcí (fitness funkce), která každému jedinci přiřazuje určitou hodnotu. Fitness funkce vyjadřuje schopnost jedince konkurovat ostatním (jeho kvalitu). To znamená, že čím vyšší bude hodnota fitness funkce jedince (jedinec představuje kvalitnější řešení), tím vyšší bude pravděpodobnost, že onen jedinec bude vybrán pro reprodukci. Vhodná individua jsou vybrána pomocí selekce. Poté jsou aplikovány rekombinační operátory – křížení a mutace. Při křížení jsou vybráni dva jedinci (rodiče), jejichž geny se zkombinují a vzniknou dva potomci. Při mutaci vzniká jeden potomek a to tak, že jeden či více genů je změněn. Z nově vzniklých potomků, jenž byli ohodnoceni fitness funkcí, a jedinců z původní generace jsou vybráni ti, kteří mají nejvyšší hodnotu fitness a vytvoří tak novou generaci. Proces ohodnocení, selekce, křížení a mutace se opakuje dokud není splněna podmínka pro ukončení. Tou může být buď nalezení optimálního řešení, nebo dosažení předem daného počtu iterací. V takovém případě se pak vybere doposud nejlepší výsledek.

2.1.2 Selekce

Selekce slouží k výběru jedinců z populace, kteří budou produkovat nové potomky. Cílem selekce je vybírat lepší jedince, aby mohli předat své geny do další generace. Avšak by měla by být zachována rozmanitost populace. Pokud by selekce významně upřednostňovala příliš silné jedince, dílčí řešení by se sobě brzy začaly podobat a došlo by k předčasné konvergenci, což je nežádoucí. Naopak příliš slabý výběr by znamenal velmi pomalou evoluci.

2.1.2.1 Ruletové kolo

Selekce ruletového kola je stochastická metoda, kde jsou jedinci vybíráni podle jejich hodnoty fitness funkce. Jedná se tedy o fitness proporcionální selekci (FPS). Platí, že čím vyšší je hodnota fitness jedince, tím je pravděpodobnější jeho vybrání pro reprodukci. Tuto metodu si lze představit jako ruletové kolo, které je tvořeno celou populací, a každému jednotlivci je přiřazena díl úměrný jeho fitness hodnotě, takže zdatnější jedinec zabírá větší část kola. Na obvodu kola je zvolen pevný bod a kolo je N-krát roztočeno, kde N odpovídá počtu jedinců v populaci. Při každém roztočení je vybrán jedinec, jehož oblast se nachází u pevného bodu.

2.1.2.2 Stochastické univerzální vzorkování

Stochastické univerzální vzorkování (SUS) patří mezi metody fitness proporcionální selekce. Je velmi podobné ruletovému kolu. Jedincům je opět přiřazena část kola odpovídající jejich fitness hodnotě, avšak namísto zvolení jednoho bodu a otáčení kola N-krát, je zvoleno více bodů rovnoměrné rozmístěných. Tím pádem jsou vybráni všichni jedinci během jednoho roztočení kola.

2.1.2.3 Rank (pořadová) selekce

Rank selekce dokáže pracovat s negativními hodnotami fitness. Jedinci v populaci jsou seřazeni podle jejich fitness hodnoty, kdy nejhorší má pořadí 1, druhý nejhorší pořadí 2 až nejlepší má pořadí N (počet jedinců v populaci). Avšak dále výběr rodiče nezáleží na fitness, pouze na jeho

pořadí. Na rozdíl od metody ruletového kola ji lze snadno aplikovat i v případě, že provádíme minimalizaci problému, tj. kvalitnější jedinci mají menší hodnotu fitness funkce. Pravděpodobnost selekce je lineárně přiřazena jedincům podle jejich pořadí. Tímto způsobem mají všechny chromozomy šanci být vybrány, což brání příliš rychlé konvergenci. V některých případech (např. když jedinci mají velmi podobné hodnoty fitness) tato selekce, kdy je zachována větší rozmanitost, může vést k lepšímu nalezení kvalitnějších řešení než u FPS [3]. Pokud by pravděpodobnost selekce seřazených jedinců měla exponenciální váhu, jednalo by se o exponenciální rank selekci.

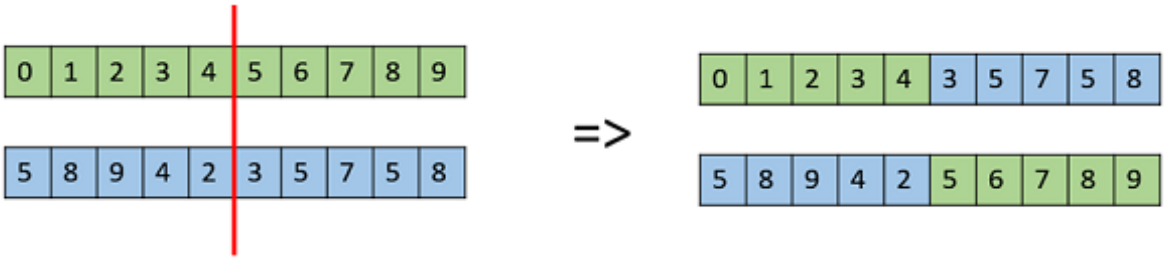
2.1.2.4 Turnajová selekce

Turnajová selekce stejně jako rank selekce umí pracovat se zápornými hodnotami fitness, avšak na rozdíl od rank selekce nevyžaduje seřazení jedinců, což ji dělá výpočetně efektivnější. Lze ji také snadno použít pro minimalizaci problému. Turnajová selekce probíhá tak, že je náhodně vybráno několik jedinců (obvykle dva nebo tři) a z nich je vybrán ten nejlepší (ten s nejvyšší/nejmenší hodnotou fitness dle typu problému). Mezi vybranými jedinci tedy proběhne turnaj, jehož vítěz se stane rodičem. Chromozomy jsou pak vráceny do původní populace a lze je znova vybrat. Rodiče vybraní turnajovou selekcí mají vyšší průměrné fitness než průměrné fitness populace. Tento rozdíl ve fitness poskytuje selekční tlak, který vede GA ke zlepšení fitness každé následující generace [4]. Výhodou turnajové selekce je snadná implementace nezávisle na tom, zda je problém minimalizační nebo maximalizační.

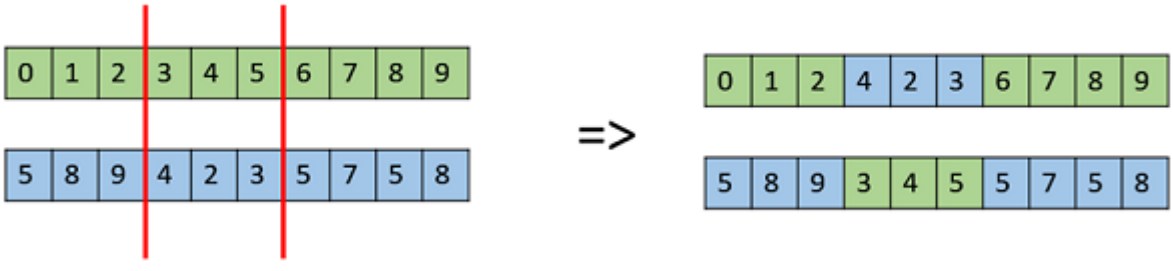
2.1.3 Křížení

Operátor křížení se používá k produkci nových jedinců, kteří mohou vést k lepším výsledkům. Pokud nově vzniklí potomci budou představovat lepší řešení, nahradí méně vhodné jedince z předchozí generace.

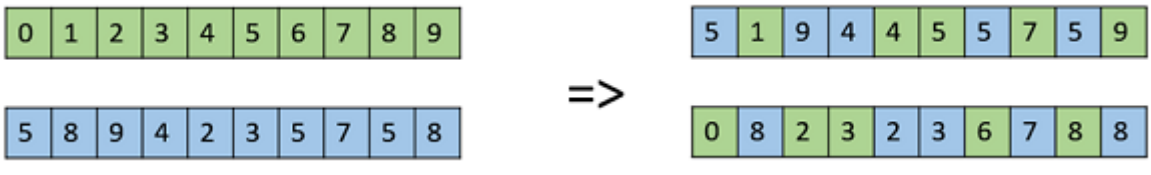
Dva jedinci (rodiče), kteří byli vybráni v kroku selekce, jsou spojeni dohromady, aby vytvořili dva nové potomky obsahující geny od obou rodičů. Křížení je v genetických algoritmech realizováno s poměrně velkou pravděpodobností P_C (cca 0,8). Existuje několik možností, jak provést křížení, z nichž nejběžnější je jednobodové křížení. Při jednobodovém křížení je náhodně vybrán jeden bod křížení a geny rodičů jsou prohozeny. Jeden potomek zdědí od prvního rodiče geny nacházející se před bodem křížení a od druhého rodiče geny nalézající se za bodem křížení. Druhý potomek zdědí geny v opačném pořadí, od prvního rodiče dostane geny za bodem křížení a od druhého rodiče geny před bodem křížení. Jednobodové křížení je znázorněno na obrázku 2.



Obrázek 2: Jednobodové křížení



Obrázek 3: Dvoubodové křížení



Obrázek 4: Uniformní křížení

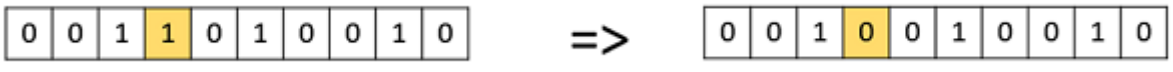
Zobecněním jednobodového křížení je vícebodové křížení, kde je zvoleno více bodů křížení a jednotlivé geny jsou prohozeny mezi střídajícími se úseky. Avšak často se nevolí více než dva body křížení, jedná se pak o dvoubodové křížení, které lze vidět na obrázku 3. Jiný způsob křížení je například uniformní křížení, kdy není chromozom rozdělen na úseky, ale jednotlivé geny od prvního rodiče jsou vybrány náhodně a zbytek je doplněn odpovídajícími geny druhého rodiče. Pro každý gen v chromozomu je tedy samostatně rozhodnuto, zda se bude nacházet v prvním potomku nebo ne. Druhý potomek je poté tvořen opačnými odpovídajícími si geny. Tento druh křížení ukazuje obrázek 4. V některých genetických algoritmech může při křížení vzniknout neplatný jedinec. V některých případech je možné použít speciální operátory křížení, které produkují potomky tak, aby nedošlo k porušení omezení problému. Takovými metodami křížení může být například pořadové křížení (ordered crossover) nebo partially mapped crossover.

2.1.4 Mutace

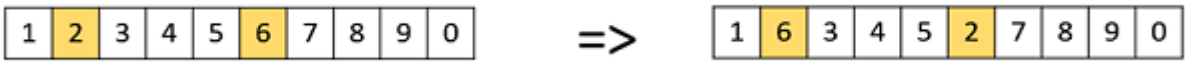
Mutace pomáhá zachovat rozmanitost v populaci a tím pádem zabraňuje předčasné konvergenci. Bez mutace by se jednotlivé chromozomy v populaci brzy začaly sobě podobat, tudíž by se mohlo

stát, že se algoritmus zasekne v lokálním optimu, aniž by byl prohledán celý stavový prostor a nalezeno optimální řešení.

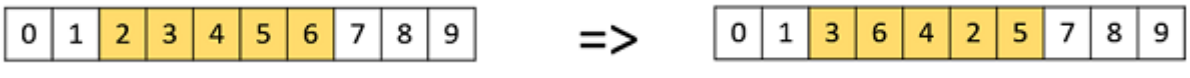
Jednotlivci v chromozomu jsou mutováni s pravděpodobností mutace genu P_M (zpravidla <0,5). Tato pravděpodobnost bývá obvykle malá, tudíž k mutaci dochází méně než ke křížení. Nejjednodušší způsob mutace je náhodná změna jednoho nebo více genů jednoho jedince. V případě bitové reprezentace se jedná o prohození 0 a 1 ve zvoleném genu (bit flip mutace), při jiné reprezentaci je hodnota genu zaměněna za náhodnou hodnotu z daného rozsahu. Jednobodovou bitovou mutaci lze vidět na obrázku 5.



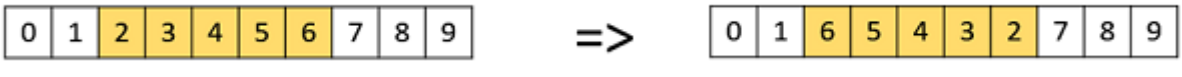
Obrázek 5: Jednobodová mutace



Obrázek 6: Swap mutace



Obrázek 7: Scramble mutace



Obrázek 8: Inverzní mutace

Další možné způsoby mutace jsou například swap mutace, scramble mutace a inverzní mutace. Při swap mutaci, kterou ukazuje obrázek 6, dojde k prohození (swap) hodnot dvou náhodně vybraných genů v chromozomu. U scramble mutace je vybrána podmnožina genů, jenž mohou, ale nemusí spolu sousedit, a jejich hodnoty jsou náhodně zamíchány. Scramble mutace, kde je vybrána podmnožina za sebou jdoucích genů, je znázorněna na obrázku 7. V inverzní mutaci opět vybereme podmnožinu genů, avšak v tomto případě geny musí spolu sousedit, a převrátíme jejich pořadí. Možný způsob řešení inverzní mutace zobrazuje obrázek 8.

2.2 Genetické programování

Genetické programování je metoda strojového učení, která využívá darwinovský princip přirozeného výběru pro automatické řešení problému bez manuálního zásahu. GP je velmi

užitečné při řešení problémů, kde není předem známá podoba výsledku a přibližné řešení je přijatelné.

Podobně jako v genetických algoritmech i v genetickém programování je na počátku inicializována sada náhodných jedinců, kteří se postupně vyvíjí v průběhu generací. Avšak hlavní rozdíl mezi GA a GP je v reprezentaci jedinců, kdy v GA se jedná o konkrétní řešení daného problému a v GP jde zpravidla o počítačový program. V některých případech se může jednat třeba jen o výraz či funkci. Programy bývají nejčastěji reprezentovány jako stromové struktury. Uzly stromu mohou být buď funkce nebo terminály. Funkcemi mohou být standardní aritmetické operace (sčítání, odčítání, násobení, dělení), matematické operace (sin, cos, tg, apod.), logické operace (and, or, not, apod.), podmíněné příkazy (if-else) a jiné. Terminály mohou být proměnné nezávislé na problému, konstanty nebo funkce bez argumentů. Uzly s funkcemi naznačují instrukce, které mají být provedeny, a terminály označují argumenty pro každou instrukci. Další možnou reprezentací programů v GP může být lineární genetické programování, kde jsou jedinci reprezentování jako posloupnost instrukcí. Tato reprezentace více vyhovuje imperativním programovacím jazykům.

Programy jsou v každé generaci ohodnoceni fitness funkcí a nejlepší jedinci jsou vybráni k produkování potomků pomocí genetických operátorů. Těmito operátory mohou být reprodukce, která kopíruje vybrané jedince do nové populace; křížení, kde nové programy jsou vytvořeny rekombinací náhodně vybraných částí od dvou programů; mutace, kde je nový program vytvořen náhodnou změnou vybrané části tohoto programu. Algoritmy GP jsou obvykle ukončeny, když je dosažena předem definovaná hodnota fitness nebo je dosažen maximální počet generací [5].

2.3 Evoluční strategie

Evoluční strategie jsou stochastické optimalizační metody založené na principu přirozeného výběru. ES začínají inicializováním populace jedinců. Jedinec v ES kódován jako vektor obsahující nejen soubor parametrů popisujících jedince a hodnotu fitness, ale obvykle také soubor endogenních (vyvíjejících se) strategických parametrů. Parametry endogenní strategie jsou zvláštností ES. Používají se ke kontrole určitých statistických vlastností genetických operátorů, zejména mutace. Mohou se vyvíjet během evolučního procesu a jsou potřebné v autoadaptaci [6].

Po inicializaci populace je určena hodnota fitness každého jedince a následuje cyklus, dokud není splněno kritérium ukončení. Běžnými kritérii ukončení bývá dosažení určité hodnoty fitness nebo překročení maximálního počtu iterací. V cyklu probíhají následující kroky: selekce, rekombinace, mutace a vyhodnocení fitness. Selekce v ES funguje tak, že šanci na reprodukci dostanou pouze jedinci se slibnými vlastnostmi (vysokou hodnotou fitness). Existují dvě běžné metody rekombinace v ES – intermediate rekombinace a dominantní rekombinace. Intermediate rekombinace tvoří potomky zprůměrováním dílčích komponent rodičů. Dominantní rekombinace nastavuje i-tou pozici vektoru potomka náhodně na jednu z hodnot rodičů. Hlavním zdrojem genetických změn v evolučních strategiích je operátor mutace. Mutace náhodně modifikuje každou složku. Rozsah této modifikace je založen na strategických proměnných [7].

2.3.1 Autoadaptace

Autoadaptace (self-adaption) je charakteristickým rysem evolučních strategií. Základní myšlenka autoadaptace spočívá v pozměnění endogenních strategických parametrů. Tyto parametry mohou podléhat rekombinaci a vždy podléhají mutaci. Takto pozměněné parametry se pak používají k ovládání mutačního operátoru aplikovaného na jedince. Díky zahrnutí strategických parametrů ve vektoru jedince jsou vybrány a zděděny společně s jedincem. Tudíž mají větší šanci na přežití, pokud pozmění jedince tak, aby by měl větší hodnotu fitness [6].

2.4 Diferenciální evoluce

Diferenciální evoluce patří mezi evoluční techniky a používá se pro optimalizaci nelineárních a nediferencovatelných spojitých prostorových funkcí. Podobně jako jiné evoluční techniky, algoritmus DE začíná inicializací počáteční populace kandidátských řešení. Jednotlivci jsou nejčastěji reprezentováni pomocí vektoru reálných čísel. Takto vytvoření jedinci jsou ohodnoceny účelovou funkcí. Poté probíhá iterační fáze, ve které jsou vytvářeny nové generace. Jedinci nové generace vzniknou tak, že k náhodně vybraným vektorům aktuální generace jsou vytvořeny pokusné vektory a lepší z dvojice původní vektor a z něho vzniklý pokusný vektor je vybrán do další generace. Pokusné vektory vznikají aplikací operátorů mutace a křížení. DE používá proces nazývaný diferenciální mutace, který funguje tak, že vážený rozdíl dvou vektorů původní generace přidá ke třetímu vektoru. Po mutaci probíhá křížení, kde jsou parametry mutovaného vektoru smíchány s jiným, předem vybraným vektorem a vznikne pokusný vektor. Ten je následně ohodnocen účelovou funkcí. Proces mutace, křížení a selekce se opakuje dokud není splněna ukončovací podmínka. Tou může být např. nalezení optima nebo předem definovaný počet generací.

3 Aplikace evolučních technik na učení neuronových sítí

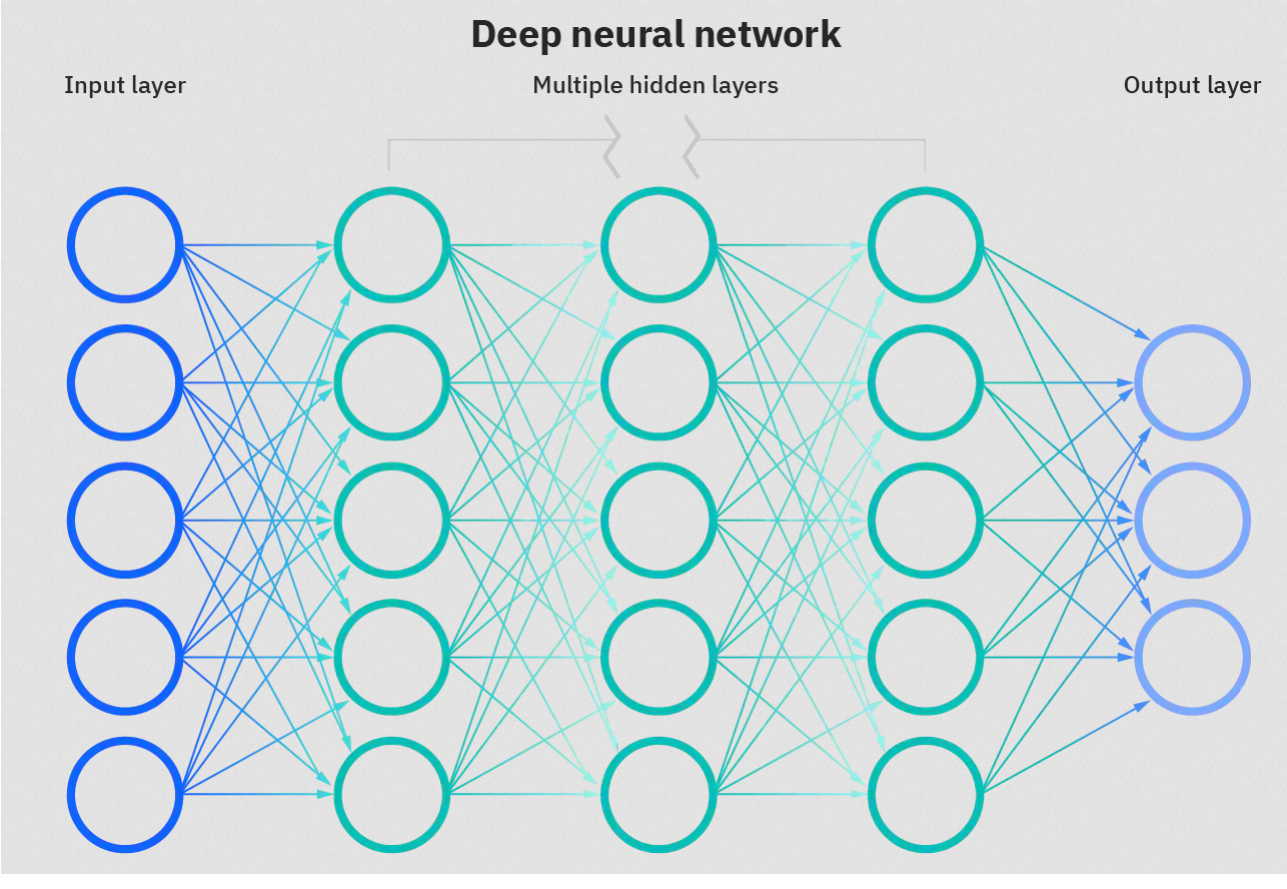
3.1 Neuronové sítě

Neuronové sítě jsou druhem strojového učení. Jsou inspirovány chováním lidského mozku a napodobují způsob, jakým fungují biologické neurony. NS umožňují počítačovým programům například rozpoznat vzorce a řešit problémy v oblasti umělé inteligence. Jsou vhodné pro nelineární data s velkým počtem vstupů, jako jsou obrázky. Mohou pracovat s libovolným počtem vstupů a řešit úlohy paralelně. NS pomáhají seskupovat neoznačená data podle jejich podobností a klasifikovat data, pokud mají množinu, na které mohou trénovat.

Algoritmus neuronových sítí se učí zpracováváním mnoha příkladů (trénovací data), kdy je dáno nějaké kritérium, jenž určuje, který výstup je správný. Jedná se pak o tzv. učení s učitelem. U tohoto druhu učení se při trénování modelu přesnost ohodnocuje chybovou funkcí (cost function). Cílem je minimalizovat tuto funkci, což znamená, že výstup je klasifikován správně. Při

každém tréninkovém příkladu jsou parametry modelu upraveny tak, aby se postupně přibližovaly k minimu. Po provedení dostatečného množství příkladů, může NS zpracovávat nové vstupy a úspěšně vracet přesné výstupy. Čím větší je trénovací sada, tím přesnější výsledky bude NS vracet. Při učení, kdy neexistuje vnější kritérium určující správnost výsledku, tzv. učení bez učitele, algoritmus pracuje pouze s informacemi získanými během učícího procesu. Jedná se tudíž o samoorganizaci [8].

Neuronové sítě se skládají z vrstev uzlů, které obsahují vstupní vrstvu (input layer), jednu nebo více skrytých vrstev (hidden layer) a výstupní vrstvu (output layer). Neuronová síť složená z mnoha vrstev se nazývá hluboké učení (deep learning), kde hloubka poukazuje na velký počet vrstev. NS s pouze dvěma nebo třemi vrstvami je jen základní neuronová síť. Samotné výpočty probíhají v uzlech (nodes), neboli také umělých neuronech, které jsou vzájemně propojeny. Počet uzlů pro každou vrstvu nemusí být stejný. Je možno ho definovat podle potřeby a konkrétní řešené úlohy. Hluboká neuronová síť (hluboké učení) je zobrazena na obrázku 9.



Obrázek 9: Schéma hluboké neuronové sítě

Každý neuron má přidruženou synaptickou váhu a prahovou hodnotu. Synaptické váhy pomáhají určit důležitost jakékoli dané proměnné, přičemž větší váhy jsou přiřazeny vstupům s větším významem s ohledem na konkrétní úlohu, kterou se algoritmus snaží naučit. Tyto váhy jsou přiřazeny, jakmile je určena vstupní vrstva. Všechny vstupy jsou poté vynásobeny jejich

příslušnými váhami a sečteny. Součet výstupů potom prochází přenosovou funkcí. Data jsou předána další vrstvě (dochází k aktivaci uzlu), pokud překročí danou prahovou hodnotu. Ta určuje, jak velká musí být suma vstupů, aby byl uzel aktivován [9]. Výstup z jednoho uzlu se tedy stane vstupem dalšího uzlu. Formálně:

$$Y = S\left(\sum_{i=1}^{N} \left(w_i x_i\right) - \Theta\right) \quad ,$$

kde x_i jsou vstupy neuronu, w_i jsou synaptické váhy, Θ je práh, S(x) ke přenosová funkce, Y je výstup neuronu.

3.1.1 Druhy neuronových sítí

Neuronová síť, kde jsou data předávána z jedné vrstvy do další, se nazývá dopředná síť (feedforward network). Nejjednodušším a nejstarším modelem dopředné sítě je perceptron, který se skládá pouze z jednoho neuronu, a byl představen Frankem Rosenblattem v roce 1958. Dopředná síť složená z mnoha vrstev se nazývá vícevrstvý perceptron (multilayer perceptron). Tato síť používá nelineární přenosovou funkci. K trénování využívá algoritmus zpětného šíření chyby (backpropagation), který umožňuje vypočítat a přiřadit chybu spojenou s každým neuronem, čehož využívá k vhodnému přizpůsobení parametrů modelu. Pro rozpoznávání a zpracování obrázků se používá konvoluční neuronová síť (CNN). Jedná se o dopřednou vícevrstvou síť.
Dalším druhem NS jsou rekurentní neboli zpětnovazební sítě (RNN), kde signály mohou procházet v obou směrech a vytvářet tak smyčky. Cykly zpětné vazby mohou způsobit změnu chování sítě v průběhu času [10]. Příkladem takové sítě může být Long short-term memory (LSTM). Rekurentní sítě jsou výkonné, ale bývají velmi komplikované.

3.2 Učení neuronových sítí pomocí evolučních technik

K umělým neuronovým sítím je v poslední době ubírána čím dál větší pozornost kvůli jejich aplikaci v různých odvětvích, jako například data mining, klasifikace obrázků, rozpoznání řeči a emocí apod. Jsou často schopny poskytovat správné výstupy, přestože jsou trénovány na relativně řídké trénovací sadě. Tradičně se k učení NS používá algoritmus zpětného šíření chyby (backpropagation – BP), který mnohdy dokáže najít dobré hodnoty synaptických vah a prahu, avšak má několik nevýhod. Za prvé, BP může snadno uvíznout v lokálním minimu, tudíž nedojde k prohledání celého prostoru. Dále BP vyžaduje, aby přenosová funkce byla diferencovatelná a kritérium chyby bylo spojité. Navíc má pomalou rychlost konvergence. K překonání těchto problémů bylo navrhnuto několik různých technik, mezi něž patří i neuroevoluce. Ta se zabývá trénováním umělých neuronových sítí pomocí evolučních algoritmů. Evoluční algoritmy mohou být použity k vývoji synaptických vah nebo i celé topologie sítě včetně jejích vah.
Velmi často jsou používány k trénování NS genetické algoritmy. V [11] je navrhnut genetický algoritmus pro trénování dopředné neuronové sítě. Synaptické váhy a prahy jsou zde zakódované

jako seznam reálných čísel. Tento seznam představuje chromozom. Pro GA zde bylo vytvořeno mnoho různých genetických operátorů, z nichž na základě několika experimentů byly vybrány ty nejlepší a ty poté byly použity v závěrečném experimentu. Závěrečný experiment porovnával výkonost klasického algoritmu zpětného šíření chyby s navrženým genetickým algoritmem. Výsledek tohoto experimentu ukázal, že navržený GA překonal BP. Tento algoritmus též dokáže řešit problémy s nespojitou přenosovou funkcí a nespojitým kritériem chyby.

Pokus o optimalizaci synaptických vah pomocí genetického algoritmu řeší také [12]. Je zde představena metoda kombinující GA a BP (GABP) k trénování neuronových sítí, a poté je porovnána s klasickým BP a metodou, která používá k učení synaptických vah pouze GA. Výsledky ukazují, že BP je relativně rychlý, pokud neuvázne v lokálním minimu. Avšak navržený algoritmus GABP se v lokálním minimu nezasekává, navíc má lepší schopnost generalizace než zbylé dvě metody. Další provedený experiment testuje výkonost metod na různých datových sadách. Z něho vyplývá, že GABP je efektivní jen pro některé druhy dat a ne vždy dává platné výsledky.

Použití GA je praktické v mnoha oblastech, ale má problém s kódováním, jelikož se snaží řešit spojité problémy diskrétní metodou. Toto může být vyřešeno použitím diferenciální evoluce, která používá vektory. Aplikace DE v dopředné neuronové síti pro zlepšení procesu učení, rychlosti konvergence a přesnosti klasifikace je prezentována ve [13]. Diferenciální evoluce je zde použita pro optimalizaci synaptických vah. Využití DE pro učení neuronové sítě (DENN) je zde porovnáno s PSO a GA, které byly aplikovány na BP. DENN zde dosahuje rychlejší konvergence než zbylé dvě metody.

Ve [14] je navrhnuta nová samoadaptivní verze DE pro optimalizaci vah neuronových sítí s přihlédnutím ke struktuře sítě. Poté je porovnána s různými druhy diferenciální evoluce. Dále je zde navržen a použit nový operátor křížení. Prezentované výsledky ukazují, že navržená metoda je ve většině případech lepší než BP.

V [15] je použit algoritmus DEGL (DE with global and local neighborhood based mutation) ke hledání hodnot synaptických váh neuronové sítě a k minimalizaci chyb při učení. DEGL je vhodný pro trénování NS, jelikož poskytuje dobrou rovnováhu mezi místním a globálním vyhledáváním. Porovnání DE a DEGL ukazuje, že obě metody mají dobrou efektivitu při trénování i testování výkonu NS, avšak DEGL je o něco účinější při učení NS.

Využití evolučních algoritmů pro trénování neuronových sítí se ukazuje jako lepší varianta oproti klasickému algoritmu zpětného šíření chyby, jelikož nedochází k uváznutí v lokálním extrému. V mnoha případech dosahuje neuroevoluce srovnatelných, nebo dokonce i lepších výsledků než BP.

4 Knihovny implementující neuronové sítě a evoluční techniky

V této sekci jsou popsány vybrané knihovny implementující evoluční techniky a neuronové sítě v různých programovacích jazycích. U knihoven zabývajících se ET je popis zaměřen především na reprezentaci jedinců v chromozomu a fitness funkci. Blíže je zde popsána implementace různých metod selekce a genetických operátorů, pokud se liší od výše popsaných postupů. Popis knihoven neuronových sítí přibližuje jejich strukturu a fungování.

4.1 Evoluční techniky

4.1.1 OpenGA

OpenGA je knihovna poskytující flexibilní rámec v jazyce C++. Je vhodná pro optimalizační problémy s výpočetně náročným ohodnocením řešení. Způsob reprezentace genů nemusí být vektor, nýbrž je zcela na programátorovi. Lze tedy definovat libovolnou strukturu nebo třídu představující řešení. Avšak díky tomu musí programátor definovat i genetické operátory křížení a mutaci. Stejně tak způsob reprezentace výsledků u každé generace (zda se mají například zobrazit na obrazovku nebo uložit do souboru) musí určit uživatel.

Tato knihovna odděluje ohodnocování od konečné fitness hodnoty. Náročný proces ohodnocování řešení probíhá v hodnotící funkci (evaluation function) a při tomto procesu jsou získány veškeré potřebné informace. Výsledek této funkce se nazývá střední náklady (middle costs). Ty jsou dále předány fitness funkci. Střední náklady mohou ukládat podrobnější údaje, než jaké jsou potřeba pro výpočet konečného fitness. Přehodnocení některých informací může být výpočetně náročné a je tedy zbytečné je počítat znovu, když již byly získány hodnotící funkcí. Hodnotící funkce má také za úkol kontrolu omezení. Pokud některý jedinec poruší jakékoli omezení, tato funkce je schopna ho nahradit jiným. Architektura knihovny OpenGA umožňuje programátorovi posunout kontrolu omezení po nebo uprostřed ohodnocování [16].

4.1.2 GALGO

Galgo je C++ knihovna navržená k řešení problémů s omezeními maximalizací nebo minimalizací účelové funkce v rámci daných mezí. Knihovna Galgo obsahuje některé z nejpoužívanějších metod pro selekci, křížení a mutaci, ale umožňuje i snadné přidávání jiných metod. Programátor si tedy může vybrat z již zahrnutých metod nebo přidat vlastní.

Chromozomy jsou reprezentovány jako binární řetězec obsahující zakódované parametry. Počet bitů potřebných pro zakódování každého z nich si může uživatel zvolit z intervalu [1;64]. Při inicializování populace je vygenerován náhodný 64 bitový unsigned integer pro každý parametr v intervalu [0, MAXVAL], kde MAXVAL je největší unsigned integer pro zvolený počet bitů. Toto náhodně vygenerované číslo bude poté převedeno do binárního řetězce. Binární řetězec bude

zkrácen na požadovaný počet bitů a přidán do chromozomu. Po provedení selekce, křížení a mutace je řetězec převeden zpět na unsigned integer a je uplatněna rovnice s omezeními parametru. Tato metoda umožňuje dosáhnout rychlejší konvergence, protože jak při inicializaci populace, tak při křížení a mutaci jsou generovány pouze hodnoty uvnitř mezí [17].

Metody selekce implementované v této knihovně jsou ruletové kolo, stochastické univerzální vzorkování, turnajová selekce, rank selekce, rank selekce se selektivním tlakem, transformační rank selekce. K dispozici jsou tři metody křížení, a to jednobodové, dvoubodové a uniformní, a tři metody mutace – jednobodová, uniformní a mezní (boundary mutation). Při jednobodovém, resp. dvoubodovém, křížení jsou náhodně vybráni dva rodiče a náhodně je vybrán jeden bod, resp. jsou vybrány dva body, křížení. Poté jsou příslušné geny prohozeny. U uniformního křížení jsou také náhodně vybráni rodiče. Následně probíhá cyklus, kde je vždy namátkou vybrán jeden rodič a jeho gen je přidán do chromozomu potomka. U jednobodové mutace je náhodně převrácena hodnota jednoho bitu. V uniformní mutaci jsou geny chromozomu procházeny v cyklu a náhodně je jeden gen nahrazen jiným. Poslední druh implementované mutace je mezní mutace, při které je gen v chromozomu nahrazen buď svou horní, nebo dolní mezí.

4.1.3 GAlib

Galib je C++ knihovna určená pro optimalizaci pomocí genetických algoritmů. Zahrnuje několik metod pro selekci, křížení a mutaci, umožňuje jejich přizpůsobení i vkládání vlastních tříd a funkcí. Tato knihovna obsahuje čtyři způsoby, jak reprezentovat chromozomy, a to pomocí bitového pole, pole reálných čísel, seznamu a stromu. Funkci pro ohodnocování fitness musí poskytnout uživatel.

Lze si vybrat ze šesti vestavěných metod selekce, konkrétně rank selekce, ruletové kolo, SUS, turnajová selekce, deterministické vzorkování (DS) a stochastické zbytkové vzorkování (SRS). V DS je nejprve vypočítáno očekávané zastoupení každého jednotlivce a dočasná populace je naplněna jedinci s nejvyššími očekávanými počty. Jakékoli zbývající pozice jsou zaplněny tak, že se seřadí původní jedinci podle desetinné části jejich očekávaného zastoupení a nejvýše položení jedinci jsou vybráni. Nakonec probíhá náhodný výběr jednotlivců z dočasné populace. V SRS je stejně jako v DS zpočátku vypočítáno očekávané zastoupení každého jednotlivce a dočasná populace je naplněna jedinci s nejvyššími očekávanými počty. Jakékoli zlomkové očekávané reprezentace jsou použity k tomu, aby jednotlivec získal větší pravděpodobnost vyplnění prostoru. Poté jsou z dočasné populace náhodně vybráni jedinci [18].

Vestavěné metody pro křížení jsou jednobodové, dvoubodové, uniformní, sudé, liché, cyklické, ordered (pořadové), partial match a jednobodové křížení stromu. Jednobodové, dvoubodové a uniformní křížení je implementováno tak, jak je popsáno v sekci 2.1.3. Sudé a liché křížení bere střídavě geny od prvního a druhého rodiče. V sudém křížení jsou brány sudé geny od prvního rodiče a liché geny od druhého rodiče. Liché křížení probíhá opačně, tedy od prvního rodiče jsou brány liché geny a od druhého rodiče sudé geny (nultá pozice se bere jako sudá). Partial match

křížení vezme podmnožinu genů od jednoho rodiče a vloží ji do druhého rodiče. Snaží se zachovat původní pořadí genů, ale pokud jsou některé geny v novém chromozomu duplikátní, jsou nahrazeny genem prvního rodiče, který se v daném chromozomu ještě nenachází. Pořadové křížení vybere podmnožinu po sobě jdoucích genů od jednoho rodiče a zkopíruje ji do potomka. Zbytek genů potomek dostane od druhého rodiče tak, že jsou ve druhém rodiči vyškrtnuty geny již obsažené v potomkovi a zbylé geny jsou popořadě vloženy do potomka. Cyklické křížení vytváří potomka tak, že každá pozice je obsazena odpovídajícím prvkem od jednoho z rodičů. První gen je vždy brán od prvního rodiče, potom je každý následující gen vybrán od jednoho z rodičů tak, aby měl stejnou pozici jako rodič a byl vytvořen cyklus. Partial match, pořadové a cyklické křížení je použito pouze pro chromozomy reprezentované polem a seznamem. Jednobodové křížení u stromu vezme uzel s celým jeho podstromem od jednoho rodiče a nahradí ho uzlem od druhého rodiče (opět s celým jeho podstromem). Tento druh křížení lze využít pouze u jedinců definovaných jako stromová struktura.

Obsažené mutační operátory v GAlib jsou random flip, swap mutace, destruktivní mutace, swap node a swap subtree. Random flip mutace mění geny podle toho, jak velká je pravděpodobnost mutace. Pokud je pravděpodobnost malá, je tato metoda zavolána pro každý bit v řetězci a zda bude gen mutován je určeno náhodně pomocí hodu mincí. Jinak je mutován potřebný počet bitů určený mutačním poměrem. Swap mutace náhodně prohodí geny. Tato mutace není k dispozici pro chromozomy reprezentované jako bity. Destruktivní mutace jde použít jen u seznamu a u stromu. Funguje tak, že odstraní vybrané uzly. Každý uzel má pravděpodobnost odstranění rovnou pravděpodobnosti mutace. Swap node mutace prohodí konkrétní uzly a ponechá jejich podstromy nedotčené. Tím pádem se nemění hloubka stromu. Swap subtree prohodí uzly i s podstromy, ve kterých jsou obsaženy. Tato metoda funguje pouze u uzlů, kteří spolu přímo nesouvisí.

4.1.4 GeneAI

GeneAI je knihovna v jazyce Python pro genetické algoritmy. Jsou zde definovány dvě třídy, a to pro binární nebo spojité problémy. Tyto třídy vyžadují dva argumenty, konkrétně počet genů v každém chromozomu a fitness funkci. Fitness funkce není zde implementovaná, proto uživatel musí použít vlastní. Knihovna je nastavena tak, že všechny problémy musí být maximalizační, tudíž je potřeba upravit fitness funkci vhodným způsobem. Ve spojité třídě je také možno definovat, zda je problém typu int nebo float a určit jejich minimální a maximální hodnoty.

K selekci lze využít čtyři metody: ruletové kolo, turnajová selekce, random selekce a two by two. V random selekci jsou jedinci vybíráni náhodně se stejnou pravděpodobností. Two by two selekce seskupuje jedince po dvou od shora dolů. Knihovna implementuje jednu metodu pro křížení a jednu metodu pro mutaci v obou třídách. V binární třídě jde o klasické jednobodové křížení, kde potomci vznikají výměnou genů dvou rodičů mezi náhodně zvoleným bodem křížení, a bit flip mutaci, kde je náhodně prohozen gen v chromozomu. Ve spojité třídě je implementovaná

jednobodová mutace, při které je náhodný gen zaměněn za jinou hodnotu z dané množiny, a křížení podle následujícího pravidla:
potomek = [první_rodič[:bod_křížení], p_new, druhý_rodič[bod_křížení + 1:]
p_new = první_rodič[bod_křížení] + beta * (první_rodič[bod_křížení] – druhý_rodič[bod_křížení]),
kde beta je náhodné číslo mezi 0 a 1 a může být kladná nebo záporná podle toho, zda se jedná o prvního nebo druhého potomka [19].

4.1.5 PyGAD

PyGAD je knihovna open-source v jazyce Python pro genetické algoritmy a optimalizaci algoritmů strojového učení. Umožňuje optimalizaci různých problémů přizpůsobením fitness funkce. Knihovna obsahuje funkci pro výpočet fitness, ale uživatel může použít vlastní funkci. Tato funkce musí být maximalizační. Počáteční populace je vytvořena náhodně jako NumPy pole. Lze specifikovat nejnižší a nejvyšší hodnoty, kterých geny mohou nabývat. PyGAD obsahuje šest metod pro výběr rodičů k produkci potomků. Jedná se o rank selekci, random selekci, ruletové kolo, SUS, turnajovou selekci a steady state selekci. Steady state selekce v každé generaci vybere několik jedinců s vysokou hodnotou fitness, kteří poté produkují potomky. Jedinci s nízkou hodnotou fitness jsou poté odstraněni a noví potomci jsou umístěni místo nich. Zbytek jedinců je zkopírován do další generace.
PyGAD podporuje různé typy genetických operátorů křížení a mutace [20]. Jedná se o jednobodové, dvoubodové a uniformní křížení a random, swap, scramble, inverzní a adapivní mutaci. Jednobodové a dvoubodové křížení probíhá stejně jak líčí popis v sekci 2.1.3. V uniformním křížení je pro každý gen potomka náhodně vybrán jeden rodič a jeho gen je zkopírován do nového jedince. Random mutace náhodně mění hodnoty některých genů z určeného rozsahu. Počet genů, které budou změněny, lze specifikovat. Swap mutace, inverzní mutace a scramble mutace jsou implementovány shodně s popisem v sekci 2.1.4. Jen u scramble mutace geny musí jít za sebou. Posledním implementovaným způsobem mutace je adaptivní mutace, kde je aplikována řada mutací na základě fitness jedince. Jedinci s vysokou hodnotou fitness prodělají méně mutací než jedinci s nízkou hodnotou fitness. Novější verze PyGAD umožňuje uživateli použít vlastní metody křížení, mutace a selekce.

4.1.6 Jenetics

Jenetics je knihovna napsaná v Javě pro genetické algoritmy, evoluční algoritmy a genetické programování. Architektura knihovny Jenetics je založena na Evolučním proudu implementovaným jako Java proud. Potřebné evoluční kroky probíhají v třídě *Evolution Engine*. V této třídě je i implementovaná fitness funkce, avšak je možno použít vlastní funkci, která může být jak minimalizační, tak maximalizační. Geny lze v Jenetics kódovat pomocí bitů, reálných čísel (int-, double-, long-), nebo pomocí znaků. Další nestandardní způsob kódování genů a jiné datové

struktury včetně operátorů jimi používanými jsou implementovány v modulu *io.jenerics.ext*, kterým lze knihovnu rozšířit.

Knihovna Jenetics poskytuje několik různých metod pro selekci, křížení i mutaci, nicméně ji lze rozšířit o vlastní metody. Jsou zde zahrnuty tyto druhy selekce: turnajová selekce, ruletové kolo, SUS, lineární rank selekce, exponenciální rank selekce, Boltzmannova selekce, elitní selekce, pravděpodobnostní selekce, truncation selekce a Monte Carlo selekce. V Boltzmannově selekci pravděpodobnost selekce záleží na daném parametru. Jestliže je pravděpodobnost parametru kladná, pravděpodobnost selekce jedince s vysokou hodnotou fitness se zvýší a naopak. Elitní selekce kopíruje malý počet nejlepších jedinců do další generace bez jakýchkoliv změn. Pravděpodobnostní selekce funguje podobně jako fitness proporcionální druhy selekce s tím rozdílem, že fitness je nahrazeno pravděpodobností výběru jedince. V truncation selekci jsou jedinci seřazeni podle jejich fitness a pouze daný počet nejlepších jedinců je vybrán. Monte Carlo selekce vybírá náhodně jedince z populace.

Druhy křížení používané v Jenetics jsou jednobodové a vícebodové (definované stejně jak je popsáno v sekci 2.1.3), uniformní, line crossover, intermediate crossover a partially-matched crossover. V uniformním křížení jsou geny na i-té pozici dvou rodičů prohozeny s danou pravděpodobností. Line crossover zachází s chromozomy jako s vektory reálného čísla a vrací potomka, který leží na přímce spojující oba rodiče. Pokud číselné hodnoty nového genu jsou mimo povolené meze, je použit původní gen a nově vygenerovaný gen je zahozen. Vnitřní parametry, které určují bod křížení, jsou generovány jednou pro celý vektor (chromozom). Intermediate crossover se chová podobně jako line crossover, hlavní rozdíl je v generování vnitřních parametrů. Ty jsou vytvořeny pro každý gen v chromozomu, namísto jednou pro všechny geny. Pokud nově vytvořený gen přesahuje povolené meze, je vygenerován jiný. Partially-matched crossover garantuje, že všechny geny se nachází v chromozomu přesně jednou. Je vybrán úsek genů jednoho rodiče, který je poté zkopírován na stejnou pozici do druhého rodiče. Pokud se v takto vzniklém jedinci některé geny opakují, jsou nahrazeny geny od prvního rodiče tak, aby byl každý gen v potomkovi jednou. Žádný gen není tedy duplikován při použití tohoto křížení.

Pro mutaci jsou implementovány dvě metody. Swap mutace, která mění pořadí genů v chromozomu s nadějí, že příbuzné geny budou blízko sebe. Gaussova mutace, kde nové hodnoty genů jsou vybrány podle Gaussova rozložení kolem aktuální hodnoty geny [21].

4.1.7 GPLAB

GPLAB je všestranný a snadno rozšiřitelný toolbox pro genetické programování v Matlabu. Jedinci v GPLABu jsou reprezentováni pomocí stromových struktur. Počáteční populace stromů je vytvořena výběrem náhodných funkcí a terminálů. GPLAB může používat libovolné funkce Matlab, které zajišťují uzavřenost, některé logické funkce a if-then-else příkaz. Funkce, které má algoritmus používat, určí uživatel nastavením parametru funkcí. Jako terminál lze použít jakoukoli konstantu nebo číslo mezi 0 a 1. Deklarace terminálů se provádí podobně jako deklarace funkcí

přímým nastavením parametru terminálů. Maximální hloubka a velikost nových stromů, která je určená zadaným parametrem, nesmí být překročena. Dále toolbox nabízí různé možnosti, jak může programátor ovlivnit strukturu stromů.
GPLAB nabízí tři metody pro výpočet prvotního fitness, které si může uživatel vybrat na základě řešeného problému. Po zavolání jedné z těchto metod je zavolána funkce pro úpravu fitness. Výstup této funkce je poté použit v selekci. Pokud chce uživatel použít jinou metodu výpočtu fitness, je možno ji vytvořit. Fitness funkce jsou implementovány jako minimalizační, tudíž by měla být takto implementována i nová funkce.
V GPLABu jsou rodiče vybíráni pomocí jedné z pěti metod: ruletové kolo, SUS, turnajová selekce, dvojitý turnaj a turnaj lexikografického úsporného tlaku (lexicographic parsimony pressure tournament). Ve dvojitém turnaji probíhá turnaj dvakrát. Nejprve jsou jedinci vybráni na základě hodnoty fitness, a poté na základě velikosti (počtu uzlů). V turnaji lexikografického úsporného tlaku jsou stejně jako v turnajové selekci vybráni náhodně jedinci z populace a nejlepší z nich je vybrán jako rodič. Avšak hlavní rozdíl je v tom, že v této selekční metodě je vybrán jedinec s menším počtem uzlů v případě, že fitness jedinců je stejné.
K vytvoření nových jedinců je možno v GPLABu použít libovolný počet genetických operátorů. Lze i část jedinců zkopírovat do další generace, aniž by byly nějak změněny. Genetické operátory nemusejí vracet jedince vyhovující hloubce nebo velikosti stromu. To je zajištěno filtrovacími funkcemi, které jsou k dispozici. Toolbox poskytuje standardní stromové křížení, stromovou mutaci, shrink mutaci a swap mutaci. Ve stromovém křížení jsou vybrány náhodné uzly od obou rodičů a příslušné větve jsou prohozeny. Ve stromové mutaci je vybrán náhodný uzel rodiče a je nahrazen novým stromem vytvořeným pomocí dostupných funkcí a terminálů. Ve shrink mutaci je vybrán náhodný podstrom z rodičovského stromu a nahrazen jiným podstromem. Ve výjimečném případě se potomek může rovnat rodiči. Ve swap mutaci jsou vybrány dva náhodné podstromy z jednoho rodičovského stromu a jsou prohozeny [22].

4.1.8 GA

Balíček GA poskytuje flexibilní toolbox v jazyce R implementující generické algoritmy. V balíčku GA je obsaženo několik funkcí pro generování počáteční populace a pro použití genetických operátorů [23]. Fitness funkce zde není zahrnuta, proto musí programátor použít vlastní. Balíček GA je kompatibilní s jakoukoli maximalizační funkcí napsanou v jazyce R. Pro generování počáteční populace lze použít tři funkce, jednu pro každou možnou reprezentaci jedinců. Jedinci mohou být reprezentováni buď binárně, jako reálné hodnoty pomocí typu float, nebo jako seznam hodnot typu int. Pro všechny tři typy jedinců balíček GA obsahuje následující funkce implementující selekci: lineární a nelineární rank selekce, ruletové kolo a turnajová selekce.
Pro binární a reálnou reprezentaci jsou k dispozici jednobodové a uniformní funkce křížení. Pro reálné hodnoty jsou dále implementovány funkce whole a local arithmetic crossover, kde potomci vznikají z váženého průměru rodičů, a blend crossover, kde geny potomků jsou vybrány z daného

rozsahu získaného pomocí genů rodičů. Pro hodnoty typu integer jsou definovány čtyři funkce pro křížení. Cyklické křížení, kde jedinci jsou rozděleni do cyklů. Cyklus je podmnožina, ve které je každý gen jednoho rodiče spárovaný s odpovídajícím genem druhého rodiče. Potomci vznikají náhodným výběrem cyklů od rodičů. Partially matched crossover, která vezme část genů jednoho rodiče a vloží je do druhého rodiče, přičemž zachová původní pořadí. Pořadové křížení, kde jsou vytvořeny dva náhodné body křížení a úsek mezi nimi od prvního rodiče je zkopírován jednomu potomkovi. Poté počínaje od druhého bodu křížení jsou od druhého rodiče zbývající geny zkopírovány na zbývající pozice. Posledním druhem křížení je position-based, kde vybrané geny od prvního rodiče jsou zkopírovány na ty samé pozice do potomka a zbytek genů je doplněn geny od druhého rodiče ve stejném pořadí.

Pro binární a reálnou reprezentaci je implementovaná uniformní mutace. Dále lze pro reálné hodnoty použít náhodnou mutaci a non-uniformní mutaci, kde pravděpodobnost mutace klesá se zlepšující se populací. Pro integer hodnoty si lze vybrat z pěti funkcí pro mutaci – inverzní, insertion, swap, displacement a scramble. Inverzní mutace prohodí sekvenci genů. Insertion mutace vybere náhodný gen a vloží ho na náhodně vybranou pozici. Swap mutace prohodí dva náhodně vybrané geny. Displacement mutace náhodně vybere úsek genů a vloží jej na náhodně vybrané místo. U scramble mutace je náhodně vybrána podmnožina genů a jejich pozice jsou zamíchány.

4.2 Neuronové sítě

4.2.1 Genann

Genann je minimalistická knihovna v jazyce C pro dopředné neuronové sítě. Především je zaměřena na to, aby byla jednoduchá, rychlá a spolehlivá. Proto obsahuje pouze nezbytné funkce. Nicméně se dá snadno rozšířit a je kompatibilní s alternativními trénovacími metodami jako jsou např. genetické algoritmy.

Umělá neuronová síť je v Genann vytvořena pomocí funkce, která za argumenty vyžaduje počet vstupů, výstupů, skrytých vrstev a počet neuronů nacházející se v každé skryté vrstvě. Poté je síť učena pomocí algoritmu zpětného šíření chyby. Jednoduchost a efektivita učení vah neuronové sítě spočívá v tom, že váhy jsou uloženy jako souvislý blok paměti. Proto lze snadno použít numerické optimalizační algoritmy s přímým vyhledáváním [24].

4.2.2 FANN

FANN je všestranná, rychlá a snadno použitelná knihovna implementující vícevrstvé neuronové sítě v jazyce C s propojením do více než 20-ti programovacích jazyků. Podporuje jak zcela propojené, tak i řídce propojené NS. Obsahuje několik funkcí, které zjednodušují vytváření, trénování a testování NS. FANN používá dva zásadně odlišné přístupy: pevnou topologii a vyvíjející se topologii.

V pevné topologii je velikost a topologie NS určena předem a trénink mění váhy tak, aby byl minimalizován rozdíl mezi požadovanými a skutečnými výstupními hodnotami. Při vytváření takovéto neuronové sítě je potřeba definovat kolik vrstev, neuronů a spojení by síť měla mít. Pokud by byla příliš velká, měla by problémy s učením. Naopak příliš malá síť není schopna dosáhnout dostatečně nízké chybovosti. Po nastavení počátečních parametrů probíhá trénování. To je ve FANN provedeno pomocí standardního algoritmu zpětného šíření chyby. Tudíž jsou neustále nastavovány váhy tak, aby výstup NS odpovídal výstupu v tréninkovém souboru. Jeden takový cyklus, ve kterém se váhy upravují, se nazývá epocha. Je důležité specifikovat, kolik epoch by mělo být použito pro učení, aby byla NS byla schopna správně klasifikovat trénovací data. To lze zjistit pomocí testovaní na datech, které NS neviděla během učení. FANN z tohoto důvodu obsahuje rámec pro snadné zacházení s tréninkovými datovými sadami a funkce pro testování dat a jejich manipulaci.

Vyvíjející se (kaskádovitý) trénink začíná s prázdnou NS obsahující pouze vstupní a výstupní neurony. Postupně jsou do ní přidávány neurony, přičemž je neuronová síť trénována. Tudíž není třeba před učením zadávat počet skrytých vrstev a neuronů. FANN pro tento druh učení používá Cascade2 trénovací algoritmus s vlastními vestavěnými funkcemi [25].

4.2.3 OpenNN

OpenNN je knihovna pro neuronové sítě napsaná v programovacím jazyce C++. Tato knihovna vyniká z hlediska rychlosti a alokace paměti. Koncepční model OpenNN představuje pět tříd – *DataSet*, *NeuralNetwork*, *TrainingStrategy*, *ModelSelection a TestingAnalysis*. Třída *DataSet* obsahuje nástroje pro zpracování dat a sestavení modelu. Třída *NeuralNetwork* je odpovědná za sestavení NS a řádnou organizaci vrstev neuronů. Obsahuje několik druhů předdefinovaných vrstev, z toho jsou čtyři vrstvy použitelné pro učení (perceptronová, pravděpodobnostní, rekurentní a LSTM vrstva). Tato třída potřebuje znát čtyři argumenty: typ modelu (aproximační, klasifikační, predikční), počet vstupů, počet neuronů v perceptronové vrstvě a počet neuronů v pravděpodobnostní vrstvě. Třída *TrainingStrategy* provádí trénování NS. Skládá se ze dvou abstraktních tříd: *LossIndex*, která odkazuje na typ chyb, *a OptimizationAlgorithm*, která se používá k optimalizaci NS. *ModelSelection* se používá pro nalezení takové topologie neuronové sítě, která má umožňuje maximální zobecnění a minimalizuje chybu pro nová data. OpenNN nabízí dva způsoby, jak získat optimální topologii: výběr neuronů a výběr vstupů. Výběr neuronů se používá pro získání ideálního počtu neuronů v perceptronové vrstvě. Výběr vstupů hledá optimální podmnožiny vstupů. Pro vyhodnocení modelu se používá třída *TestingAnalysis*. Ta slouží k ověření výkonu modelu. Porovnává výstupy neuronové sítě s jejich odpovídajícími cíli v testovacích sadách [26].

4.2.4 TensorFlow

TensorFlow je knihovna vytvořená týmem Google Brain pro strojové učení a umělou inteligenci. Poskytuje rozhraní zejména pro jazyk Python, ale lze ji použít i pro jiné programovací jazyky jako např. Javascript, C++, Java. TensorFlow umožňuje vývojářům vytvářet grafy toku dat, tedy struktury popisující pohyb dat v grafu nebo sérii uzlů. Každý uzel v grafu představuje matematickou operaci a každé spojení neboli hrana mezi uzly je vícerozměrné datové pole čili tenzor. Všechny tenzory jsou neměnné, tedy není možné aktualizovat obsah tenzoru, pouze vytvořit nový. Pro reprezentaci hodnot, které je potřeba měnit v průběhu algoritmu (synaptické váhy), lze použít TensorFlow proměnné [27].

Vysokoúrovňové rozhraní pro knihovnu TensorFlow je poskytováno knihovnou Keras. Keras umožňuje jednodušší použití, jelikož je uživatelsky přívětivý a snadno rozšiřitelný. Nicméně TensorFlow lze použít i samostatně, obzvláště pokud je potřeba rychlé učení sítě a vysoký výkon.

4.2.5 Keras

Keras je knihovna hlubokého učení v jazyce Python. Keras byl integrován do TensorFlow, který slouží jako výchozí backend pro Keras. Základní datové struktury v Kerasu jsou vrstvy a modely. Každá vrstva přijímá tensory jako vstupy, provádí výpočty a nakonec produkuje výstupní tensory. Keras vrstva potřebuje znát tvar vstupu k pochopení struktury vstupních dat, inicializátor pro nastavení vah a aktivační (převodní) funkci. Lze nastavit i další argumenty, jako např. rozsah hodnot vah nebo regulátory pro optimalizaci. Keras nabízí spoustu vestavěných vrstev, ale lze si jednoduše vytvořit vlastní.

Neuronová síť je reprezentována modelem Keras. Jsou k dispozici dva druhy modelů, a to *Sequential* model *a Functional* API. *Sequential* model je nenáročný a snadný k používání. Je vhodný pro jednoduchý zásobník vrstev, kde každá vrstva má jeden vstupní tenzor a jeden výstupní tenzor. Pro flexibilnější a více pokročilejší použití je zde *Functional* API. Ten si poradí s nelineární topologií, sdílenými vrstvami i s více vstupy a výstupy.

Po nadefinování modelu probíhá jeho trénování. To probíhá po epochách a každá epocha je rozdělena do dávek (batch). Pro trénování modelu je potřeba určit ztrátovou funkci (loss function), optimalizátor, případně další parametry pro monitorování učícího procesu. Lze si vybrat z již vestavěných funkcí nebo implementovat vlastní. Učící proces probíhá po pevně stanovený počet iterací. Nakonec je model vyhodnocen pomocí testovacích dat [28].

5 Demonstrace využití knihoven

Pro demonstrací využití knihoven jsem si vybrala dvě knihovny: jednu implementující genetický algoritmus (OpenGA) a jednu implementující neuronovou síť (OpenNN). Výběr jsem konzultovala s vedoucím práce. K výběru těchto knihoven mě vedly dva důvody:

- Fakt, že dnes je propagován jazyk Python a knihovna TensorFlow, eventuálně Matlab, a proto jsem chtěla vyzkoušet knihovny v jiném jazyku.
- Jazyk C/C++ zaručuje rychlost a efektivitu.

Kódy byly laděny pod MS Windows 10, procesorem Intel(R) Core(TM) i5-1035G4 CPU @ 1.10 GHz 1.50 GHz, 8 GB pamětí, v prostředí CodeBlocks 20.03, 64 bitovým překladačem gcc 8.0.1.

5.1 Demonstrace knihovny OpenGA

Použití knihovny OpenGA je demonstrováno na řešení logického problému (dopravní problém nebyl zvolen, protože řešení problému obchodního cestujícího je mezi příklady v knihovně).

Knihovna OpenGA poskytuje rámec pro genetický algoritmus. Genetický algoritmus je definován jako šablona třídy. Jelikož metody pro počáteční inicializaci chromozomu, operace křížení a mutace a fitness funkci nejsou v knihovně nijak implementovány, jsou tyto operace implementovány jako samostatné funkce a jejich adresy jsou přiřazeny v rámci inicializace genetického algoritmu do atributů třídy – ukazatelů na příslušné funkce. Celá knihovna je definována v jediném hlavičkovém souboru openGA.hpp. Není ji tedy nutné samostatně překládat, stačí pouze vložit hlavičkový soubor pomocí direktivy #include. Knihovna využívá paralelismu, je implementován eliticismus.

Knihovna má také některé "nedostatky". Manuál je stručný a pro zjištění detailů je potřeba prozkoumat zdrojový kód knihovny. Selekce vybírá jedince s menší hodnotou fitness funkce, GA tedy minimalizuje problém a nelze to změnit žádným nastavením. Zastavovací kritérium je také pevně zabudováno a nelze jej modifikovat: algoritmus se zastaví, pokud je dosaženo daného počtu generací nebo se po určitý počet generací (lze nastavit) nemění hodnota fitness nejlepšího jedince nebo průměrná hodnota fitness v generaci. Nelze zastavit algoritmus při dosažení určité hodnoty fitness funkce; je zde ale možnost uživatelského zastavení algoritmu. Pokud chce uživatel využít pouze podmínku dosažení určitého počtu generací, musí se nastavit limit maximálního počtu generací, kdy nedochází ke změně průměrné, resp. nejlepší fitness funkce, na hodnotu rovnu maximálnímu počtu generací (parametry best_stall_max, average_stall_max).

Logický problém je definován následovně:

Na pouti je pět chlapců různého věku. Konzumují různé druhy jídel a každý dává přednost různým atrakcím. Podmínky jsou následující:

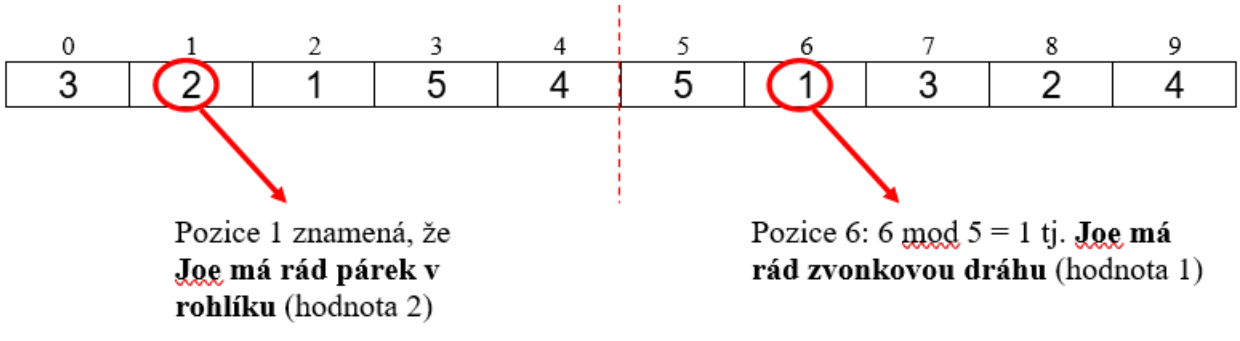
1. Ron jí zmrzlinu,
2. Joe nemá rad žvýkačky
3. Samovi je 14
4. Sam není na horské dráze
5. Chlapci na zvonkové dráze je 15
6. Len není ve strašidelném zámku
7. Don je na kolotoči

8. Chlapci, který nemá rád zmrzlinu, je 13 let
9. Chlapec ve strašidelném zámku jí párek v rohlíku
10. Joe je jedenáctiletý
11. Joe jí hranolky
12. Joe je na lochnesce
13. Donovi je 12 let
14. Don si pochutnává na cukrové vatě

Chromozom je implementován jako vektor <int> z knihovny STL o velikosti 15 položek (genů). Každá pětice kóduje určitou vlastnost (jídlo, atrakci, věk). Je zavedeno následující kódování:

Jména	Jídla	Atrakce	Věk
0 – Ron	0 – žvýkačka	0 – horská dráha	0 – 11
1 – Joe	1 – zmrzlina	1 – zvonková dráha	1 – 12
2 – Sam	2 – párek v rohlíku	2 – strašidelný zámek	2 – 15
3 – Len	3 – hranolky	3 – kolotoč	3 – 13
4 – Don	4 – cukrová vata	4 – lochneska	4 – 14

Část konkrétního chromozomu je na obrázku 10.

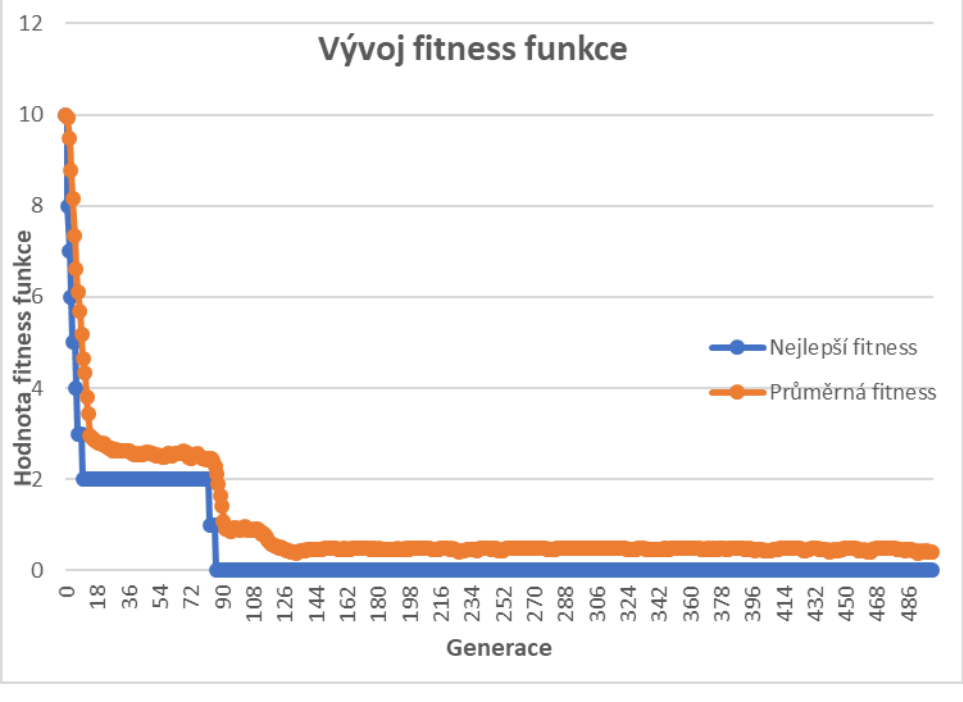


Obrázek 10: Ukázka konkrétního chromozomu

Hodnotící funkce (evaluation function) počítá počet splněných pravidel, ale vzhledem k tomu, že knihovna minimalizuje problém, ukládá do hodnoty middle cost počet nesplněných pravidel. Fitness funkce vrací hodnotu middle cost (tedy počet nesplněných pravidel) beze změny. Při křížení (jednobodovém) je náhodně zvolena jedna ze dvou hranic mezi pěticemi a vytvořen potomek. Křížení tvoří pouze jednoho potomka (tak je navržen prototyp funkce pro křížení v knihovně). Při mutaci je zvolena jedna z pětic a jsou prohozeny vlastnosti dvou náhodně vygenerovaných chlapců (např. Joe a Len si vymění jídla).
V příloze je ukázka výpisu z jednoho algoritmu. V následujícím textu je fragment hlavního programu s nastavením parametrů GA. Vlastní genetický algoritmus se spouští metodou solve:

```
GA ga;
ga.problem_mode= EA::GA_MODE::SOGA; // single opt. problem
ga.multi_threading=true;
ga.idle_delay_us=1;
ga.verbose=false;
ga.population=200;
ga.generation_max=500;
ga.calculate_SO_total_fitness=finalni_fitness;
ga.init_genes= inicializuj_chromozom;
ga.eval_solution= ohodnot_reseni;
ga.mutate= mutace;
ga.crossover= krizeni;
ga.SO_report_generation= statistika_generace;
ga.best_stall_max=1000;
ga.average_stall_max=1000;
ga.elite_count=100;
ga.crossover_fraction=0.8;
ga.mutation_rate=0.4;
ga.solve();
```

Vývoj hodnot fitness funkce pro jeden běh algoritmu ukazuje graf na obrázku 11.

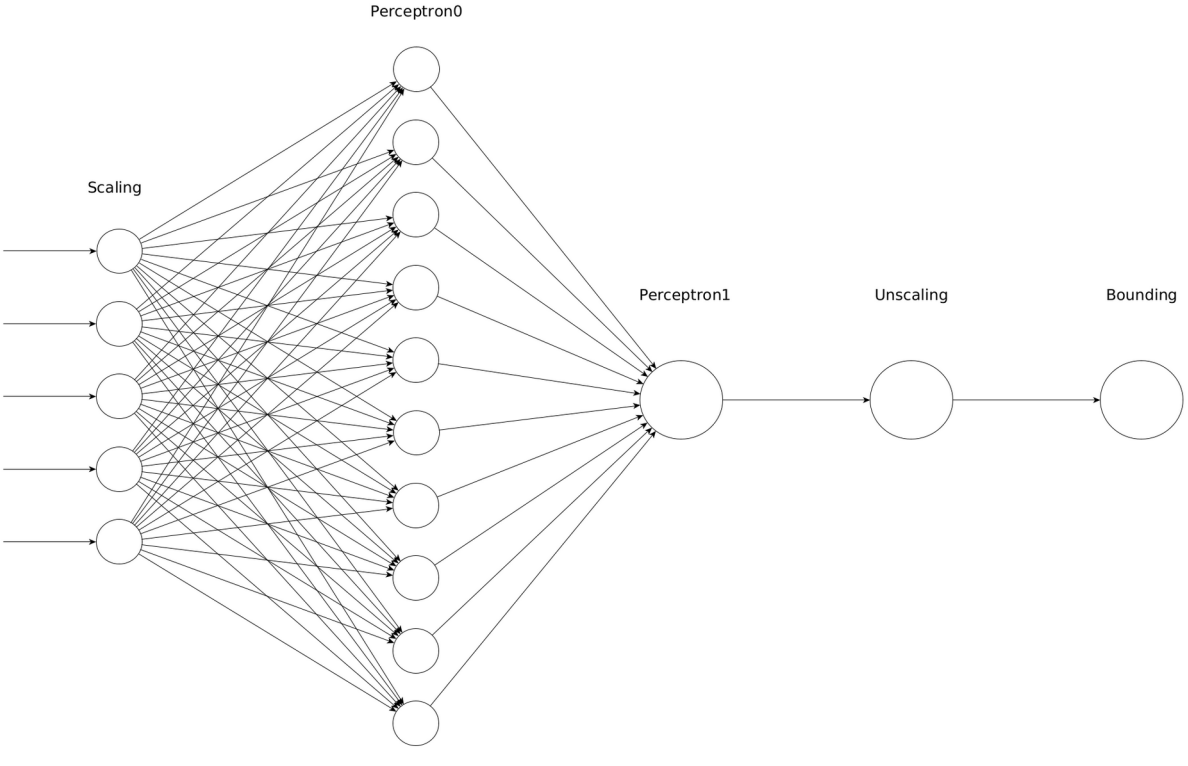


Obrázek 11: Vývoj hodnot fitness funkce pro jeden běh algoritmu

5.2 Demonstrace knihovny OpenNN

Ukázka využití knihovny OpenNN je demonstrována na příkladu predikci hluku (Airfoil self-noise prediction) z tutoriálu knihovny, který byl upraven a doplněn. Jedná se o aproximační příklad, jehož cílem je předpovědět hluk generovaný profilem křídla letadla. Předpovídaná proměnná je spojitá. Datová sada [29] vychází z měření v aerodynamickém tunelu a obsahuje šest proměnných: frekvence [Hz], úhel proudu vzduchu a profilu křídla [°], hloubka profilu křídla [m], rychlost proudícího vzduchu [m/s], pošinovací tloušťka (displacement thickness) vztlakové hrany [m] a predikce úrovně hluku [dB]. Prvních pět proměnných tvoří vstupní proměnné, poslední proměnná je cílová. Před trénováním jsou data náhodně rozdělena na tři sady – trénovací, testovací a selektivní.

Architektura neuronové sítě pro aproximační problém je dána typem vytvořené sítě (Approximation) a skládá se ze Scaling vrstvy, dvou perceptronových vrstev, Unscaling vrstvy a Bounding vrstvy. Perceptronové vrstvy Perceptron0 a Perceptron1 jsou jediné vrstvy, které podléhají procesu učení. Vrstva Perceptron0 obsahuje 10 neuronů a každý neuron má 5 vstupů (jeden od každého neuronu z vrstvy Scaling), vrstva Perceptron1 obsahuje jeden neuron s deseti vstupy. Síť je vytvořena voláním konstruktoru třídy *NeuralNetwork* s parametry: typ sítě Approximation, 5 vstupů, 10 neuronů ve skryté vrstvě a 1 výstup. Struktura sítě je zobrazena na obrázku 12.



Obrázek 12: Struktura aproximační neuronové sítě

Vrstvy Scaling a Unscaling provádějí lineární transformaci dat (škálování), vrstva Bounding pouze omezuje výstupní hodnoty, aby nepřekročily práh. Na základě analýzy trénovacích dat jsou ještě před učením nastaveny parametry neuronů ve vrstvách Scaling, Unscaling a Bounding pomocí knihovních funkcí OpenNN.

Jako trénovací metoda je použita Adaptive Moment Estimation (ADAM), která je implementovaná v knihovně OpenNN. Metoda používá implicitně ztrátovou funkci *Normalized Squared Error*. Trénování probíhalo v 10 000 epochách. Výstup jednoho běhu trénování je uložen v příloze jako soubor vystup.txt. Nalezená síť vykazovala trénovací chybu 0,224086, nalezené váhy a prahy perceptronových vrstev jsou uvedeny níže:

```
Perceptronova vrstva 0:
Prahy (biases) jednotlivych neuronu:
 0.0479335 0.599943 0.828777 -1.81386 -0.713279 -0.937972 -1.43923 -0.0362114 0.817199 -0.796321

Synapticke vahy:
 0.645784  0.737923 -1.06413 -1.77041  -0.141137 -0.188181 -0.715758 0.318084  0.757461 -1.26057
-0.187568  0.197409  1.28877  0.0681962 0.801009 -0.918991  0.677267 0.0582528 0.23992  -0.248834
-0.229312 -0.783655 -0.0152489 -1.07066 0.472834 -0.0125244 -0.602452 -0.414862 0.379107 -0.611188
 0.274305 -0.00685422 0.271319 0.101361 -0.310691 -0.368301 -0.348375 -0.518466 -0.309153 0.198844
 0.105614 -0.650533 -0.0677479 -0.317257 -0.286756 0.196967 -0.0554317 0.907383 -0.294739 -0.653882
-------------------------
Perceptronova vrstva 1:
Prahy (biases) jednotlivych neuronu:
-0.0575219

Synapticke vahy:
 0.513534
-0.934441
-0.760233
    1.054
 0.565027
-0.771886
 -1.09996
 0.608132
  1.01263
 1.30377
```

5.3 Propojení knihoven OpenGA a OpenNN

Poslední příklad využívá obě knihovny OpenGA a OpenNN současně. Jde o jednoduchý genetický algoritmus, který hledá váhy neuronové sítě z předchozí ukázky. Trénovací a testovací data jsou stejná, struktura sítě je totožná se sítí v předešlé ukázce a nemění se během procesu učení.

Síť je vytvořena voláním konstruktoru se stejnými parametry, tedy 5 vstupů, 10 neuronů ve skryté vrstvě, 1 výstup, síť typu Approximation. Na základě analýzy trénovacích dat jsou nastaveny parametry neuronů ve vrstvách Scaling, Unscaling, Bounding pomocí knihovních funkcí OpenNN, ekvivalentně příkladu 5.2. Procesu učení podléhají neurony vrstev Perceptron0 a Perceptron1.

Chromozom je reprezentován lineárním vektorem reálných čísel o počtu 71 genů, je využita třída Tensor, kterou knihovna OpenNN používá. Každý neuron vrstvy Perceptron0 má jeden práh a 5 vah pro pět vstupů, vrstva Perceptron0 s 10 neurony je tedy charakterizována (5+1)*10=60 parametry. Vrstva Perceptron1 má 1 neuron s deseti vstupy, jeho parametrů je 11 (10 vah a práh). Odtud 71 genů v chromozomu. Interní struktura chromozomu je následující: prvních deset genů jsou prahy nulté vrstvy, deset pětic reprezentuje synaptické váhy neuronů nulté vrstvy. Posledních 11 genů je práh a deset vah neuronů vrstvy 1. Váhy vrstev sítě jsou nastavovány ve funkci `ohodnot_chromozom` (hodnotící funkce – evaluation function knihovny OpenGA) před výpočtem trénovací chyby pomocí metody `set_parameters` třídy *PerceptronLayer*. Interní struktura chromozomu byla navržena dle požadovaných parametrů metody – první část tedy reprezentuje prahy.

Chromozom je generován náhodně, každý gen v chromozomu je inicializován náhodně hodnotou z intervalu (-2, 2) s rovnoměrným rozdělením; interval (-2, 2) byl pro jednoduchost zvolen na základě analýzy naučené sítě z předchozího příkladu.

Operace křížení je klasické jednobodové křížení. Operace mutace přičte ke každému genu náhodné číslo z intervalu <-0.2, 0.2>. První pokusy totiž ukázaly, že změna pouze jednoho genu v rámci operace mutace měla za následek velmi pomalou konvergenci a uváznutí v lokálním extrému.

Hodnotící funkce nastaví váhy sítě a spočítá chybu predikce. Bohužel metody třídy *LossIndex* nelze přímo využít, neboť využívají některé soukromé atributy třídy trénovacích metod. Je tedy využita třída *TestingAnalysis* a její metoda *calculate_training_errors*, která vrací tenzor se čtyřmi hodnotami různých chyb (citace z kódu):

```
errors(0) = sum_squared_error(0);
errors(1) = errors(0)/training_samples_number;
errors(2) = sqrt(errors(1));
errors(3) = calculate_normalized_squared_error(targets, outputs);
```

Všechny čtyři hodnoty uloží hodnotící funkce jako middle costs. Fitness funkce pak vrací poslední hodnotu, normalizovanou kvadratickou chybu.

Genetický algoritmus byl spuštěn desetkrát, parametry algoritmu jsou následující:

```
GA_NN ga;
        ga.problem_mode= EA::GA_MODE::SOGA;
        ga.multi_threading=false;
        ga.idle_delay_us=1;
        ga.verbose=false;
        ga.population=500;
        ga.generation_max=2000;
        ga.calculate_SO_total_fitness=finalni_fitness;
```

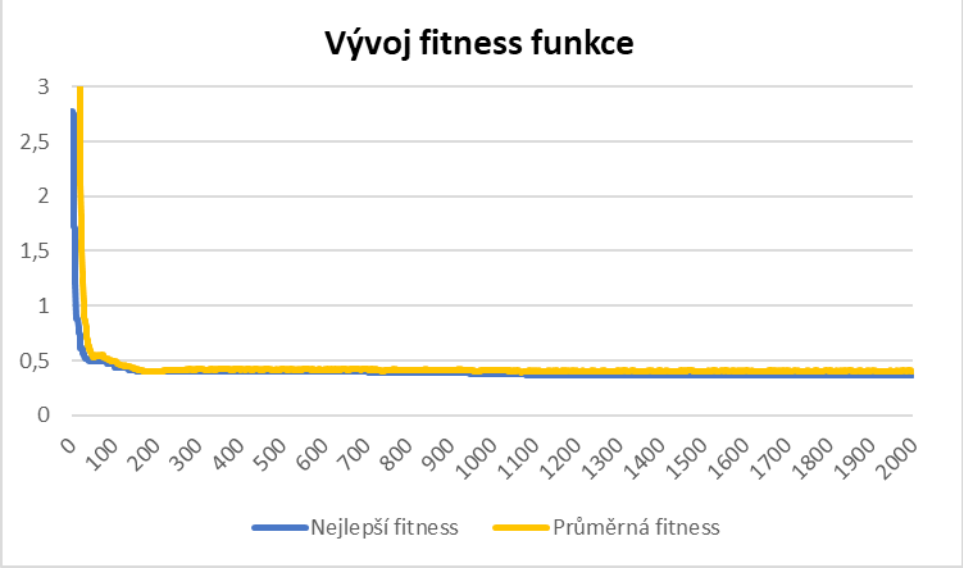
```
ga.init_genes= inicializuj_chromozom;
ga.eval_solution= ohodnot_Chromozom;
ga.mutate= mutace;
ga.crossover= krizeni;
ga.SO_report_generation= statistika_generace;
ga.best_stall_max=2000;
ga.average_stall_max=2000;
ga.elite_count=100;
ga.crossover_fraction=0.8;
ga.mutation_rate=0.6;
ga.solve();
```

Hodnota fitness funkce nejlepšího nalezeného řešení v každém běhu je v následující tabulce.

Běh algoritmu	1	2	3	4	5	6	7	8	9	10
Nejlepší fitness	0,371	0,406	0,399	0,467	0,421	0,377	0,576	0,408	0,392	0,376

Tabulka 1: Hodnota fitness funkce v každém běhu

Vývoj nejlepšího a průměrného řešení v běhu 1 je na následujícím grafu.



Obrázek 13: Vývoj nejlepší a průměrné hodnoty fitness funkce v běhu 1

Nalezené váhy a prahy nejlepšího řešení:

```
Perceptronova vrstva 0:
Prahy (biases) jednotlivych neuronu:
-0,325097 -0,428794 1,96828 0,499591 2 -0,992839 -1,49569 1,61202 2 -1,18458
```

```
Synapticke vahy:
 -0,768066 -1,82709 -1,40804 1,27545 -0,950687 0,0703713 1,79728 -2 -1,59896 -1,87253
 0,30137 0,497255 1,7454 1,98001 -1,75876 0,591263 -1,28663 1,45384 -2 -0,0960283
 -0,275235 -0,92849 1,12942 -0,773938 -0,234628 -1,24439 -0,13461 1,28321 -0,914898 -0,534661
 1,73741 0,444504 -1,74445 0,057354 -0,237035 0,482162 0,470202 -0,772413 0,0200737 0,0258446
 1,67955 -0,0158676 -1,34895 -0,425986 -0,508629 0,31236 0,420446 -0,062522 2 -0,896139
------------------------
Perceptronova vrstva 1:
Prahy (biases) jednotlivych neuronu:
-1,84214

Synapticke vahy:
0,0114699
0,0900881
 0,422348
-0,293695
 1,58539
 0,412465
 0,936142
-0,158217
 0,755445
 0,63692
```

Bohužel takto jednoduchý algoritmus vykazuje horší vlastnosti než implementovaná metoda Adaptive Moment Estimation. Pro srovnání, trénovací chyba (normalized squared error) u metody Adaptive Moment Estimation byla 0,224, zde u nejlepšího řešení 0,372. Rovněž čas běhu je u genetického algoritmu výrazně delší (108 s oproti 1 s). Avšak toto srovnání není relevantní, jelikož u genetického algoritmu musel být vypnut paralelismus (docházelo k zamrznutí programu). V následující tabulce je srovnána odpověď sítě na tři vstupní vektory.

Vstupní vektor	Odpověď sítě naučené metodou ADAM	Odpověď sítě naučení GA
850,0,0.3048,71.3,0.00266337	128.327	126.676
1100,0,0.3048,71.3,0.00266337	127.957	126.389
16020,0,0.3048,71.3,0.00266337	112.469	117.996

Tabulka 2: Porovnání výsledků metody ADAM a GA

Algoritmus lze vylepšit na základě analyzovaných článků, nejjednodušší cesta se nyní jeví: v případě stagnace hodnoty fitness funkce zvýšit pravděpodobnost mutace, resp. tuto operaci modifikovat.

6 Závěr

V této práci byly představeny teoretické základy evolučních technik jako jsou genetické algoritmy, genetické programování, evoluční strategie a diferenciální evoluce. Blíže zde byly popsány genetické operátory selekce, křížení a mutace tak, jak se nejčastěji používají v genetických algoritmech. Rovněž tady byl vysvětlen princip neuronových sítí.

Byly zde uvedeny některé způsoby, jak pomocí genetických algoritmů optimalizovat neuronové sítě. Avšak bylo zjištěno, že genetické algoritmy nejsou vždy nejlepší volbou, jelikož dokáží řešit problémy pouze diskrétní metodou. Jako lepší řešení se jeví diferenciální evoluce, které používají vektory. Z nalezených článků vyplývá, že diferenciální evoluce dosahuje rychlejší konvergence a je o něco účinější při učení neuronových sítí než genetický algoritmus. Nicméně jak genetický algoritmus, tak diferenciální evoluce se ukazují být lepší možnost oproti klasickému algoritmu zpětného šíření chyby, který se běžně používá k učení neuronových sítí.

Dále tu byly popsány knihovny implementující evoluční techniky i neuronové sítě. Popis knihoven evolučních technik se soustředil především na různé druhy křížení a mutace, které jsou v knihovnách naprogramovány, jelikož se jedná o stěžejní části evolučních technik. Též popis pojednával o tom, jak odlišné knihovny implementují reprezentaci jedinců v chromozomu. Charakteristika knihoven neuronových sítí se zaměřovala na jejich fungování a způsob trénování. Bohužel u některých knihoven dokumentace neobsahovala všechny potřebné informace, proto bylo potřeba některé detaily zjistit ze zdrojového kódu.

V poslední části bylo demonstrováno použití knihovny OpenGA na řešení logického problému genetickým algoritmem, OpenNN (neuronová síť) na příkladu aproximace úrovně hluku a nakonec jednoduché učení neuronové sítě genetickým algoritmem využívající obě knihovny, resp. jejich propojení. Knihovna OpenGA poskytuje uživateli poměrně velkou flexibilitu, avšak některé její postupy (minimalizace fitness funkce, pevně zabudované zastavovací kritérium) řešení o něco zkomplikovaly, jelikož nešly změnit. Problém se vyskytl i u knihovny OpenNN, a to že nešly použít metody třídy *LossIndex,* jelikož využívají některé soukromé atributy jiné třídy. Nakonec byly tyto knihovny zkombinovány a neuronová síť byla naučena pomocí genetického algoritmu. Bohužel se nepodařilo naučit neuronovou síť pomocí genetického algoritmu na lepší výsledek, než kterého bylo dosaženo pomocí metody ADAM. Jednak byla trénovací chyba u GA vyšší, jednak čas běhu GA byl výrazně delší. Nicméně u genetického algoritmu byl vypnut paralelismus, takže je potřeba brát toto srovnání s nadhledem.

Uvedené poznatky mohou v budoucnu sloužit nejen jako podklad k dalším výzkumům a projektům zabývající se otázkou učení neuronových sítí pomocí evolučních technik, ale i k výběru vhodné knihovny pro daný problém.

7 Použitá literatura

[1] Holland, J. H. *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975

[2] Fogel, L., Owens, A.J., Walsh, M.J. *Artificial Intelligence through Simulated Evolution*. Wiley, New York, 1966

[3] Mitchel, Melanie. *An Introduction to Genetic Algorithms* [online]. Cambridge: MIT Press, 2002 [cit. 2022-08-05]. ISBN 02-626-3185-7. Dostupné z: https://www.boente.eti.br/fuzzy/ebook-fuzzy-mitchell.pdf

[4] Miller, Brad L., Goldberg, David E., Genetic Algorithms, Tournament Selection, and the Effects of Noise*. Complex Systems.*[online]. 1995, (9), 193-212 [cit. 2022-08-06]. Dostupné z: https://wpmedia.wolfram.com/uploads/sites/13/2018/02/09-3-2.pdf

[5] Koza, John R. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* Cambridge: Bradford Book, 1992. ISBN 02-621-1170-5.

[6] Beyer, HG., Schwefel, HP. Evolution strategies – A comprehensive introduction. *Natural Computing 1***,** 2002, 3–52 [cit. 2022-08-06]. Dostupné z: https://doi.org/10.1023/A:1015059928466

[7] Emmerich Michael, Ofer M. Shir a Hao Wang. Evolution Strategies. *Handbook of Heuristics* [online]. Cham: Springer International Publishing, 2018, 2018-08-14, 89-119 [cit. 2022-08-06]. ISBN 978-3-319-07123-7. Dostupné z: doi: 10.1007/978-3-319-07124-4_13

[8] Šnorek, Miroslav. *Neuronové sítě a neuropočítače* [online]. Praha: Vydavatelství ČVUT, 2002. [cit. 2022-08-06]. ISBN 80-010-2549-7.

[9] IBM Cloud Education. Neural Networks. In: *Ibm.com* [online]. IBM, 2020 [cit. 2022-08-06]. Dostupné z: https://www.ibm.com/cloud/learn/neural-networks

[10] Navlani, Avinash. Neural Network Models in R. *Datacamp.com* [online]. 2019 [cit. 2022-08-01]. Dostupné z: www.datacamp.com/tutorial/neural-network-models-r

[11] David J. Montana and Lawrence Davis. Training feedforward neural networks using genetic algorithms. In: *Proceedings of the 11th international joint conference on Artificial intelligence* - Volume 1 (IJCAI'89). 1989, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 762–767.

[12] Ding, Shifei, Chunyang Su a Junzhao Yu. An optimizing BP neural network algorithm based on genetic algorithm. *Artificial Intelligence Review* [online]. 2011, **36**(2), 153-162 [cit. 2022-08-01]. ISSN 0269-2821. Dostupné z: doi:10.1007/s10462-011-9208-z

[13] Wdaa, Abdul Sttar Ismail. *DIFFERENTIAL EVOLUTION FOR NEURAL NETWORKS LEARNING ENHANCEMENT*. Malaysia, 2008. Thesis. Universiti Teknologi, Faculty of Computer Science and Information System.

[14] Baioletti, Marco, Gabriele Di Bari, Alfredo Milani a Valentina Poggioni. Differential Evolution for Neural Networks Optimization. *Mathematics* [online]. 2020, **8**(1) [cit. 2022-08-06]. ISSN 2227-7390. Dostupné z: doi:10.3390/math8010069

[15] Si, Tapas, Simanta Hazra a N. D. Jana. Artificial Neural Network Training Using Differential Evolutionary Algorithm for Classification. *Proceedings of the International Conference on Information Systems Design and Intelligent Applications 2012 (INDIA 2012) held in Visakhapatnam, India, January 2012* [online]. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, 2012, 769-778 [cit. 2022-08-01]. Advances in Intelligent and Soft Computing. ISBN 978-3-642-27442-8. Dostupné z: doi:10.1007/978-3-642-27443-5_88

[16] Mohammadi, Arash & Asadi, Houshyar & Mohamed, Shady & Nelson, Kyle & Nahavandi, Saeid. 2017. openGA, a C++ Genetic Algorithm library. 10.1109/SMC.2017.8122921.

[17] Mallet, Olivier. GALGO-2.0. *Github.com* [online]. c2017 [cit. 2022-08-01]. Dostupné z: github.com/olmallet81/GALGO-2.0

[18] Wall, Matthew. *GAlib: A C++ Library of Genetic Algorithm Components* [online]. Cambridge (Massachusetts) Mechanical Engineering Department Massachusetts Institute of Technology. c1996 [cit. 2022-08-01]. Dostupné z: http://lancet.mit.edu/ga/dist/galibdoc.pdf

[19] Chaves, Diogo Matos. GeneAl. *Github.com* [online]. c2020 [cit. 2022-08-01]. Dostupné z: github.com/diogomatoschaves/geneal

[20] Gad, Ahmed Fawzy. *PyGAD: Python Genetic Algorithm!* [online]. c2020 [cit. 2022-08-01]. Dostupné z: pygad.readthedocs.io/en/latest/README_pygad_ReadTheDocs.html

[21] Wilhelmstötter, Franz. *JENETICS: LIBRARY USER'S MANUAL 7.1* [online]. c2022 [cit. 2022-08-01]. Dostupné z: https://jenetics.io/manual/manual-7.1.0.pdf

[22] Silva, Sara. *GPLAB: A Genetic Programming Toolbox for MATLAB* [online]. Portugal, 2007 [cit. 2022-08-01]. Dostupné z: https://master.dl.sourceforge.net/project/gplab/gplab/3.0/gplab.manual.3.pdf?viasf=1

[23] Scrucca, Luca. GA: A Package for Genetic Algorithms in R. *Journal of Statistical Software* [online]. 2013, **53**(4) [cit. 2022-08-06]. ISSN 1548-7660. Dostupné z: doi: https://doi.org/10.18637/jss.v053.i04

[24] Van Winkle, Lewis. *C Neural Network Library: Genann* [online]. Iowa, USA, 2020 [cit. 2022-08-03]. Dostupné z: https://github.com/codeplea/genann

[25] Nissen, Steffen. *Fast Artificial Neural Network Library* [online]. 2003 [cit. 2022-08-03]. Dostupné z: http://leenissen.dk/fann/wp/

[26] Martin, P., Barranquero C., Sanchez J., et al. *OpenNN: Open Neural Network Library* [online]. 2021 [cit. 2022-08-02]. Dostupné z: https://www.opennn.net/

[27] Abadi M., Ashish A., Barham P., et al. *TensorFlow: Large-scale machine learning on heterogeneous systems* [online]. 2015 [cit. 2022-08-03]. Dostupné z: tensorflow.org.

[28] Chollet, F., et al. *Keras* [online]. GitHub. 2015. Dostupné z: https://github.com/fchollet/keras

[29] Dua, D. and Graff, C. UCI Machine Learning Repository. [online] Irvine, CA: University of California, School of Information and Computer Science. 2019 [cit. 2022-08-01]. Dostupné z: http://archive.ics.uci.edu/ml

8 Seznam obrázků

9 Seznam tabulek

10 Seznam příloh

Elektronické přílohy ve formátu archivu .zip jsou:

- openGA-1.0.5.zip – stažený balík knihovny OpenGA
- opennn-5.0.5.zip – stažený balík knihovny OpenNN
- airfoil_self_noise.zip – aplikace demonstrující využití knihovny OpenNN – predikce hluku (projektový soubor, zdrojový kód, přeložená aplikace včetně knihoven DLL, vstupní data a ukázka výstupu programu)
- GA_NN.zip – aplikace učící neuronovou síť genetickým algoritmem (projektový soubor, zdrojový kód, přeložená aplikace včetně knihoven DLL, vstupní data a výstup programu z deseti spuštění)
- libopennn.zip – přeložená knihovna OpenNN (ve verzi release – libopennn.a)
- log_problem.zip – aplikace demonstrující využití knihovny OpenGA – řešení logického problému (projektový soubor, zdrojová kód, přeložená aplikace včetně knihoven DLL, ukázka výstupu programu)