VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

AUTOMATIZOVANÉ VYTVÁŘENÍ REPREZENTACE PRO KARTÉZSKÉ GENETICKÉ PROGRAMOVÁNÍ POMOCÍ NEURONOVÝCH SÍTÍ

AUTOMATED REPRESENTATION LEARNING FOR CARTESIAN GENETIC PROGRAMMING USING NEURAL NETWORKS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. MARTIN KOČI**
AUTHOR

VEDOUCÍ PRÁCE **prof. Ing. LUKÁŠ SEKANINA, Ph.D.**
SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání diplomové práce

154232

Ústav:	Ústav počítačových systémů (UPSY)
Student:	**Koči Martin, Bc.**
Program:	Informační technologie a umělá inteligence
Specializace:	Strojové učení
Název:	**Automatizované vytváření reprezentace pro kartézské genetické programování pomocí neuronových sítí**
Kategorie:	Umělá inteligence
Akademický rok:	2023/24

Zadání:

1. Zpracujte studii o automatizovaném vytváření reprezentace problému v oblasti strojového učení. Zaměřte se na reprezentace generované pomocí hlubokých neuronových sítí a jejich využití v kontextu genetického programování.
2. Navrhněte metodu pro automatizované vytvoření reprezentace pro kartézské genetické programování (CGP), která využije vhodnou neuronovou síť. Navrhněte alespoň dvě použití takto vytvořené reprezentace.
3. Vygenerujte nebo jinak získejte vhodnou datovou sadu pro automatizovaný návrh reprezentace pomocí neuronové sítě.
4. Implementujte metodu navrženou v bodu 2. Můžete využít existující knihovny pro práci s neuronovými sítěmi, genetickým programováním a pro zpracování dat.
5. S využitím vytvořeného software a datové sady získané v bodu 3 automatizovaně navrhněte reprezentaci, popř. různé reprezentace, které budou využitelné v oblasti CGP.
6. Ověřte funkčnost a chování vytvořené reprezentace ve zvolených úlohách.
7. Zhodnoťte dosažené výsledky a diskutujte možnosti pokračování projektu.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Sekanina Lukáš, prof. Ing., Ph.D.**
Vedoucí ústavu:	Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání:	1.11.2023
Termín pro odevzdání:	17.5.2024
Datum schválení:	30.10.2023

Abstrakt

Táto diplomová práca sa zaoberá prepojením neurónových sietí a kartézskeho genetického programovania (CGP). Skúma využitie neurónových sietí na automatické vytváranie reprezentácie pre CGP a ich využitie pre vylepšenie evolučného procesu v CGP. Štúdia pokrýva základné koncepty strojového učenia, vrátane rôznych typov učenia a modelov neurónových sietí. Ďalej sa dotýka evolučných algoritmov s dôrazom na ich základné princípy, všeobecné algoritmy a typy reprezentácií. Táto práca tiež zahŕňa princípy učenia reprezentácii a dve základné architektúry pre ich tvorbu. Popisuje aj následné využitie učenia reprezentácií v genetickom programovaní. Návrh riešenia zahŕňa získavanie a predspracovanie dát, procesy tvorby reprezentácií a využitie výsledných reprezentácií. Práca taktiež implementuje dva nové prístupy pre vytváranie reprezentácii kartézskych genetických programov. Ďalej skúma ich využitie v dvoch nových mutačných operátoroch, kde jeden je založený na priamej úprave vektorovej reprezentácie a druhý na výbere génov pre mutáciu na základe ich podobnosti. Posledná zo skúmaných oblastí je predikovanie vhodnosti kandidátnych riešení za použitia de novo vzniknutých reprezentácií.

Abstract

This master's thesis addresses the integration of neural networks and Cartesian Genetic Programming (CGP). It explores the use of neural networks for automated representation creation for CGP and their application to improve the evolutionary process in CGP. The study covers basic concepts of machine learning, including various types of learning and neural network models. It also touches on evolutionary algorithms with an emphasis on their basic principles, general algorithms, and types of representations. This work also includes principles of representation learning and two fundamental architectures for their creation. It describes the subsequent use of representation learning in genetic programming. The solution design includes data acquisition and preprocessing, representation creation processes, and the utilization of the resulting representations. The thesis also implements two new approaches for creating representations for Cartesian genetic programs. It further explores their use in two new mutation operators, where one is based on direct modification of the vector representation and the other on the selection of genes for mutation based on their similarity. The last of the explored areas is predicting the suitability of candidate solutions using newly emerged representations.

Kľúčové slová

evolučné algoritmy, genetické programovanie, neurónové siete, konvolučné neuronové siete, transformery, viacvrstvový perceptron, perceptron, grafové neuronové siete, kartézske genetické programovanie, učenie reprezentácie, strojové učenie

Keywords

evolutionary algorithms, genetic programming, cartesian genetic programming, neural networks, transformers, convolutial neural networks, graph neural networks, multilayer perceptron, perceptron, representation learning

Citácia

KOČI, Martin. *Automatizované vytváření reprezentace pro kartézské genetické programování pomocí neuronových sítí.* Brno, 2024. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Lukáš Sekanina, Ph.D.

Automatizované vytváření reprezentace pro kartézské genetické programování pomocí neuronových sítí

Prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne pod vedením pána prof. Ing. Lukáša Sekaniny, Ph.D. a uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Martin Koči
17. mája 2024

Poďakovanie

Rád by som poďakoval pánovi prof. Ing. Lukášovi Sekaninovi, Ph.D. za jeho odborné vedenie, cenné rady a konzultácie k vypracovaniu tejto diplomovej práce.

Obsah

Zoznam obrázkov

Kapitola 1

Úvod

Strojové učenie za posledné dve dekády dokázalo veľké pokroky v oblastiach ako sú spracovanie prirodzeného jazyka, rozpoznávanie jazyka a počítačového videnia. V každej z týchto oblastí sa používajú modely strojového učenia, ktoré sú výpočtovo náročné. Na svete existuje veľa problémov, ktoré si túto výpočtovú náročnosť nemôžu dovoliť. Tu prichádzajú algoritmy a metódy inšpirované prírodou. Kartézske genetické programovanie je jedným z mnohých algoritmov, ktoré sú inšpirované prírodou. Táto metóda sa odlišuje svojou jedinečnou reprezentáciou programov, ktorá umožňuje efektívne evolučné prehľadávanie kandidátnych riešení. Vďaka svojej schopnosti efektívne riešiť zložité optimalizačné problémy sa stalo populárnou voľbou pri inžinierskych aj výskumných projektoch.

Táto práca sa venuje výskumu využitia hlbokých neurónových sietí pre vytváranie reprezentácií pre kartézske genetické programy a potenciálne vytvorí priestor pre ďalší výskum v tejto oblasti. De novo vzniknuté reprezentácie môžu mať mnoho aplikácii. Táto práca sa primárne zameriava na využitie v mutačných operátoroch a predikcii fitness kandidátnych riešení.

V kapitole 2 sú popísané základné typy učenia a rôzne druhy neurónových sietí. Kapitola 3 oboznamuje o základných princípoch a metódach v evolučných algoritmoch. Špecifickejšie je popísané v kapitole 4 genetické programovanie, kde sa táto kapitola zameriava na oboznámenie so stromovým a kartézskym genetickým programovaním. Základné princípy učenia reprezentácii sú popísané v kapitole 5. V kapitole 6 sa nachádza návrh riešenia, ktorý popisuje získanie a predspracovanie dát, proces vytvárania nových reprezentácii a navrhuje ich možné použitie. Kapitola 7 popisuje implementáciu návrhu riešenia. V kapitole 8 sú vykonané experimenty na zvolených úlohách a kapitola 9 diskutuje o výsledkoch a možnom pokračovaní projektu.

Kapitola 2

Strojové učenie

Strojové učenie je odvetvie počítačových vied, ktoré sa zaoberá vytváraním algoritmov na optimalizáciu kritéria výkonnosti použitím trénovacích dát alebo predchádzajúcich skúseností [2]. História strojového učenia siaha až do polovice 20. storočia, kedy sa rozvíjali prvé teórie a modely, ktoré položili základy pre dnešnú umelú inteligenciu. Alan Turing [62] a ďalší preskúmali koncepty, ktoré položili základy pre dnešné algoritmy. V 70. a 80. rokoch došlo k rozvoju teórie neurónových sietí [55], ktoré sú dnes základom mnohých pokročilých aplikácií strojového učenia.

V strojovom učení máme model, ktorý je definovaný parametrami a učenie je proces, ktorý ich optimalizuje pomocou trénovacích dát alebo predchádzajúcich skúseností. Model môže byť prediktívny (čím môže vytvoriť predikcie do budúcnosti), deskriptívny (aby získal znalosti z dát) alebo môže niesť prvky oboch. Rola počítačových vied v strojovom učení spočíva v tom, že pri trénovaní potrebujeme efektívne algoritmy, ktoré dokážu efektívne vyriešiť optimalizačný problém a taktiež efektívne ukladať a spracovávať rozsiahle dáta. Po trénovaní musia byť reprezentácia a algoritmus pre inferenciu tiež efektívni [2].

Vývoj strojového učenia bol významne ovplyvnený zvýšením výpočtovej kapacity a dostupnosti veľkého množstva dát v posledných desaťročiach, čo umožnilo vytvorenie sofistikovanejších a presnejších modelov.

2.1 Typy učenia

V strojovom učení rozlišujeme štyri základné druhy učenia: učenie s učiteľom, učenie bez učiteľa, učenie posilňovaním a kombinácia učenia s a bez učiteľa.

Učenie s učiteľom

Učenie s učiteľom (anglicky – supervised learning) je typ učenia, pri ktorom sa model učí z dátovej sady pozostávajúcej zo vstupných a príslušných výstupných dvojíc. Úlohou pri učení s učiteľom je naučiť model mapovať vstup na odpovedajúci výstup. Vstup reprezentuje vektor príznakov. Každá dimenzia vektora príznakov reprezentuje jeden príznak, ktorý popisuje daný vstup. Výstup, tzv. označenie vstupného vektora, môže nadobúdať hodnoty z konečnej množiny tried, reálne číslo alebo komplexnejšiu štruktúru ako je vektor, matica, strom alebo graf. Napríklad v prípade klasifikácie e-mailov sa model učí rozlišovať, že čo je a nie je spam na základe vlastností obsahu e-mailu [9].

Učenie bez učiteľa

Učenie bez učiteľa (anglicky – unsupervised learning) je učenie, pri ktorom dátové sady neobsahujú žiadne označenia výstupov. To nastáva v prípadoch, keď pre označenie vstupov potrebujeme znalosti z danej oblasti, ktoré nám chýbajú alebo by to pri rozsiahlych dátach vyžadovalo veľa manuálnej práce. Preto potrebujeme vedieť vybrať vhodné vektory príznakov, ktoré dané vstupy budú dobre popisovať. Cieľom je nájsť skryté štruktúry alebo vzorce v neoznačených dátach. Typickým príkladom sú metódy zhlukovania, ktoré organizujú dáta do skupín na základe podobnosti, alebo metódy na zníženie dimenzií, ktoré zjednodušujú dáta pre lepšie spracovanie a vizualizáciu. Napríklad, pri analýze zákazníckych dát môže byť cieľom identifikovať rôzne segmenty zákazníkov na základe ich nákupného správania [9].

Posilňované učenie

Učenie posilňovaním (anglicky – reinforcment learning) je odvetvie strojového učenia, kde agent je integrovaný v prostredí. Agent vníma stavy prostredia ako vektory príznakov a v každom stave vykonáva akcie, ktoré môžu viesť k odmenám alebo trestom. Cieľom posilňovaného učenia je vytvoriť taktiku, ktorá bude maximalizovať kumulatívnu odmenu. Taktika je často reprezentovaná ako funkcia, ktorá má ako vstup vektor príznakov popisujúci aktuálny stav prostredia a výstup je akcia, ktorá sa má vykonať. Tento prístup nachádza uplatnenie napríklad pri trénovaní modelov pre hry alebo autonómne autá [61].

Kombinácia učenia s a bez učiteľa

V kombinácii učenia s a bez učiteľa (anglicky – semi–supervised learning) obsahuje dátová sada označené aj neoznačené vstupy. Tento prístup sa používa v situáciach, keď je označených vstupov veľmi málo a neoznačených príliš mnoho. Cieľom tohto učenia je využiť neoznačené dáta pre lepšie porozumenie štruktúry a distribúcie dát, čím zlepšíme presnosť pri klasifikačných alebo regresných úlohách [9].

2.2 Neurón

Základnou jednotkou ľudského mozgu je neurón. Malá časť mozgu, ktorá je veľká asi ako jedno zrnko ryže obsahuje vyše 10000 neurónov, ktoré majú priemerne 6000 prepojení s inými neurónmi [53].

Neurón je optimalizovaný na prijímanie , spracovanie a odosielanie informácií. Neurón prijíma informácie prostredníctvom dendritov. Každé prepojenie dynamicky mení svoju silu prepojenia na základe toho ako často sa používa. Sila tohto prepojenia určuje príspevok daného vstupu na výstup daného neurónu. Po prijatí všetkých vážených vstupov sú v jadre neurónu sčítané a transformované na nový signál a poslané po axóne do ostatných neurónov. Tento proces je základom pre vývoj umelých neurónov, ktoré napodobňujú biologickú štruktúru neurónu. Prvý model umelého neurónu bol navrhnutý v roku 1943 [40] a obsahuje vstupný vektor $x = [x_1, x_2, \ldots, x_n]$, vektor váh $w = [w_1, w_2, \ldots, w_n]$, bias b a transformačnú funkciu f, kde f je zvyčajne nelineárná funkcia. Výstup $y = f(\mathbf{x} \cdot \mathbf{w} + b)$ môže byť následne poslaný ako vstup do ďalších neurónov [8].

2.3 Perceptrón

Perceptrón predstavuje jeden z najzákladnejších umelých neurónov. Bol to jeden z prvých modelov strojového učenia predstavený v roku 1962 [54], navrhnutý ako algoritmus pre binárnu klasifikáciu.

Štruktúra perceptrónu je veľmi podobná základnému modelu umelého neurónu ako bol opísaný McCullochom a Pittsom [40]. Rozdiel spočíva v tom, že transformačná funkcia f je skoková funkcia. Keď táto funkcia prekročí určitý prah, tak perceptrón produkuje hodnotu 0 alebo 1, a to reprezentuje dve triedy. Výstup perceptrónu sa spočíta nasledovne:

$$y = f(\mathbf{x} \cdot \mathbf{y} + b) \tag{2.1}$$

Kde f je:

$$f(z) = \begin{cases} 1 & z > 0 \\ 0 & z \leq 0 \end{cases} \tag{2.2}$$

Perceptrón bol významným krokom v oblasti strojového učenia a položil základy pre vývoj komplexnejších modelov neurónových sietí. Jeho jednoduchosť a efektívnosť v určitých úlohách z neho robia dôležitý model aj v súčastnosti. Napriek tomu sú jeho schopnosti obmedzené hlavne kvôli tomu, že nedokáže riešiť problémy, ktoré nie sú lineárne separovateľné.

2.4 Viacvrstvový perceptrón

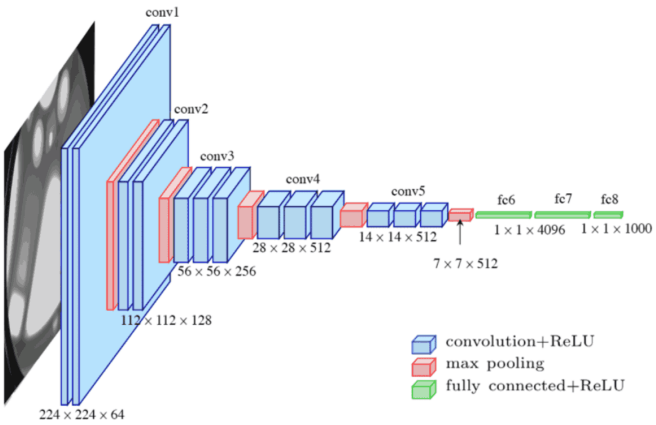
Viacvrstvový perceptrón (anglicky – multilayer perceptron, MLP) predstavuje rozšírenie základného modelu perceptrónu do architektúry s viacerými vrstvami. Na rozdiel od perceptrónu, ktorý dokáže riešiť len lineárne separovateľné problémy, viacvrstvový perceptrón túto vlastnosť prekonáva a dokáže riešiť aj komplexnejšie úlohy [2].

Viacvrstvový perceptrón sa skladá z troch hlavných typov vrstiev: vstupnej vrstvy, jednej alebo viacerých skrytých vrstiev a z výstupnej vrstvy. Neuróny v prvej vrstve sú plne prepojené s neurónmi v nasledujúcej vrstve. Neuróny nemajú medzi sebou žiadne spätné väzby ani cykly. Jedná sa o dopredný acyklický graf. Oproti perceptrónu v ktorom neurón vykonával skokovú aktivačnú funkciu, viacvrstvový perceptrón využíva nelineárne aktivačné funkcie (napr. sigmoidu, ReLU) čo umožňuje učenie nelineárnych vzťahov [24].

Učenie viacvrstvového perceptrónu prebieha pomocou algoritmu spätného šírenia chyby. Tento algoritmus umožňuje upravovanie váh na základe chyby medzi očakávaným výstupom a skutočným výstupom. Spätné šírenie chyby využíva metódu gradientového zostupu na minimalizovanie chybovej funkcie, čím sa postupne upravujú váhy v sieti aby dosiahla lepšie výsledky [55].

2.5 Konvolučné neurónové siete

Konvolučné neurónové siete (anglicky – convolutional neural networks, CNNs) sú jedny z najpoužívanejších neurónových sieti v oblasti spracovania obrázkov a videa. Na rozdiel od klasických dopredných neurónových sietí, ako sú napríklad viacvrstvový perceptrón, ktoré pracujú s dátami o rozmeroch jednej dimenzie, CNNs pracujú s dátami v ich plných rozmeroch. Preto sa prirodzene hodia na spracovanie obrazov a videa. Aplikácie CNNs sú napríklad detekcia objektov, identifikácia objektov, segmentácia a mnoho ďalších [1]. Príklad architektúry CNN je zobrazený na obr. 2.1.



Obr. 2.1: Príklad architektúry konvolučnej neurónovej siete (VGGNet). Prevzaté z [37].

Konvolučný operátor

Základom CNNs je konvolučný operátor, ktorý sa nachádza v konvolučných vrstvách. Ako pri bežných neurónových sieťach, akými je napríklad viacvrstvový perceptrón, ktorý má svoje učiace parametre (váhy) siete na jednotlivých hranách, ktoré spájajú neuróny, v CNNs sa tieto učiace parametre nachádzajú vo filtroch, ktoré reprezentujú konvolučný operátor. Tieto filtre prechádzajú vstupný obrázok a zachytávajú dôležité príznaky z daného vstupu. V priebehu učenia CNNs sa tieto váhy vo filtroch optimalizujú. Výstup konvolučnej vrstvy ovplyvňujú parametre ako sú výplň, posun filtra a veľkosť filtra [1].

Veľkosť posunu, výplne a filtra

Posun hrá veľkú rolu pri konvolučnom operátore. Riadi rýchlosť posunu filtra po vstupe. Väčšinou sa nastavuje na hodnotu jedna a to znamená, že sa filter vždy posunie o jeden pixel. Jeden z ďalších veľkých faktorov má výplň, ktorá sa dáva okolo vstupu. Väčšinou sa jedná o nulovú výplň. Posledná je veľkosť filtra tá sa spravidla pre konvolučnú vrstvu určuje buď 3×3 alebo 5×5. Všetky tieto parametre majú veľký vplyv na učenie a na výstup konvolučnej vrstvy. Posun a veľkosť filtra zmenšujú výstup z konvolučnej vrstvy a výplň ho zväčšuje. To môže zaniesť určitý druh šumu do výstupu [1].

Zoskupovacie vrstvy

Zoskupovacie vrstvy znižujú priestorové rozmery (šírku a výšku) vstupu pre nasledujúce vrstvy siete. Tento proces zaistí zníženie výpočtovej náročnosti a zároveň zníži aj parametre modelu. Najčastejšie sa používajú dva druhy zoskupovacích vrstiev: priemerné zoskupovanie a zoskupovanie maxima. V zoskupovacej vrstve sa nachádza tiež filter väčšinou o veľkosti 2×2, ktorý zníži priestorové rozmery o polovicu. Priemerné zoskupovanie spriemeruje hodnoty, ktoré sa nachádzajú v okne daného filtra a zoskupovanie podľa maximálnej hodnoty zase vyberá maximálnu hodnotu z okna filtra. [1]

2.6 Grafové neurónové siete

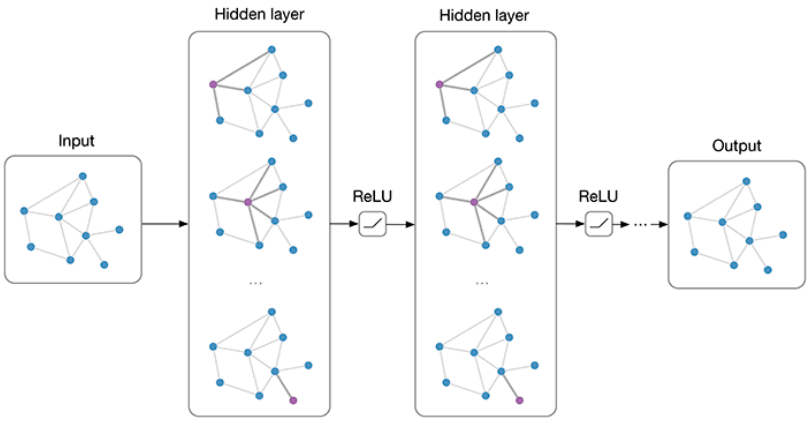
Grafové neurónové siete (anglicky – graph neural networks, GNNs) sú významným pokrokom v odbore strojového učenia. Veľkou prednosťou GNNs je ich schopnosť pracovať

s dátami, ktoré sú prirodzene štrukturované ako grafy. Grafy môžeme nájsť všade okolo nás. Napríklad v biologických dátach, sociálnych sieťach, v doprave a v ďalších. Rôzne druhy dát sa dajú aj previesť do grafového formátu (napr. text, obrázky).

GNNs boli prvý krát predstavené v roku 2008 [57]. Motiváciou k vytvoreniu GNNs bolo hlavne to, že tradičné neuronové siete, ako sú napríklad konvolučné neuronové siete, dokážu veľmi dobre pracovať s textom alebo obrázkami, ale veľmi ťažko dokážu pracovať s dátami, ktoré nie sú Euklidovské. Euklidovské sú napríklad grafy.

Z toho vyplýva, že GNNs sú vytvorené na to, aby pracovali s dátami, ktoré sú štrukturované ako grafy. Pracujú na úrovni uzlov a hrán. Reprezentujú objekty a vzťahy medzi nimi. Ich prednosťou je učiť sa z grafovej topológie a príznakov, ktoré v sebe majú uložené hrany a uzly. Príklad architektúry GNNs je zobrazený na obr. 2.2.

Aplikácie GNNs sú napríklad v predpovedaní dopravy, doručovacích systémoch a mnohých ďalších. Táto sekcia bola čerpaná z [33] a [56].



Obr. 2.2: Príklad architektúry grafových neurónových sietí. Prevzaté z [33]

Reprezentácia uzlov a hrán

Ako bolo spomenuté na začiatku tejto sekcie, dáta v GNNs sú reprezentované ako grafy, ktoré sa skladajú z uzlov a hrán. Uzle reprezentujú určité entity a hrany reprezentujú vzťahy medzi nimi. Uzle sú väčšinou reprezentované pomocou vektoru príznakov, ktoré ich charakterizujú a hrany môžu byť reprezentované povahou alebo silou vzťahu medzi uzlami ktoré spájajú.

Konvolúcia nad grafmi

Konvolúcia nad grafmi je základným operátorom v GNNs. Tento operátor pochádza z klasických konvolučných neuronových sietí, ale je adekvátne upravený aby dokázal pracovať nad grafmi.

Princíp konvolúcie nad grafmi

Konvolúcia nad grafmi prebieha tak, že uzol v sieti dostáva informácie o svojom okolí pomocou agregácie a transformácie príznakov zo susedných uzlov. Týmto spôsobom sa model dokáže naučiť lokálne reprezentácie a po niekoľkých iteráciách sa dokáže model naučiť aj globálne reprezentácie po tom, ako si všetky uzle medzi sebou poprenášajú svoje lokálne informácie.

Zoskupovanie v grafoch

Zoskupovanie v grafoch je veľmi podobné ako zoskupovanie v konvolučných neuronových sieťach. Redukuje dimenzionalitu príznakov v uzloch a pomáha pri získavaní vyšších úrovní príznakov z grafu. Existuje viacero druhov zoskupovania. Príkladom môže byť globálne zoskupenie, kde sa agregujú príznaky z celého grafu alebo hierarchické zoskupovanie, kde sa naopak zoskupujú uzle a ich príznaky.

2.7 Transformátory

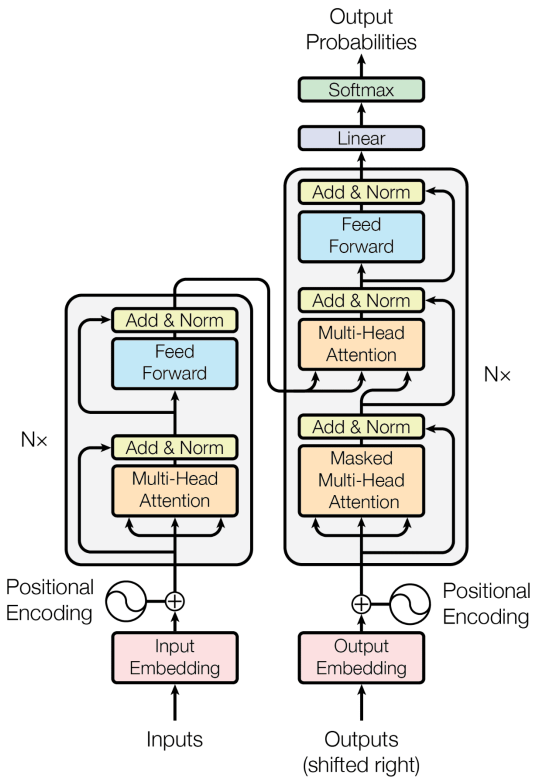
Transformátory sú jednou z revolúcií v strojovom učení, ktoré v podstate predefinovali spôsob akým sa spracúvajú sekvenčné dáta v prirodzenom jazyku. Na rozdiel od ostatných modelov, ktoré spracúvajú sekvenčné dáta, tak transformátorová architektúra ponúka paralelné spracovanie, čo výrazne zvyšuje efektivitu a porozumenie prirodzeného jazyka.

Z historického konceptu transformátory vznikli ako odozva na limitácie predchádzajúcich typov rekurentných neurónových sietí, akými boli klasické rekurentné neurónové siete, LSTM [28] alebo GRU [12]. Napriek tomu, že LSTM a GRU riešili problém miznúceho gradientu v hlbších vrstvách siete, aj tak mali problémy zachytávať dlhodobé závislosti v dátach. To je veľmi podstatné pri snažení sa pochopiť kontextu v jazyku.

Transformátory vznikli v roku 2017 [65] a odchýlili sa od tradičných sietí ako sú RNN, LSTM, GRU a spoliehali sa výhradne na mechanizmus pozornosti (anglicky – attention mechanism). Mechanizmus pozornosti modelom umožňuje zameriavať sa na rozdielne časti vstupnej sekvencie pri každej predikcii. To umožnilo zachytiť dlhodobé závislosti a komplexné vzťahy v dátach bez toho, aby boli obmedzované sekvenčným spracovaním.

Limitácia transformátorov je cena trénovania a nutnosť mať veľkú dátovú sadu. Preto v horizonte nasledujúcich rokov od ich vzniku došlo k rozvoju mnohých modelov ako sú napríklad BERT, GPT a mnoho ďalších. Tie sú predtrénované na veľkých dátových sadách a často stačí málo času a dát na dotrénovanie takého modelu pre konkrétnu doménu. Príklad architektúry transformátoru je zobrazený na obr. 2.3.

Aplikácia transformátorov je napríklad v generovaní, prekladaní textu, sentimentálnej analýze a mnohých ďalších aplikáciach. Jedny z najznámejších variantov transformátorov sú napríklad GPT [52] a BERT [15].



Obr. 2.3: Architektúra transformátoru. Prevzaté z [65].

Self–attention

Jedná sa o inováciu v mechanizme pozornosti. Tento nový mechanizmus dovolil porovnávať každý element danej sekvencie s každým iným elementom v sekvencii, aby sa zistilo, aký majú tieto elementy na seba vzájomný vplyv [65].

Multi–head attention

Jedná sa o rozšírenie self–attention mechanizmu o paralelizáciu. Princíp spočíva v spustení viacerých self–attention mechanizmov nad rôznymi časťami sekvencie. To výrazne zvýšilo možnosti modelu naučiť sa komplexnejšie štruktúry v dátach a lepšiu generalizáciu naprieč úlohami [65].

Architektúra

Architektúra sa skladá z dvoch hlavných komponent: kodéru a dekodéru.

Kodér

Kodér spracováva vstupnú reprezentáciu a generuje reprezentáciu, ktorá zachytáva najdôležitejšie informácie/príznaky z danej sekvencie. Kodér obsahuje viac druhov vrstiev, sú to multi–head self–attention a plne prepojená neurónová sieť. Medzi týmito vrstvami sa môže ešte nachádzať normalizácia a prípadne reziduálne prepojenia pre stabilizáciu trénovania a efektivitu [65].

Dekodér

Dekodér používa výstup z kodéru pre generovanie výstupnej sekvencie. Dekodér obsahuje rovnaké vrstvy ako aj kodér s tým, že obsahuje daľšie samostatné vrstvy pre multi–head self–attention výhradne pre výstup kodéru a ešte obsahuje maskovanú multi–head self–attention vrstvu, ktorá v zásade funguje rovnako ako bežná vrstva s tým rozdielom, že sú zamaskované tie vstupné údaje, ktoré by mali vplyv na generovanie aktuálneho výstupu dekodéru na základe pozornosti [65].

Pozičné kódovanie

Keďže transformátory spracovávajú dáta ako sekvencie, tak im chýba informácia o pozícii jednotlivých elementov v sekvencii. Preto vzniklo pozičné kódovanie, kde pri trénovaní modelu sa vždy do každého elementu v sekvencii zakóduje pozícia daného elementu. Toto kódovanie zaistí, aby model bral do úvahy poradie jednotlivých slov vo vstupe. Tento mechanizmus je veľmi dôležitý pre porozumenie daného jazyka [65].

Kapitola 3

Evolučné algoritmy

Evolučné algoritmy sú špeciálne prípady prehľadávacích algoritmov, ktoré sa snažia pomocou operácií inšpirovaných prírodou prehľadávať stavový priestor kandidátnych riešení zložitých problémov a nájsť riešenie, ktoré je optimálne alebo sa k nemu blíži. Často sa používajú tam, kde by konvenčným algoritmom trvalo niekoľko rokov než by našli optimálne riešenie [38].

Za posledných 65 rokov, od kedy sa v odbornej literatúre pravdepodobne prvýkrát objavil pojem evolučné algoritmy [6], vzniklo veľa variácií evolučných algoritmov. Medzi tieto variácie patria napr. evolučné stratégie, evolučné programovanie, genetické algoritmy, genetické programovanie, kartézske genetické programovanie a mnoho ďalších [14].

V tejto kapitole sa budeme ďalej zaoberať už len základnými princípmi evolučných algoritmov a následne v kapitole 4 je popísané genetické programovanie a kartézske genetické programovanie.

3.1 Základné pojmy

Táto podkapitola obsahuje základné pojmy, ktoré sa často používajú v evolučných algoritmoch a sú adoptované z biológie. Vzhľadom na to, že sa tieto pojmy vyskytujú ďalej v tejto práci je vhodné im porozumieť [34].

- **Gén:** Je to základný element chromozómu. Nadobúda hodnotu z konečnej množiny definovaných hodnôt.
- **Chromozóm, jedinec, genotyp:** Je to zoskupenie génov, ktoré reprezentuje jeden stav v stavovom priestore. Tieto pojmy sú medzi sebou navzájom zamieňané v tejto práci.
- **Fenotyp, kandidátne riešenie:** Potenciálne riešenie daného problému.
- **Populácia:** Je to štruktúra jedincov.
- **Generácia:** Jedná sa o jednu iteráciu evolučného algoritmu.

3.2 Všeobecný algoritmus

Ako bolo spomenuté na začiatku tejto kapitoly, evolučné algoritmy sú inšpirované prírodou. To znamená, že v evolučných algoritmoch prebieha evolučný proces (tzv. evolúcia),

pri ktorom je určitý počet jedincov vystavovaný selekčnému tlaku, na základe ktorého sa dynamicky menia a vyberajú tí najlepší jedinci pre konkrétny problém. Dalo by sa povedať, že sa jedná o algoritmus prirodzeného výberu jedincov, ktorý prebieha nasledovne:

- Počas priebehu algoritmu je udržiavaná populácia jedincov, ktorá reprezentuje potenciálne kandidátne riešenie daného problému.
- Jedinci majú priradenú hodnotu fitness, ktorá určuje kvalitu daného kandidátneho riešenia pri riešení problému.
- Na základe zvoleného selekčného mechanizmu a fitness sú zvolení rodičia, ktorí pomocou genetických operátorov vytvoria potomkov.
- Ako posledný krok sa vyberajú jedinci do novej populácie. Výber jedincov z rodičov a potomkov prebieha už podľa zvoleného evolučného algoritmu.

Ak sa tento proces opakuje určitý počet generácií, tak hodnota fitness celej populácie sa pomaly zvyšuje. Na základe tohto pozorovania vieme zostaviť jednoduchý všeobecný Algoritmus 3.1 [14].

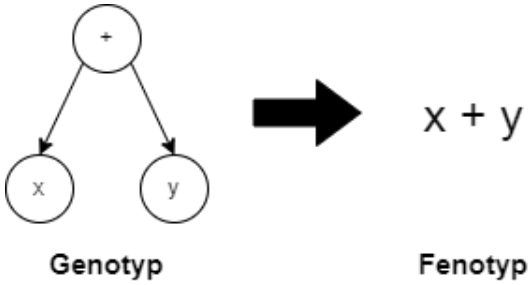
Algoritmus 3.1 Základný Evolučný Algoritmus

```
1: Inicializuj populáciu náhodnými jedincami
2: Urči fitness každého jedinca
3: while nebolo dosiahnuté kritérium ukončenia do
4:     Výber rodičov z populácie
5:     Kríženie rodičov pre vytvorenie potomkov
6:     Mutácia potomkov s určitou pravdepodobnosťou
7:     Urči fitness potomkov
8:     Výber jedincov pre ďalšiu generáciu z rodičov a potomkov
9: end while
```

Jednotlivé časti algoritmu sú dôkladnejšie popísané v nasledujúcich podkapitolách.

3.3 Reprezentácia

Výber internej reprezentácie jedinca spadá do dvoch biológiou inšpirovaných kategórií. Jedná sa o reprezentáciu fenotypom alebo genotypom. Pri reprezentácii genotypom sa kandidátne riešenia interne zakódujú. Typicky sa jedná o reťazec písmen, číslic alebo binárnych hodnôt. Z pohľadu biológie je podobnosť s DNA. Naopak, reprezentácia fenotypom reprezentuje dané kandidátne riešenie priamo bez interného kódovania. Typicky sa môže jednať o neurónovú sieť, matematický výraz, súradnice atď [14]. Napríklad pri genetickom programovaní reprezentuje kandidátne riešenie strom (genotyp), ktorým je zakódovaný matematický výraz (fenotyp), viz. obr. 3.1.



Obr. 3.1: Príklad genotypu a fenotypu. V tomto prípade nám genotyp stromovou reprezentáciou reprezentuje aritmetický výraz.

3.4 Ohodnocovanie kandidátneho riešenia

Pri ohodnocovaní kandidátnych riešení sa používajú dve funkcie – Účelová funkcia (anglicky – objective function) a fitness funkcia. Účelová funkcia určuje kvalitu kandidátneho riešenia pre problém, ktorý sa snažíme vyriešiť. Fitness funkcia určuje ako je dané kandidátne riešenie vhodné v zmysle prežitia a reprodukcie v rámci evolučného algoritmu. Účelová funkcia je to, čo sa snažíme optimalizovať a fitness funkcia riadi proces optimalizácie [17].

Fitness funkcia sa môže maximalizovať alebo minimalizovať v závislosti od špecifikácie algoritmu a charakteru problému. Aj keď je často prirodzenejšie a zvyčajnejšie maximalizovať fitness funkciu (v súlade s princípom prežitia najvhodnejších), je dôležité zdôrazniť, že voľba medzi maximalizáciou a minimalizáciou závisí od konkrétnej aplikácie a nastavenia algoritmu [17].

Pri úlohe obchodného cestujúceho by mohla byť účelová funkcia reprezentovaná ako vzdialenosť najkratšej cesty. Menšia vzdialenosť naznačuje lepšie kandidátne riešenie. Pri fitness funkcii môžeme túto hodnotu transformovať tak, aby sa získali hodnoty vo vhodnom rozsahu pre evolučný algoritmus. Tento rozsah môže byť napríklad v intervale [0, 1], kde väčšia hodnota znamená lepšie kandidátne riešenie.

3.5 Selekcia

Selekcia je jedna z najdôležitejších častí evolučného algoritmu. Využíva sa vo dvoch prípadoch, a to pri výbere rodičov a pri výbere jedincov, ktorí prežijú do ďalšej generácie. To, či sa bude v druhom prípade vyberať len z potomkov alebo aj rodičov závisí už od konkrétneho prístupu. Pri oboch sa selekcia môže riadiť pomocou fitness alebo aj vekom populácie. Hodnota fitness značí to, ako sú jednotliví jedinci vhodní pre reprodukciu [17].

V tejto práci sa predpokladá, že hodnota fitness sa maximalizuje a má vždy pozitívne hodnoty. Objektívna funkcia sa síce môže podľa zvoleného problému a reprezentácie minimalizovať, ale výstup z objektívnej funkcie sa dá fitness funkciou vhodne transformovať tak, aby sa hodnota fitness maximalizovala.

Výber rodičov

Nasleduje niekoľko vybraných metód pre výber rodičov.

Proporcionálna selekcia založená na fitness

Proporcionálna selekcia založená na fitness (anglicky – Fitness Proportional Selection) bola prvýkrát predstavená Holland [29] v roku 1975. Od vtedy bola a je predmetom výskumu. Výber jedinca k reprodukcii je na základe pravdepodobnosti, ktorá sa vypočíta ako absolútna hodnota fitness ku absolútnej hodnote fitness celej populácie. Čiže pravdepodobnosť pre i-tého jedinca by sa vypočítala ako $P(i) = \frac{f_i}{\sum_{j=1}^{\mu} f_j}$. Táto metóda má mnoho nevýhod. Jednou z nich je rýchla konvergencia. Druhá je, že v neskorších generáciách, keď rozdiely medzi hodnotami fitness sú minimálne, nedochádza takmer k žiadnemu selekčnému tlaku. Tieto nevýhody sú väčšinou riešené pomocou posuvného okna (anglicky – windowing) [17].

Selekcia na základe poradia

Selekcia na základe poradia (anglicky – Ranking Selection) je metóda, ktorá bola inšpirovaná pozorovaním nevýhod pri proporcionálnej selekcii založenej na fitness [3]. Táto metóda zachováva konštantný selekčný tlak naprieč generáciami. Ako prvý krok zoradí jedincov na základe fitness od najhoršieho po najlepšieho jedinca. Následne je ku jednotlivým jedincom priradená pravdepodobnosť výberu na základe ich poradia podľa rovnice:

$$P(i) = \frac{(2-s)}{\mu} + \frac{2i(s-1)}{\mu(\mu-1)} \tag{3.1}$$

Kde i je poradie daného jedinca, μ je veľkosť populácie a parameter s $(1 < s \leq 2)$ môže byť interpretovaný ako predpokladaný počet potomkov [17].

Selekcia turnajom

Predchádzajúce dve selekčné metódy sa spoliehajú na znalosť celej populácie. Táto možnosť môže byť v niektorých prípadoch nemožná alebo veľmi drahá. To môže nastať, ak máme veľkú populáciu alebo keď je populácia rozdistribuovaná na nejakom paralelnom systéme. Taktiež sa predpokladá, že fitness jedincov je kvantifikovateľná, (podľa nejakej účelovej funkcie, ktorú sa snažíme optimalizovať) čo nemusí byť vždy pravda.

Selekcia turnajom je metóda, ktorá má výhodu, že nepotrebuje mať znalosť celej populácie a ani kvantifikovateľnú hodnotu fitness. V jednoduchosti, zameriava sa na porovnanie k jedincov za účelom určenia ich poradia. Selekcia turnajom vyberie z množiny μ jedincov λ rodičov pre reprodukciu a to Algoritmom 3.2 [17].

Algoritmus 3.2 Selekcia turnajom

Vstup: Populácia P o veľkosti μ
Vstup: Veľkosť výberu λ
Vstup: Veľkosť turnaja k
Výstup: Skupina rodičov R veľkosti λ

```
c ← 1
while c ≤ λ do
    Vyber podmnožinu S ⊂ P veľkosti k náhodne
    i ← jedinec v S s najvyššou fitness
    Pridaj i do R
    c ← c + 1
end while
```

Výber novej populácie

Pri výbere novej populácie existujú dva modely, podľa ktorých sa vyberá nová populácia – generačný (anglicky – generational) a inkrementálny (anglicky – steady–state) model. Pri generačnom modely sa nová populácia skladá len z de novo vygenerovaných potomkov. Na druhú stranu, opačný extrém je pri inkrementálnom modely, kde sa vyberie malé množstvo potomkov, ktorí nahradia podľa zvolenej metódy rovnaký počet jedincov v pôvodnej populácii. Nasleduje stručný popis niektorých vybraných metód [17].

Nahradenie najhorších

Pri tejto metóde je λ najhorších jedincov vybraných z populácie aby boli nahradení potomkami. Aj keď táto metóda vedie k veľmi rýchlemu zlepšovaniu priemernej fitness v populácii, tak to tiež môže viesť k predčasnej konvergencii, pretože sa algoritmus sústredí na najlepších jedincov [17].

Elitizmus

Elitizmus uchováva doteraz najlepšieho jedinca v populácii. Ak z de novo vytvorených potomkov žiadny nemá lepšiu alebo rovnakú fitness, tak jeden z potomkov sa zahodí. V opačnom prípade je doteraz najlepší jedinec nahradený novým potomkom [17].

Turnaj každý s každým

Tento mechanizmus bol zavedený v evolučnom programovaní, kde je použitý na výber μ jedincov, ktorí prežijú do ďalšej generácie. Tomuto mechanizmu nič nebráni v tom, aby bol použitý aj pre výber rodičov. Metóda funguje tak, že každý jedinec je porovnaný s ďalšími q jedincami. Na konci turnaja sa vyberie μ jedincov, ktorí mali najviac výhier a prežijú do ďalšej generácie [17].

$(\mu + \lambda)$ selekcia

Pri tejto metóde sa potomkovia a rodičia zjednotia a sú zoradení na základe ich fitness. Potom sa vyberie μ najlepších, ktorí budú tvoriť novú populáciu do ďalšej generácie [17].

(μ, λ) selekcia

Táto metóda funguje na princípe, kde je zvyčajne viac potomkov ako rodičov $\lambda > \mu$. Potomkovia sú zoradení podľa fitness a μ najlepších nahradí aktuálnych rodičov [17].

3.6 Genetické operátory

Genetické operátory majú za úlohu z už existujúcich kandidátnych riešení vytvárať nové a zaujímavejšie kandidátne riešenia. Existujú dve hlavné kategórie reprodukčných operátorov – sexuálne a asexuálne. Pod sexuálne genetické operátory patrí napríklad kríženie, pri ktorom sa pomocou selekčného mechanizmu vyberú rodičia a zamiešaním ich génov medzi sebou vytvoria jedného alebo viacerých potomkov. Naopak, pri asexuálnych reprodukčných operátoroch sa tiež vyberie rodič, ale na rozdiel od sexuálnej reprodukcie sa jedinec sám naklonuje a modifikuje. Táto modifikácia sa nazýva mutácia. Pri mutácii sa v jedincovi

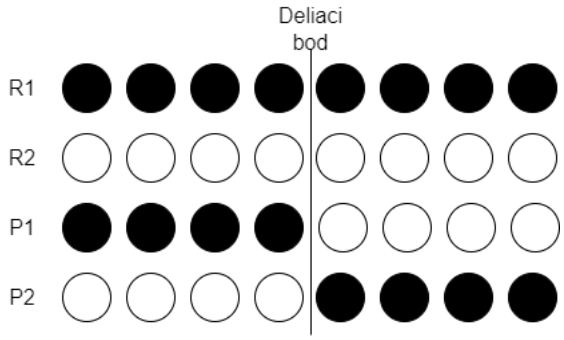
náhodne vyberú a modifikujú gény. Vo väčšine evolučných algoritmov sa tieto dva princípy navzájom kombinujú [14].

Kríženie

Ako bolo spomenuté vyššie, kríženie patrí medzi sexuálne genetické operátory, pri ktorom sa samotné gény alebo časti genotypu vybraných jedincov tzv. rodičov náhodne zamenia medzi sebou a vytvoria potomkov. Proces výberu génov alebo častí genotypu nie je jednoduchý, pretože závisí na zvolenej reprezentácii. Medzi klasické operátory kríženia pre binárnu reprezentáciu genotypu patrí jednobodové kríženie, dvojbodové kríženie a uniformné kríženie, ktoré sú popísané nižšie.

Jednobodové kríženie

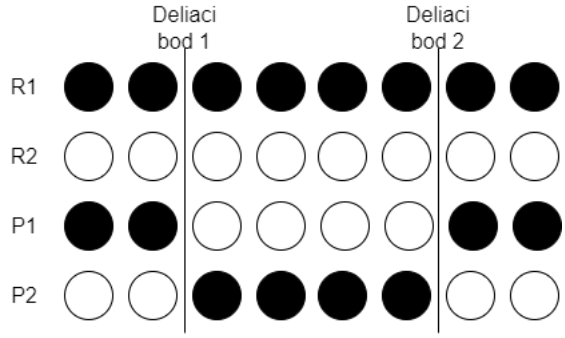
Pri tomto druhu kríženia sa medzi rodičmi náhodne vyberie jeden deliaci bod. Bity na ľavej strane od deliaceho bodu prvého (resp. druhého) rodiča sa priradia prvému (resp. druhému) potomkovi a bity na pravej strane druhému (resp. prvému) potomkovi. Obrázok 3.2 znázorňuje princíp tohto kríženia [18] [38].



Obr. 3.2: Tento obrázok znázorňuje princíp jednobodového kríženia, kde R1 a R2 sú rodičia a P1 a P2 sú dvaja potomkovia.

Dvojbodové kríženie

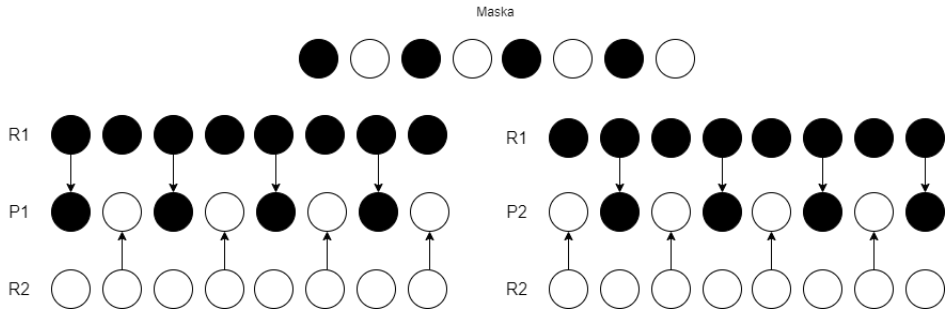
Pri dvojbodovom krížení sa náhodne vyberú dva deliace body. Bity pred prvým a za druhým deliacim bodom sa z prvého (resp. druhého) rodiča priradia prvému (resp. druhému) potomkovi. Bity uprostred sa z prvého (resp. druhého) rodiča priradia druhému (resp. prvému) potomkovi. Obrázok 3.3 znázorňuje princíp tohto kríženia [18] [38].



Obr. 3.3: Tento obrázok znázorňuje princíp dvojbodového kríženia, kde R1 a R2 sú rodičia a P1 a P2 sú potomkovia.

Uniformné kríženie

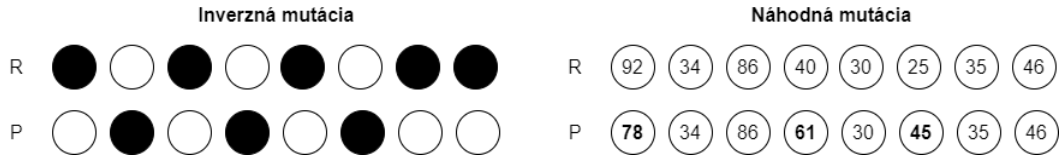
Pri uniformnom krížení sa používa náhodne vygenerovaná binárna maska. Maska má rovnakú dĺžku ako každý z rodičov. Prvý (resp. druhý) potomok si vyberá z prvého (resp. druhého) rodiča gény, ktoré majú v maske hodnotu 1 (resp. 0) a z druhého (resp. prvého) rodiča gény, kde maska má hodnotou 0 (resp. 1) [18] [38]. Obrázok 3.4 znázorňuje princíp tohto kríženia.



Obr. 3.4: Tento obrázok znázorňuje uniformné kríženie. R1 a R2 sú rodičia a P1 a P2 sú potomkovia. Ďalej je tu maska, ktorá určuje, ktoré bity z R1 resp. R2 pôjdu do P1 a P2

Mutácia

Operátor mutácie spadá pod kategóriu asexuálnych genetických operátorov. Rovnako ako pri krížení, je mutácia závislá od reprezentácie. Gény pre mutáciu sú vyberané podľa predefinovanej pravdepodobnosti mutácie, ktorá je rovnaká pre každý gén. Táto pravdepodobnosť sa nastavuje veľmi malá, aby de novo vytvorení jedinci neboli až príliš odlišní od svojich rodičov. Pri binárnej reprezentácii existuje napríklad inverzná mutácia, pri ktorej náhodne vybrané gény zmenia svoju hodnotu z 0 na 1 (resp. 1 na 0). Od inverznej mutácie je odvodená náhodná mutácia, ktorá sa používa pri celočíselnej reprezentácii. Vybrané gény pri mutácii naberú nové hodnoty z intervalu platných hodnôt [38] [18]. Na obrázku 3.5. sú príklady týchto dvoch mutácií.



Obr. 3.5: Na tomto obrázku je zobrazená inverzná mutácia používaná pri binárnej reprezentácii a náhodná mutácia, ktorá sa používa pri celočíselnej reprezentácii z intervalu $[0, 100]$

Kapitola 4

Genetické programovanie

Genetické programovanie je systematická metóda pre automatické riešenie problémov počítačmi, začínajúca od vysokej úrovne abstrakcie vyhlásením toho, čo je potrebné urobiť [36].

História genetického programovania siaha až ku vzniku evolučných algoritmov. V roku 1958 Friedberg [21] vytvoril algoritmus, ktorý dokázal vyhodnotiť kvalitu počítačového programu, náhodne ho upraviť a skontrolovať či sa daný program zlepšil. Smith vytvoril určitú formu genetického programovania vo svojej PhD práci v roku 1980 [60]. V roku 1981 Forsyth využil genetické programovanie na tri rôzne druhy predikčných problémov [19]. Cramer v roku 1985 vyvinul sekvenčné programy, tam bola prvá zmienka o symbolických výrazových stromoch [13]. Napriek predošlému výskumu genetické programovanie dosiahlo väčšiu popularitu až po publikácii knihy v roku 1992 [35], ktorej autorom bol John Koza [46].

4.1 Stromové genetické programovanie

Stromové genetické programovanie je genetické programovanie, kde genotyp je reprezentovaný ako strom. Algoritmus 4.1 zobrazuje princíp prehľadávania v stromovom genetickom programovaní.

Algoritmus 4.1 Stromové genetické programovanie [64]

Vstup: Počet generácií G, veľkosť populácie N, pravdepodobnosti operácií p_c, p_m, p_r
Výstup: Najlepší nájdený jedinec

```
1: Inicializuj populáciu P náhodnými jedincami o veľkosti N
2: Vyhodnoť fitness každého jedinca v P
3: for g = 1 to G do
4:    P_new ← ∅
5:    while |P_new| < N do
6:       Vyber operáciu (kríženie, mutácia, reprodukcia) na základe pravdepodobností p_c,
         p_m, p_r
7:       if operácia je kríženie then
8:          Vyber dvoch jedincov z P pomocou selekčného mechanizmu
9:          Vykonaj kríženie a pridaj potomkov do P_new
10:      else if operácia je mutácia then
11:         Vyber jedinca z P pomocou selekčného mechanizmu
12:         Vykonaj mutáciu a pridaj výsledok do P_new
13:      else
14:         Vyber jedinca z P pomocou selekčného mechanizmu
15:         Vykonaj reprodukciu a pridaj kopiu jedinca do P_new
16:      end if
17:   end while
18:   P ← P_new
19:   Vyhodnoť fitness jedincov v P
20: end for
21: return Jedinec z P s najvyššou fitness
```

Reprezentácia

Množina všetkých validných štruktúr, ktoré sa pri stromovom genetickom programovaní môže vygenerovať zahŕňa všetky stromy ktoré sa dajú rekurzívne zostaviť z funkčných symbolov $\mathscr{F} = \{f_1, f_2, \ldots, f_n\}$ a terminálnych symbolov $\mathscr{T} = \{t_1, t_1 \ldots, t_m\}$. Funkcie sa používajú v interných uzloch a terminály v listových uzloch stromu. Funkčné symboly môžu byť aritmetické operátory, matematické funkcie, booleovské operácie, podmienkové operácie, iteračné operátory a ďalšie operátory, ktoré vyžaduje riešený problém. Terminálne symboly sú väčšinou konštanty a premenné [64].

Pri výbere funkčných a terminálnych množín sa osvedčilo vyberať také množiny, ktoré budú splňovať dve podmienky – uzavretosť (anglicky – closure) a dostatočnosť (anglicky – sufficiency). Pod uzavretosťou sa myslí, že akákoľvek funkcia z množiny funkčných symbolov musí vedieť prijať výstup alebo dátový typ akejkoľvek inej funkcie z množiny funkčných symbolov a akúkoľvek hodnotu alebo dátový typ, ktorý sa môže objaviť v akejkoľvek sade terminálov. To zabezpečí, že akákoľvek kompozícia funkcií a terminálov v systéme bude platná. Zároveň sa požaduje, aby bola vybraná taká sada funkčných a terminálnych symbolov, ktoré budú dostatočné na zostavenie riešenia pre daný problém. To znamená, že budú spĺňať dostatočnosť (anglicky – sufficiency) [64].

Inicializácia počiatočnej populácie

V genetickom programovaní existuje niekoľko inicializačných metód populácie. Medzi najzákladnejšie patria *grow* metóda, *full* metóda a *ramped half-and-half* metóda. Všetky tieto metódy sú ovládané nastaviteľnou maximálnou povolenou hĺbkou stromov, d [64].

Grow metóda

Inicializácia pomocou *grow* metódy začína výberom symbolu pre koreňový uzol stromu. Symbol je vybraný uniformnou pravdepodobnosťou z množiny funkčných symbolov $\mathscr{F}$. Na začiatku je metóda obmedzená len na funkčné symboly, aby sa nevytvárali degenerované stromy s jediným terminálnym uzlom. Podľa arity zvoleného funkčného symbolu vyberieme s uniformnou pravdepodobnosťou daný počet uzlov (detí) zo zjednotenia množiny funkčných a terminálnych symbolov. Pre každý funkčný symbol medzi týmito uzlami sa rekurzívne volá *grow* metóda až do momentu, kedy uzol nemá hĺbku $d - 1$. V tom momente sa deti vyberajú už len z množiny terminálnych symbolov $\mathscr{T}$ [35].

Full metóda

Full metóda, na rozdiel od *grow* metódy, vyberá len z funkčných symbolov až kým nedosiahne hĺbku $d-1$. Následne vyberá už len z terminálnych symbolov. Vďaka tomuto procesu všetky vetvy stromu majú maximálnu hĺbku [35].

Ramped half-and-half metóda

Pre väčšiu diverzitu populácie sa používa *ramped half-and-half* metóda. Pri tejto metóde sa pri veľkosti populácie μ vytvorí $\frac{\mu}{d}$ skupín. Polovica každej skupiny jedincov je inicializovaná pomocou *grow* metódy a druhá polovica pomocou *full* metódy. Pritom prvá skupina je inicializovaná s hĺbkou 1, druhá s hĺbkou 2 atď. Týmto spôsobom táto metóda vytvorí stromy o rôznych veľkostiach a tvaroch. To zaistí počiatočnú diverzitu v populácii [35].

Ohodnotenie populácie

Ako bolo spomenuté už v kapitole o evolučných algoritmoch, každému jedincovi je priraďovaná hodnota fitness, ktorá reprezentuje to, ako daný jedinec rieši daný problém. V genetickom programovaní sa často používajú štyri druhy počítania hodnoty fitness – neupravená, štandardizovaná, upravená a normalizovaná fitness [35].

Neupravená fitness

Neupravená fitness (anglicky – raw fitness) je považovaná za výsledok účelovej funkcie. Napríklad pri probléme, kde cieľom je nájsť najkratšiu cestu pre obchodného cestujúceho, by sa jednalo priamo o hodnotu vzdialenosti.

Mnohokrát sa pri výpočte fitness používajú testovacie prípady. Testovacie prípady sú očakávané výstupy daného jedinca. Často sa jedná o vzorku celého doménového priestoru, ktorý je v mnohých prípadoch príliš veľký alebo nekonečný. V prípade, keď je doménový priestor veľmi malý, môžu sa použiť všetky jeho testovacie prípady. Pri vzorke z celého doménového priestoru sa ponúka možnosť vytvárať novú testovaciu sadu pre každú generáciu. Tento prístup by mohol viesť k lepšej generalizácii daného riešenia, ale často sa testovacia sada vyberie na začiatku evolúcie a počas jej behu sa už nemení [35].

Štandardizovaná fitness

Štandardizovaná fitness (anglicky – standardized fitness) transformuje fitness tak, že vždy menšia hodnota fitness je lepšia. V prípade keď máme problém, pri ktorom menšie hodnoty neupravenej fitness sú lepšie, tak by štandardizovaná fitness bola rovná neupravenej fitness. V prípade kedy by to bolo opačne, potrebujeme poznať maximálnu hodnotu neupravenej fitness a odčítať ju od aktuálnej. V takom prípade získame znovu štandardizovanú fitness [35].

Upravená fitness

Upravená fitness (anglicky – adjusted fitness) je počítaná zo štandardizovanej fitness nasledovne:

$$a(i,t) = \frac{1}{1 + s(i,t)} \tag{4.1}$$

Kde a(i, t) je vypočítaná upravená fitness pre jedinca i v generácii t a $s(i,t)$ je štandardizovaná fitness pre jedinca i v generácii t. Táto fitness nadobúda hodnoty od 0 (najhoršie riešenie) do 1 (najlepšie riešenie) [35].

Normalizovaná fitness

Pre potreby proporcionálnej selekcii založenej na fitness sa vyžaduje normalizovaná fitness (anglicky – normalized fitness), ktorá sa vypočíta z upravenej fitness nasledovne:

$$f_n(i,t) = \frac{a(i,t)}{\sum_{k=1}^{M} a(k,t)} \tag{4.2}$$

Táto fitness nadobúda hodnoty od 0 do 1. Väčšie číslo znamená lepšie riešenie. Súčet všetkých normalizovaných fitness je 1 [35].

Selekcia

V genetickom programovaní sa najčastejšie používajú tri selekčné mechanizmy: proporcionálna selekcia založená na fitness, turnajová selekcia a selekcia založená na poradí. Všetky tieto selekcie sú založené na fitness. Pritom pri prvej z nich je potrebná normalizovaná fitness, kde normalizovaná fitness určuje pravdepodobnosť výberu daného jedinca k reprodukcii [35].

Všetky tieto selekčné mechanizmy boli popísané v Podkapitole 3.5.

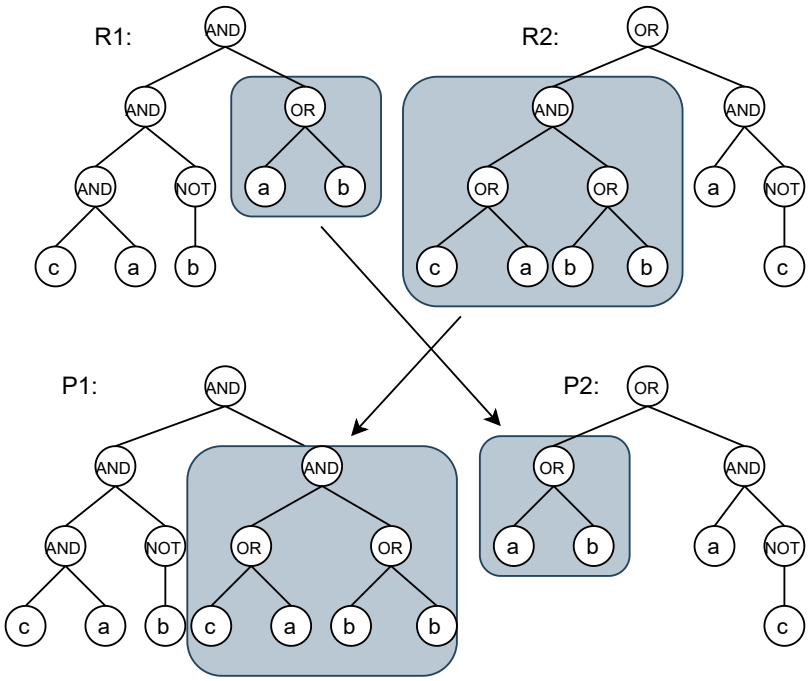
Reprodukcia

Reprodukcia (anglicky – reproduction) je jednou zo základných a najjednoduchších genetických operátorov v genetickom programovaní. Jedná sa o operátor, pri ktorom sa na základe vybraného selekčného mechanizmu vyberie jeden rodič, ktorý sa následne nezmenený skopíruje do novej populácie. Tento rodič sa ale vzápätí z aktuálnej populácie nemaže. To znamená, že jeden rodič môže byť skopírovaný do novej populácie viackrát. Ak sa testovacie prípady v priebehu generáciami nemenia, je vhodné aby takto vybraný jedinci naprieč generáciami neboli znovu vyhodnocovaní a to ušetrí výpočtové zdroje [35].

Kríženie

Štandardné kríženie v genetickom programovaní, ako bolo popísané v [35], začína selekciou dvoch rodičov pomocou vybraného selekčného mechanizmu. V oboch rodičoch sa následne nezávisle na sebe vyberie tzv. bod kríženia (anglicky – crossover point). Bod kríženia sa často vyberá z pravdepodobnosťou 0.9 pre interný uzol a 0.1 pre všetky uzle (interné aj listové). Od bodu kríženia vznikne tzv. fragment kríženia. Jedná sa o podstrom, ktorý má koreň v bode kríženia. Týmto operátorom vzniknú dvaja potomkovia a to nasledovne. Z prvého rodiča a druhého rodiča sa vyberie a odstráni fragment kríženia. Následne sa fragment kríženia prvého (resp. druhého) rodiča pridá na bod kríženia druhého (resp. prvého) rodiča. Problém môže nastať, keď jeden bod kríženia je interný uzol a druhý je listový uzol. Toto môže byť pri začiatku evolučného procesu žiadúce, ale neskôr začne nekontrolovateľne rásť hĺbka stromu. Tento fenomén sa nazýva nafúknutie (anglicky – bloat). Stromy, ktoré takýmto spôsobom prekonajú svoju maximálnu hĺbku sa buď zamietnu, alebo sa im priradí veľmi malá fitness [64].

V odbornej literatúre bolo navrhnutých viacero variácií kríženia. Najčastejšie sa stretávame s optimalizáciou priraďovania pravdepodobnosti pre optimálnejší výber bodu kríženia. Tam sa pre koreňové a listové uzle dáva najnižšia pravdepodobnosť [64].

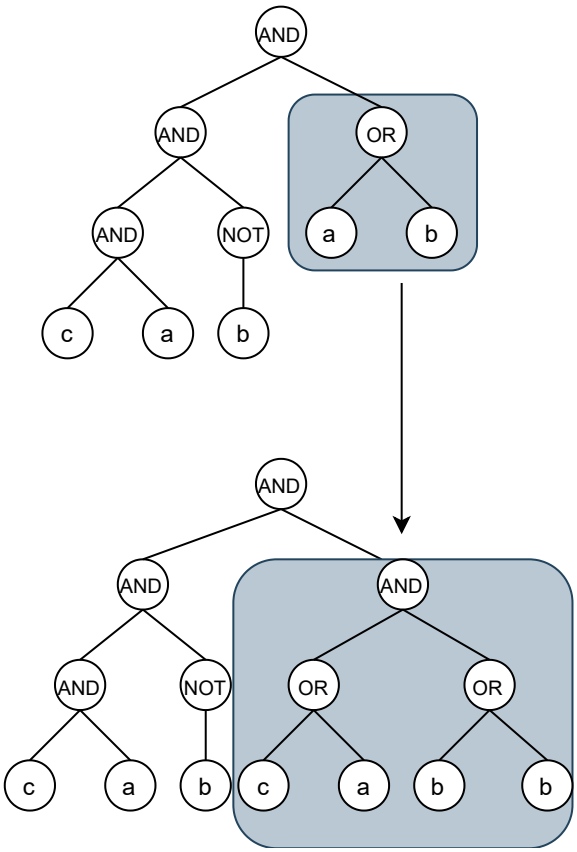


Obr. 4.1: Príklad kríženia stromov. Prevzaté z [34].

Mutácia

Štandardný operátor mutácie začína tým, že sa vyberie jeden rodič pomocou zvoleného selekčného mechanizmu. Následne, podobne ako pri operátore kríženia, sa vyberie bod mutácie (anglicky – mutation point). Podstrom, ktorý je zakorenený v tomto bode mutácie sa zmaže a nahradí de novo vygenerovaným podstromom. Tento vygenerovaný podstrom je taktiež obmedzený maximálnou hĺbkou celého stromu [64].

Podobne ako pri krížení, aj tu bolo navrhnutých viac variácií štandardného mutačného operátoru. Najčastejšie sa používa obmedzenie pravdepodobnosti pre výber koreňového uzla a listových uzlov ako body mutácie. Jedným z ďalších príkladov je bodová mutácia (anglicky – point mutation) [51], ktorá náhodne vyberie niekoľko uzlov. Ak sa jedná o interný uzol, zamení ho za iný funkčný symbol rovnakej arity. Naopak, ak sa jedná o terminálny symbol, tak ho zamení za iný terminálny symbol [64].



Obr. 4.2: Príklad mutácie stromu. Prevzaté z [34].

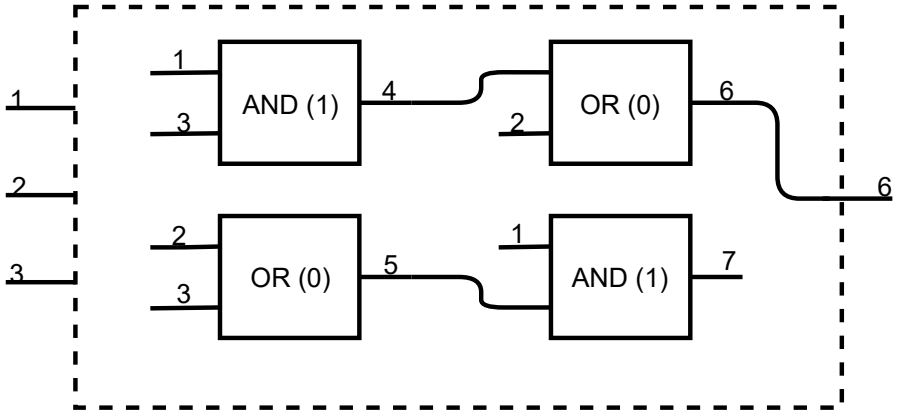
4.2 Kartézske genetické programovanie

Kartézske genetické programovanie (anglicky – cartesian genetic programming, CGP) vzniklo z metódy pre návrh digitálnych obvodov, ktorú navrhol Miller v roku 1997 [43]. Napriek tomu sa pojem kartézske genetické programovanie objavuje až o dva roky neskôr v [44] a bolo navrhnuté ako forma genetického programovania o rok neskôr v [47]. Slovo "kartézske"naznačuje, že programy sú reprezentované ako dvojdimenzinálna mriežka uzlov [45].

Oproti stromovému genetickému programovaniu, kde sú programy reprezentované ako stromy, je program v kartézskom genetickom programovaní zakódovaný ako orientovaný acyklický graf. Zakódovanie grafov namiesto stromov má výhody v tom, že uzly môžu byť použité viackrát a aj to, že grafy môžu mať viacero výstupov. Ešte k tomu táto reprezentácia dokáže reprezentovať rôzne druhy štruktúr (napr. sústavu rovníc, neurónových sietí, algoritmov a elektronických obvodov) [48].

Reprezentácia

V štandardnej forme kartézskeho genetického programovania sú programy reprezentované ako orientovaný acyklický graf. Tieto grafy sú reprezentované ako dvojdimenzionálna kartézska mriežka uzlov. Každý uzol má určitý počet vstupov a funkciu. Funkcia je reprezentovaná ako celočíselná hodnota, ktorá odkazuje do vyhľadávacej tabuľky užívateľom definovaných funkcií. Počet vstupov odpovedá maximálnej arite a všetkých funkcií. Ak niektorá funkcia používa menej vstupov, tak ostatné vstupy sú ignorované. V štandardnom kartézskom genetickom programovaní si uzle svoje vstupy môžu pripájať len na prechádzajúce uzle (ktoré sa nenachádzajú v rovnakom stĺpci) alebo primárne vstupy. Primárne vstupy a uzle sú reprezentované ako celočíselné hodnoty (adresy), na ktoré sa jednotlivé vstupy odkazujú. Primárne vstupy sú adresované od 0 do $n_i - 1$, kde n_i je počet primárnych vstupov. Uzle sú adresované od n_i do $n_i + n_n - 1$, kde n_n je počet uzlov. Ďalej genotyp obsahuje n_o výstupných génov. Výstupy naberajú hodnotu adresy uzlu ku ktorému sú pripojené. Kartézske genetické programovanie má tri parametre, ktoré definujú jeho dimenzionalitu a konektivitu. Tieto parametre sú: počet stĺpcov n_c, počet riadkov n_r a hodnota maximálneho prepojenia s prechádzajúcimi uzlami l. Vynásobením prvých dvoch parametrov $n_n = n_c * n_r$ dostaneme maximálny počet uzlov v kartézskej mriežke. Posledný parameter určuje to, kde môže daný uzol pripojiť svoje vstupy. Ak je parameter $l = 1$, tak uzle môžu svoje vstupy pripojiť len na primárne vstupy alebo prechádzajúce uzle (ktoré sa ale nenachádzajú v rovnakom stĺpci). Naopak, ak je $l = n_c$, tak sa uzle môžu pripojiť na akýkoľvek predchádzajúci uzol (ktorý nie je v tom istom stĺpci) a primárny vstup. Veľkosť grafu je určená pomocou rovnice $L = n_n(a + 1) + n_o$ [48].



Obr. 4.3: Tento obrázok obsahuje príklad kandidátneho riešenia v kartézskom genetickom programovaní s parametrami $n_r = 2$, $n_c = 2$, $n_i = 3$, $n_o = 1$, $n_n = 2$, $n_f = 2$, $l = 1$, $\Gamma = \{OR(0), AND(1)\}$, Chromozóm: (1,3,1), (2,3,0), (4,2,0), (1,5,1), (6). Prevzaté z [34]

Algoritmus

Štandardný prehľadávací algoritmus (Algoritmus 4.2) pre kartézske genetické programovanie je inšpirovaný $1 + \lambda$ evolučnou stratégiou, v ktorej je vybraný jeden genotyp ako rodič a vytvára λ potomkov. Potomkovia, ktorí majú rovnakú alebo lepšiu fitness ako ich rodič sa stávajú novým rodičom do ďalšej generácie [48].

Algoritmus 4.2 Vyhľadávací algoritmus kartézskeho genetického programovania

```
1: for i = 0 to λ − 1 do
2:    Vygeneruj náhodne jedinca i
3: end for
4: Vyber najvhodnejšieho jedinca, ktorý sa stane rodičom
5: while Nenájde sa riešenie alebo nedosiahne sa limit generácií do
6:    for i = 0 to λ − 1 do
7:       Vygeneruj potomka i mutovaním rodiča
8:    end for
9:    Vyber nového rodiča podľa nasledujúcich pravidiel:
10:   if Jeden potomok má lepšiu fitness než akýkoľvek iný člen populácie then
11:      Potomok je vybraný ako rodič
12:   else if Jeden alebo viac potomkov majú rovnakú fitness ako rodič then
13:      Náhodne vyber jedného z nich ako rodiča
14:   else
15:      Chromozóm rodiča zostáva rovnaký ako predtým
16:   end if
17: end while
```

Kríženie

V štandardnom kartézskom genetickom programovaní sa kríženie nepoužíva. V originálnom článku [44] Miller zistil, že kríženie nemalo takmer žiadny efekt na efektivitu kartézskeho genetického programovania a od vtedy bolo v nasledujúcich prácach ignorované [48]. V nasledujúcich rokoch boli skúmané viaceré variácie kríženia v kartézskom genetickom programovaní, ale žiadne nepriniesli výrazné zlepšenie štandardného prístupu.

Mutácia

V štandardnom kartézskom genetickom programovaní sa najčastejšie používa bodová mutácia alebo pravdepodobnostná mutácia. Pri bodovej mutácii sa vyberie percento z celkového počtu génov, ktoré budú zmutované pre vytvorenie nového potomka. V pravdepodobnostnej mutácii je každý gén vystavený mutácii s užívteľom vybranou pravdepodobnosťou. Bodová mutácia je jednoduchšia a efektívnejšia [48].

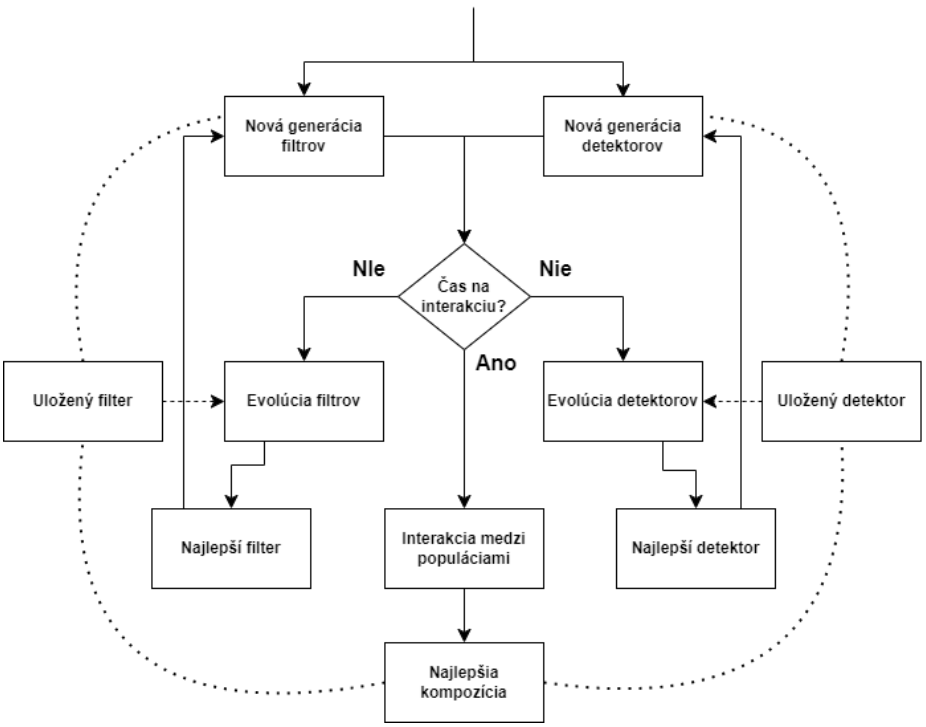
Reprezentácia kartézskeho genetického programovania umožňuje to, že pomocou mutácie môže byť nájdených veľa zaujímavých riešení len pomocou mutácie. Keďže väčšina uzlov v kartézskom genetickom programovaní sú nadbytočné, tak sa väčšina mutácii odohráva v týchto nadbytočných neaktívnych uzloch. Preto sa často stáva, že vytvorení potomkovia majú rovnaký fenotyp ako ich rodičia. V takýto prípadoch sa neoplatí vyhodnocovať tohto jedinca populácie, pretože bude mať rovnakú fitness ako jeho rodič. Z toho dôvodu bola preskúmaná metóda *skip* [22]. Táto metóda sa vyhodnotí každú generáciu a ak sa zistí, že potomok je rovnaký s rodičom podľa jeho aktívnych uzlov, tak preskočí jeho vyhodnocovanie a priradí sa mu fitness jeho rodiča. Mimo metódu *skip* navrhli aj metódy *accumulate* a *single* [22]. Pričom prvá mutuje potomka až dokým niekoľko jeho aktívnych génov nie sú rozdielne od jeho rodiča a druhá mutuje potomka až dokedy sa nezmení práve jeden aktívny gén. Z ich experimentov zistili, že metóda *skip* a *accumulate* neprodukuje výrazné zlepšenia a keďže metóda *skip* je jednoduchšia na pochopenie a implementáciu, tak metódu *accumulate* zavrhli. Metóda *single* a *skip* sú naďalej používané a potrebujú byť viac

preskúmané. V kartézskom genetickom programovaní je silná pozičná zaujatosť v tom, že je pravdepodobnejšie že budú aktívne uzly na ľavej strane genotypu. K tejto problematike boli navrhnuté a preskúmané dve metódy *reorder* a *DAG* [23]. *Reorder* v každej generácii pred mutáciou preusporiada uzle v rodičovi tak, aby zachovali jeho sémantiku. *DAG* povoľuje mutácie dopredu aj dozadu, dokým nie je detekovaný cyklus. Z experimentov vyšlo, že *reoder* má lepšie výsledky a jeho fenotypy využívajú viac aktívnych uzlov. Metóda *DAG* bola veľmi výpočetne náročná a nemala veľký prínos, preto bolo navrhnuté, aby sa metóda *DAG* už v budúcnosti nepoužívala [48].

4.3 Kompozičná koevolúcia v CGP

Kompozičná koevolúcia je pokročilá technika, ktorá obohacuje evolučný proces v CGP. V štandardnom CGP je pri riešení zložitého problému výhodné rozdeliť ho na menšie podproblémy. Tieto menšie podproblémy sa riešia nezávisle, čo nepočíta s možnou interakciou medzi riešeniami jednotlivých podproblémov [16] [58] [59].

Kompozičná koevolúcia túto medzeru vypĺňa tým, že zavádza interakciu medzi populáciami, čo vedie k vytvoreniu kvalitnejších riešení, ktoré sú schopné spolupracovať. Interakcia prebieha každých N generácií. Príliš časté ani príliš zriedkavé intervaly interakcie však nevedú k optimálnym výsledkom, pretože buď nie sú vykonané dostatočné zmeny v populácii, alebo je ich vykonaných príliš veľa [16] [58] [59].



Obr. 4.4: Príklad interakcie dvoch populácii v kartézskom genetickom programovaní s použitím kompozičnej koevolúcie. Obrázok prevzatý a upravený z [16].

Interakcia medzi populáciami sa uskutočňuje výpočtom fitness hodnoty za použitia všetkých kombinácií jedincov zo všetkých populácií. Táto operácia je veľmi výpočtovo náročná. Po interakcii sa vyberie najlepšia kompozícia jedincov, ktorá je uložená a označená ako naj-

lepšia. Jednotliví jedinci z tejto kompozície sú priradení ku všetkým ostatným populáciám. Následne sa pokračuje štandardným evolučným procesom až do ďalšej interakcie [16] [58] [59].

Pri výpočte fitness hodnoty pre jednotlivé kandidátne riešenia sa môže zohľadniť aj interakcia s ostatnými populáciami, najmä ak v danej populácii existuje viac jedincov s rovnakou fitness hodnotou. Ak sa pri nasledujúcej interakcii nenájde kompozícia s lepšou fitness hodnotou, ponechá sa pôvodná, ale jedinci z najlepšej kompozície z aktuálnej interakcie sú priradení ku všetkým ostatným populáciám [16] [58] [59]. Príklad interakcie medzi dvomi populáciami je zobrazený na obr. 4.4.

Kapitola 5

Učenie reprezentácie

Učenie reprezentácie je jeden zo základných kameňov hlbokých neurónových sietí. Zameriava sa na automatické hľadanie reprezentácie zo surových dát, aby sa táto reprezentácia dala ďalej použiť pri iných úlohách akou je napríklad klasifikácia. Tento proces hrá hlavnú rolu v tom aby modely ľahšie pochopili komplexné dáta ako sú obrázky, zvuk a text [24].

Cieľom pri učení reprezentácie je transformovať surové dáta do viac zrozumiteľného formátu, ktorý bude reprezentovať dôležité informácie, ktoré reprezentujú poskytnuté dáta a nepotrebné informácie budú zanedbané alebo úplne odstránené z reprezentácie. Táto nová reprezentácia poskytuje jednoduchšie a efektívnejšie učenie v nasledujúcich vrstvách neurónovej siete. Preto má učenie reprezentácie veľký význam v hlbokých neurónových sieťach. Tieto naučené reprezentácie reprezentujú príznaky, ktoré sa v minulosti museli manuálne vyberať. Pritom automaticky vytvorené reprezentácie dát môžu obsahovať príznaky, ktoré by dizajnér príznakov vôbec nemusel pokladať za dôležité. Tento prístup zároveň znížil požiadavky na znalosť domény, ktorej sa daný problém týka.

V nasledujúcich sekciách tejto kapitoly budú spomenuté typy učenia reprezentácie, základná technika pre vytváranie reprezentácii, ktorá sa následne dá aplikovať do viacerých druhov hlbokých neurónových sietí a ako posledná časť tejto kapitoly bude spomenutá aplikácia učenia reprezentácie v genetickom programovaní.

5.1 Typy učenia reprezentácií

Typy učenia reprezentácie priamo nadväzujú na typy učenia reprezentácie v podkapitole 2.1. Rozdiel je v tom, že táto sekcia sa zaoberá automatickým návrhom reprezentácii tzv. príznakov.

Učenie s učiteľom

V podkapitole 2.1 sa preberali základné princípy učenia s učiteľom. V tejto sekcii je vysvetlená rola učenia reprezentácie v učení s učiteľom. V tomto kontexte učenie s učiteľom nemapuje len vstupy na výstupy, ale zameriava sa aj na získavanie zmysluplných reprezentácii, ktoré zachytávajú podstatu vstupných dát. Ako príklad môžeme uviesť konvolučné neurónové siete, ktoré sa pri učení na dátovej sade obrázkov naučia hierarchické reprezentácie, ktoré sú veľmi efektívne pri úlohách, akými môžu byť detekcia objektov alebo segmentácia [5].

Učenie bez učiteľa

Ako bolo načrtnuté v podkapitole 2.1, učenie bez učiteľa zahŕňa učenie s neoznačenými dátami. Tento prístup v učení reprezentácií je nápomocný pri objavovaní skrytých štruktúr v dátach. Techniky ako autoenkóder, ktoré sa používajú pre zníženie dimenzionality, pomáhajú pri učení komprimovaných, ale zaujímavých a informatívnych reprezentácií dát, ktoré môžu byť dôležité pri úlohách ako je klasifikácia, detekcia anomálií alebo generácia nových dát [5].

Učenie s aj bez učiteľa

Učenie s aj bez učiteľa využíva pri učení reprezentácií prednosti oboch typov učenia. Využíva označené dáta pre riadenie procesu učenia a zároveň využíva neoznačené dáta pre získavanie užitočných reprezentácií príznakov. Tento prístup je veľmi užitočný ak získanie označených dát je veľmi drahé alebo nepraktické [70].

Samo–riadené učenie

Samo–riadené učenie je podmnožina učenia bez učiteľa, kde si model vytvára vlastné úlohy zo vstupných dát. Cieľom tohto prístupu je vytvorenie úlohy, pri ktorej model predpovedá niektoré časti dát z ostatných častí. Tento prístup umožňuje modelu naučiť sa kvalitné reprezentácie príznakov. Príkladom tejto úlohy môže byť maskovanie častí textu alebo obrázku a model sa musí snažiť správne doplniť chýbajúce časti. Počas tohto procesu sú získané bohaté reprezentácie [31].

Transferové učenie

Transferové učenie (anglicky – transfer learning) hrá pri učení reprezentácie významnú rolu. Toto učenie zahŕňa použitie modelu, ktorý bol trénovaný na rozsiahlom množstve dát, pri riešení jedného problému na použitie ako počiatočný bod pri druhom probléme. Tento prístup môže byť veľmi užitočný v prípadoch, keď máme obmedzené množstvo dát. Predtrénovaný model na veľkej dátovej sade môže vytvoriť veľmi kvalitnú generalizovanú reprezentáciu príznakov, ktorá môže byť doladená (anglicky – fine-tuned) pre špecifickú úlohu, čím zníži potrebný počet trénovacích dát [69].

Kontrastné učenie

Kontrastné učenie (anglicky – Contrastive Learning) je technika v učení reprezentácií, ktorá sa zameriava na učenie podobností a rozdielov medzi jednotlivými vzorkami dát. Základná myšlienka spočíva v učení modelu rozpoznať rozdiely medzi podobnými a odlišnými vzorkami vstupu. Tento prístup je najčastejšie používaný v učení bez učiteľa alebo v samo–riadenom učení [25].

5.2 Autoenkóder

Autoenkóder je typ neurónovej siete používaný na učenie sa efektívneho kódovania neoznačené dáta. Je často využívaný v úlohách učenia sa reprezentácie alebo redukcie dimenzionality [27].

Základná architektúra

Základnú štruktúru tejto neurónovej siete tvoria dve hlavné komponenty: kodér a dekodér.

Kodér

Táto časť siete komprimuje vstup do priestoru nižších dimenzií, známeho tiež ako skrytá reprezentácia. Efektívne kóduje vstupné dáta do kompaktnejšej formy [27].

Dekodér

Následne dekodér rekonštruuje vstupné dáta z ich zakódovanej podoby, pričom sa snaží minimalizovať chybu medzi pôvodnými a rekonštruovanými verziami [27].

Trénovanie a symetria

Celá sieť sa učí minimalizáciou chyby medzi pôvodným vstupom kodéru a výstupom dekodéru. Architektúra štandardného autoenkóderu je často symetrická, kde kodér a dekodér väčšinou bývajú zrkadlovým obrazom. Avšak, táto symetria nie je striktnou požiadavkou.

Flexibilita a aplikácie

Architektúra umožňuje použitie rôznych modelov neurónových sietí pre kodér a dekodér, ako sú viacvrstvové perceptróny, konvolučné neuronové siete, transfomery a iné. V kontexte konvolučných neurónových sietí môžu byť aplikácie autoenkóderu napríklad v úlohách, ako sú rekonštrukcia obrazu, redukcia šumu alebo kompresia dát.

5.3 Variačný autoenkóder

Variačný autoenkóder (anglicky – variational autoencoder) predstavuje významnú triedu autoenkóderov, ktorá je známa svojím využitím v generatívnych úlohách. Jedná sa o typ pravdepodobnostného modelu, ktorý sa učí kódovať a generovať dáta spôsobom, pri ktorom sa učí modelovať pravdepodobnostnú distribúciu týchto dát. Na rozdiel od štandardných autoenkóderov, ktoré sa zameriavajú len na kompresiu a rekonštrukciu vstupných dát, variačné autoenkódery sú navrhnuté špeciálne pre generatívne úlohy [32].

Architektúra

Podobne ako pri štandardnom autoenkóderi, variačný autoenkóder sa skladá z dvoch hlavných komponent: kodéru a dekodéru.

Kodér

Optoti štandardnému autoenkóderu, kde kodér priamo kóduje vstup do skrytej reprezentácie, variačný autoenkóder produkuje parametre (medián a smerodajnú odchýlku) pravdepodobnostnej distribúcie, ktorá reprezentuje dáta [32].

Dekodér

Dekodér vzorkuje z tejto pravdepodobnostnej distribúcie a snaží sa zrekonštruovať pôvodný vstup na základe vzorkovaných dát [32].

Aplikácie

Variačný autoenkóder nachádza uplatnenie vo viacerých oblastiach, medzi ktoré patria:

1. **Generatívne úlohy:** Pri tejto úlohe variačné autoenkódery generujú nové dáta, ktoré sú podobné tým trénovaným.
2. **Doplnenie dátovej sady:** Variačné autoenkódery môžu doplniť trénovaciu dátovú sadu o chýbajúce dáta vzorkovaním naučenej pravdepodobnostnej distribúcie.
3. **Detekcia odľahlých hodnôt:** Naučením pravdepodobnostnej distribúcie môžu byť identifikované odľahlé hodnoty alebo anomálie, ktoré do tejto distribúcie nepatria.

5.4 Učenie reprezentácie v genetickom programovaní

Táto sekcia sa zameriava na využitie učenia reprezentácie v genetickom programovaní. V nedávnom článku [10] bola vytvorená nová distribuovaná reprezentácia programov v genetickom programovaní s použitím architektúry transformátoru, ktorá prináša značné výhody v efektivite a presnosti reprezentácie.

Postup

Postup zahŕňa vygenerovanie dát na úlohe symbolickej regresie. Jednotlivé programy mali mnoho stromov rôznych veľkostí. Pre konzistenciu v dátach a kvôli tomu, že ich prístup vyžadoval aby vstup do transformátoru mal fixnú veľkosť, použili metódu výplne stromu a chýbajúce uzle stromu nahradili uzlami, ktoré reprezentovali výplň až do maximálnej hĺbky stromu. Aby mohli byť vzniknuté stromy použité pre vstup do neurónovej siete, museli byť najskôr linearizované pomocou metódy DFS (depth–first search) a následne tokenizované.

Výsledky

Analýza natrénovanej siete odhalila, že vytvorená reprezentácia sa sústreďuje primárne na syntax, pričom sémantika vyžaduje ďalšie skúmanie a možnú integráciu do trénovacích dát v budúcnosti. Jeden z ďalších experimentov bol zameraný na aplikáciu tejto reprezentácie v regulovaní diverzity populácie pomocou kosínusovej vzdialenosti medzi stromami. Tento prístup porovnali s aktuálne dvomi najpoužívanejšími metrikami v tejto oblasti a nová navrhnutá metóda ukázala pri niektorých dátových sadách výrazné zlepšenia.

Kapitola 6

Návrh riešenia

Cieľom tejto práce je automatické vytváranie reprezentácií kartézskych genetických programov pomocou hlbokých neurónových sietí a preskúmanie ich využitia a správania pri návrhu kartézskych genetických programov.

Pre splnenie cieľa tejto práce je potrebné získať tréningové dáta, ktoré po predspracovaní budú pomocou techník učenia reprezentácie použité pri tréningu hlbokej neurónovej siete, ktorá bude generovať nové vektorové reprezentácie kartészkých genetických programov. Vzniknuté vektorové reprezentácie budú použité na preskúmanie nových a na vylepšení starých metód v kartézskom genetickom programovaní.

6.1 Získanie dát

K získaniu dát bol implementovaný algoritmus kartézskeho genetického programovania s využitím kompozičnej koevolúcie. Tento prístup bol vybraný na základe dobrých výsledkov oproti štandardnému prístupu v tejto oblasti [16]. Dáta boli vygenerované na úlohe návrhu obrazových filtrov s využitím detektoru šumu. Filter a detektor šumu sú samostatné populácie a počas evolúcie sa vyvíjajú nezávisle na sebe až dokým nepríde čas na interakciu medzi populáciami. Úlohou detektoru šumu je naučiť sa správne klasifikovať poškodené pixle a úlohou filtra je naučiť sa tieto pixle opraviť. Implementácia a následné experimentovanie boli inšpirované prácou [16]. Bolo potrebné sa zamyslieť nad adekvátnym výberom parametrov pre algoritmus kartézskeho genetického programovania, tréningových dát, mutačného operátora a fitness funkcie. Taktiež bolo nevyhnutné zvážiť dátový typ pri výpočte, výber použitých funkcií, počet generácií, veľkosť filtra, frekvenciu interakcie medzi dvomi populáciami a ako často sa budú ukladať jedinci populácie pre potreby získania dát. Mieru mutácie nebolo treba určovať, pretože ako mutačný operátor bola vybraná metóda *single*, ktorá je popísaná v podkapitole 4.2. Fitness funkcia pre detektor šumu bola vybraná presnosť, ktorá sa vypočíta ako:

$$\text{Presnosť} = \frac{TP}{TP + FP + TN + FN} \tag{6.1}$$

kde:

1. **TP:** je skratka pre pozitívne klasifikované hodnoty, ktoré sú skutočne pozitívne (anglicky – true positive, TP).
2. **FP:** je skratka pre hodnoty, ktoré sú klasifikované ako pozitívne, ale v skutočnosti sú negatívne (anglicky – false positive, FP)

3. **TN:** je skratka pre hodnoty, ktoré sú klasifikované ako negatívne, ale v skutočnosti sú negatívne (anglicky – true negative, TN)

4. **FN:** je skratka pre hodnoty, ktoré sú klasifikované ako negatívne, ale v skutočnosti sú pozitívne (anglicky – false negative, FN)

Ako fitness funkciu sa pre filter vybral vrcholový pomer signálu k šumu (anglicky – Peak Signal-to-Noise Ratio, PSNR). Vzorec pre výpočet tejto funkcie je:

$$\mathrm{PSNR(X,Y)} = 10 * \log_{10}\Big(\frac{255^2}{MSE(X,Y)}\Big) \tag{6.2}$$

kde MSE je skratka pre priemernú štvorcovú chybu (anglicky – mean squared error, MSE), ktorá sa vypočíta nasledovne:

$$\mathrm{MSE(X,Y)} = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}(X(i,j) - Y(i,j))^2 \tag{6.3}$$

kde X resp. Y je originálny resp. modifikovaný obrázok a $X(i,j)$ resp. $Y(i,j)$ je pixel v obrázku na súradnici $[i,j]$ a m a n sú dimenzie obrázka jeho výška a šírka. Na trénovanie bol vybraný jeden obrázok z dátovej sady [39] (obr. 6.1). Pri trénovaní filtrov boli použité iba posuvné okná, kde posuvné okno obsahovalo iba poškodené pixle v strede filtra. Ako bolo ukázané v práci [16], tento prístup pri trénovaní filtra ukázal veľmi dobré výsledky. Dokonca aj dramaticky znížil čas potrebný na trénovanie. Detektory boli trénované na všetkých posuvných oknách. Bolo vykonaných 30 behov programu pre každé nastavenie algoritmu. Na začiatku evolúcie sa náhodne vygenerujú jedinci pre populáciu filtrov a detektorov a hneď sa vykoná kompozičná koevolúcia všetkých filtrov a detektorov. Vyberie sa kombinácia filtra a detektoru, ktorá má najlepšiu fitness z hľadiska PSNR. Z aktuálne najlepšej kompozície sa filter resp. detektor pridá k populácii detektorov resp. filtrov. Do ďalšej interakcie sa filtre vyhodnocujú samostatne podľa PSNR a detektory podľa presnosti. Ak sa vyskytne medzi filtrami alebo detektormi viac jedincov s rovnakou fitness hodnotou, tak sa vypočíta fitness hodnota z hľadiska PSNR pomocou uloženého detektoru resp. filtra a vyberie sa najlepší filter resp. detektor. Ak sa pri nasledujúcich interakciách nájde lepší pár než bol predchádzajúci, tak sa nahradí. Ak nie, tak daný pár ostáva, ale najlepší filter a detektor získaní pri aktuálnej kompozícii sa uložia do populácie detektorov a filtrov.

Všetky parametre pre kartézske genetické programovanie boli rovnaké pre detektory aj filtre. Výpočet prebiehal sekvenčne a bol ukončený po 30000 generáciách. Kompozícia bola nastavená na interakciu medzi populáciami každých 100 generácií a ukladanie populácie pre účel získania dát bol nastavený na každých 25 generácií. Prehľad použitých funkcii sa nachádza v tabuľke 6.2 a prehľad všetkých nastaviteľných parametrov sa nachádza v tabuľke 6.1.

(a) Originálny obrázok

(b) Poškodený obrázok 10 % intenzitou šumu

Obr. 6.1: Na ľavej strane sa nachádza originálny obrázok prevzatý z [39]. Na pravej strane sa nachádza poškodený obrázok 10 % intenzitou šumu.

Parametre	Zvolené hodnoty
Tvar kartézskej mriežky	8×8
L-back	Plné prepojenie
Veľkosť populácie	5
Počet generácií	30000
Dátový typ použitý pri výpočte	uint8
Kompozícia	každých 100 generácií
Ukladanie jedincov	každých 25 generácií
Veľkosť filtra	3×3
Šum v trénovacích dátach	10 %, 20 %, 30 % 40 %

Tabuľka 6.1: Popis parametrov pre kartézske genetické programovanie.

#	Funkcia	#	Funkcia
0	255	8	$x >> 1$
1	x	9	$x >> 2$
2	$\bar{y}$	10	$(x << 4) \vee (y >> 4)$
3	$x \vee y$	11	$x + y$
4	$\bar{x} \vee y$	12	$x +^s y$
5	$x \wedge y$	13	$(x + y) >> 1$
6	$x \bar{\wedge} y$	14	$max(x, y)$
7	$x \oplus y$	15	$min(x, y)$

Tabuľka 6.2: Funkcie použité pri výpočtoch prevzaté z [16].

6.2 Predspracovanie dát

Predspracovanie dátovej sady jedincov závisí od modelu strojového učenia pre ktorý sa budú dáta pripravovať. V tejto práci sa pracuje len s modelom transformátoru popísanom v kapitole 2.7. Model bol zvolený na základe práce [10], kde preskúmali transformátor a

viacvrstvový perceptrón na vytváraní reprezentácií pre stromové genetické programovanie. Keďže sa bude jednať o model transformátoru, tak budú dáta reprezentované ako text.

V tejto práci budú predspracovaním vytvorené štyri dátové sady (kompaktná a rozšírená). Každá z týchto variant bude mať menšiu a väčšiu variantu. Obe dátové sady popisujú celú štruktúru kartézskeho genetického programu, ale iným spôsobom. Pri experimentovaní sa bude pozorovať schopnosť učenia nad týmito dátovými sadami. Z dôvodu výpočtovej náročnosti sa na väčšej dátovej sade natrénujú len najlepšie modely. Dátové sady budú počas jednotlivých experimentov rozdelené do tréningovej, validačnej a testovacej dátovej sady.

V štandardnej textovej reprezentácii kartézskych genetických programov sa nachádza príliš veľa číselných hodnôt, ktoré reprezentujú indexy uzlov, funkcií, adresy vstupov a uzlov. Keď sa táto textová reprezentácia pomocou tokenizácie prevedie do číselnej podoby, tak sa indexy uzlov, funkcií, adresy nebudú od seba, s výnimkou pozičnej informácie, odlišovať a model by sa z týchto informácii ťažšie učil. Za týmto účelom vznikla motivácia vytvoriť kompaktnú a rozšírenú dátovú sadu, ktorá prináša štrukturovanejšiu textovú reprezentáciu kartézskych genetických programov.

Štandardná textová reprezentácia

Štandardná textová reprezentácia sa skladá z troch častí: konfigurácie, uzlov a výstupov.

- Konfigurácia – `{názov jedinca,počet vstupov,počet výstupov,počet stĺpcov, počet riadkov,arita uzlov,l_back}`
- Uzol – `([index]adresa1,adresa2,index funkcie)`
- Výstup – `(adresa1)`

Príklad:

```
{filter,4,1,2,2,2,1}([4]0,2,4)([5]3,2,2)([6]4,5,3)([7]2,3,1)(6)
```

Kompaktná textová reprezentácia

Podobne ako štandardná textová reprezentácia, tak kompaktná sa skladá z troch častí: konfigurácie, uzlov a výstupov.

- Konfigurácia – `<config> názov input_size_číslo output_size_číslo num_rows_číslo num_cols_číslo arity_číslo l_back_číslo`
- Uzol – `node_číslo addr1_číslo addr2_číslo func_číslo`
- Výstup – `<output> node_číslo`

Príklad:

```
<config>
    filter
    input_size_4
    output_size_1
    num_rows_2
    num_cols_2
    arity_2
```

```
    l_back_1
<nodes>
    node_4 addr1_0 addr2_2 func_4
    node_5 addr1_3 addr2_2 func_2
    node_6 addr1_4 addr2_5 func_3
    node_7 addr1_2 addr2_3 func_1
</nodes>
<outputs>
    <output> node_6
</outputs>
```

Rozšírená textová reprezentácia

Podobne ako štandardná textová reprezentácia, tak rozšírená sa skladá z troch častí: konfigurácie, uzlov a výstupov.

- Konfigurácia – `<config> <name> názov </name> <input_size> počet vstupov </input_size> <output_size> počet výstupov </output_size> <num_rows> počet riadkov </num_rows> <num_cols> počet stĺpcov </num_cols> <arity> arita </arity> <l_back> l_back </l_back> </config>`
- Uzol – `<node> <index> index </index> <addr1> adresa 1 </addr1> <addr2> adresa 2 </addr2> <func> index funkcie </func> </node>`
- Výstup – `<output> adresa </output>`

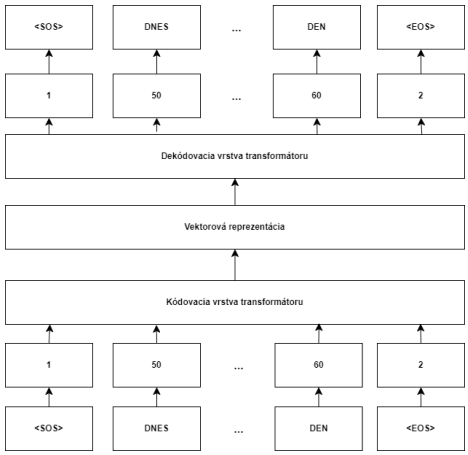
Príklad:

```
<config>
    <name> filter </name>
    <input_size> 4 </input_size>
    <output_size> 1 </output_size>
    <num_rows> 2 </num_rows>
    <num_cols> 2 </num_cols>
    <arity> 2 </arity>
    <l_back> 1 </l_back>
</config>
<nodes>
    <node>
        <index> 4 </index>
        <addr1> 0 </addr1>
        <addr2> 2 </addr2>
        <func> 4 funkcie </func>
    </node>
    <node>
        <index> 5 </index>
        <addr1> 3 </addr1>
        <addr2> 2 </addr2>
        <func> 2 </func>
    </node>
```

```
        <node>
            <index> 6 </index>
            <addr1> 4 </addr1>
            <addr2> 5 </addr2>
            <func> 3 </func>
        </node>
        <node>
            <index> 7 </index>
            <addr1> 2 </addr1>
            <addr2> 3 </addr2>
            <func> 1 </func>
        </node>
    </nodes>
    <outputs>
        <output> 6 </output>
    <outputs>
```

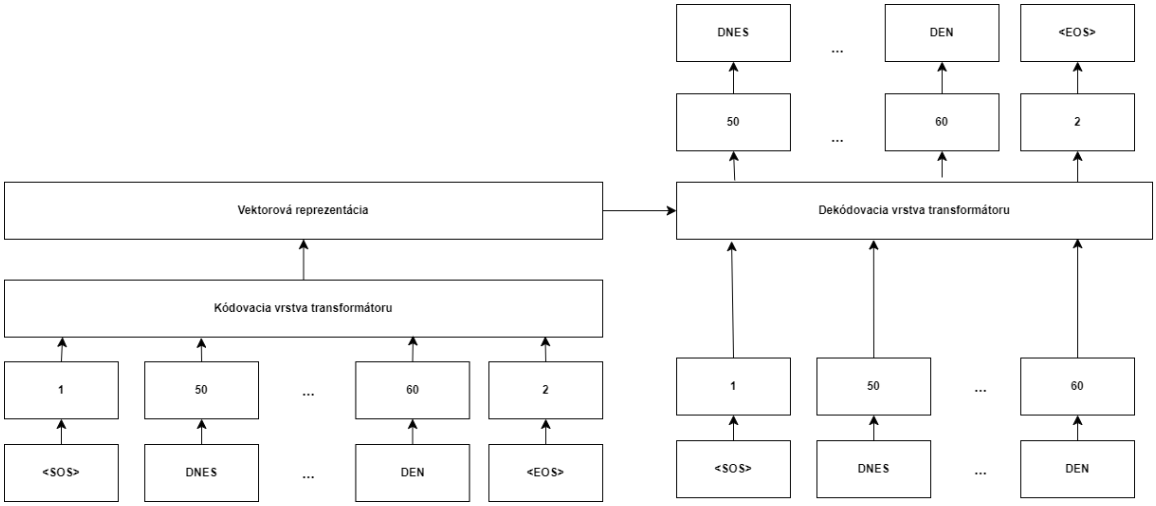
6.3 Proces vytvárania reprezentácie

Reprezentácie budú vytvorené pomocou architektúry autoenkóder. Ako kodér a dekodér bude použitý transformátor. Pri transformátore je možné použiť predtrénované modely, ale kvôli špecifickosti dát by to nemuselo byť vhodné, a preto sa v tejto práci bude transformátor učiť od začiatku aj napriek tomu, že je to veľmi výpočtovo náročné trénovanie. Transformátor sa bude učiť na dvoch úlohách. Prvá úloha je úloha rekonštrukcie pôvodného vstupu zobrazená na obr. 6.2.



Obr. 6.2: Obrázok znázorňuje proces rekonštrukcie pôvodnej sekvencie. Každý token (slovo) sa tokenizuje pomocou dopredu vytvorenej gramatiky na číslo. Táto sekvencia čísel vstupuje do kódovacej vrstvy transformátoru a výsledkom sú vektorové reprezentácie každého tokenu. Dekódovacia vrstva na základe týchto vektorových reprezentácií predikuje číselnú reprezentáciu tokenov, ktoré sú následne pomocou detokenizácie prevedené späť do textovej podoby.

Druhá úloha je zameraná na predikciu nasledujúceho tokenu v sekvencii. Táto úloha je znázornená na obr 6.3.



Obr. 6.3: Obrázok znázorňuje proces predikcie nasledujúceho tokenu. Každý token (slovo) sa tokenizuje pomocou dopredu vytvorenej gramatiky na číslo. Táto sekvencia čísel vstupuje do kódovacej vrstvy transformátoru a výsledok sú vektorové reprezentácie každého tokenu. Dekódovacia vrstva na základe týchto vektorových reprezentácií a do teraz vygenerovaných tokenov predikuje nasledujúci token v sekvencii.

V stručnosti sa teda proces vytvárania reprezentácie kandidátneho riešenia najskôr transformuje do vybranej novej textovej reprezentácie. Výber závisí od dát, nad ktorými je zvolený transformátor natrénovaný. Kandidátne riešenie reprezentované novou textovou reprezentáciou sa pomocou tokenizácie rozdelí na tokeny (slová), ktoré sú v rámci tohto procesu pomocou gramatiky prevedenú na číselnú reprezentáciu. Táto číselná reprezentácia sa pomocou natrénovaného transformátoru zakóduje do vektorovej reprezentácie, kde každý token je reprezentovaný jedným vektorom.

Schopnosť rekonštruovať vstup bude posúdená podľa mikro–spriemerovania presnosti, makro–spriemerovania presnosti a váženého–spriemerovania presnosti. Tieto metriky rozširujú základnú rovnicu pre presnosť 6.1 o viac tried. V nasledujúcich rovniciach znak N značí počet tried.

$$\text{Presnosť}_{\text{Mikro-priemer}} = \frac{1}{N} \sum_{c=0}^{N-1} \text{Presnosť}_c \tag{6.4}$$

$$\text{Presnosť}_{\text{Makro-priemer}} = \frac{\sum_{c=0}^{N-1} TP_c}{\sum_{c=0}^{N-1} (TP_c + FP_c)} \tag{6.5}$$

$$\text{Presnosť}_{\text{Vážený-priemer}} = \sum_{c=0}^{N-1} \left(\frac{\text{Počet}_c}{\text{Celkový počet}} \times \text{Presnosť}_c \right) \tag{6.6}$$

Po natrénovaní jednotlivých modelov bude môcť byť kodér a dekodér od seba oddelení a budú môcť fungovať plne samostatne pri jednotlivých úlohách.

6.4 Využitie vzniknutých reprezentácií

Po natrénovaní transformátorov na zvolených úlohách príde čas na ich využitie. Jeden z experimentov, na ktoré sa táto práca zameria bude 2D vizualizácia na vzorku dát z vali-

dačnej dátovej sady kartézskych genetickým programov. Vzorka dát sa postupne pomocou kódovacej vrstvy transformátora zakóduje. Výsledkom je vektor pre každý token v danej sekvencii. Výstupné vektory sa pomocou priemernej agregácie agregujú. Výsledný agregovaný vektor reprezentuje jedného jedinca. Keď týmto procesom budú agregované všetky dáta zo vzorku validačnej dátovej sady, tak sa najskôr tieto agregované vektory normalizujú a následne sa na ne použije metóda Uniform Manifold Aproximation and Projection (UMAP) [41]. Jedná sa o algoritmus strojového učenia, ktorý sa používa na redukciu dimenzionality. Metóda UMAP vytvára nad bodmi v priestore vysoko rozmerný graf, pomocou ktorého sa snaží optimalizovať nízko rozmerný graf tak, aby mu bol čo najviac štrukturálne podobný a zachoval tak globálnu aj lokálnu štruktúru dát. Výsledkom tejto metódy sú teda vektory v nižšej dimenzii, ktoré reprezentujú pôvodné dáta. V tejto práci to budú vektory o veľkosti dimenzie dva. Výsledné vektory sa premietnu do grafu v závislosti na type jedinca (či sa jedná o filter alebo detektor šumu). Následne sa do týchto vizualizácii premietne aj hodnota fitness, kde je hypoteticky možné, že niektoré zhluky budú obsahovať jedincov, ktorí predstavujú lepšiu fitness hodnotu. Toto zistenie by otvorilo nové experimentálne možnosti. Bolo by možné generovať jedincov z týchto oblastí vektorového priestoru. Mohli by vzniknúť nové genetické operátory, ktoré by riadili evolúciu k týmto oblastiam. Jedno z využití vytvorených modelov je ten, že by dokázali byť použité aj pri inom probléme, ktorý by obsahoval rovnaký alebo menší počet vstupov a obsahoval by tie isté funkcie alebo ich podmnožinu. Ak by ale existoval problém, ktorý by mal vstupy alebo funkcie navyše, je možné model jemne doladiť na špecifický problém malou vzorkou dát. V prípade, že by bol tento model natrénovaný na viacerých problémoch, tak by sa mohlo dosiahnuť lepšej generalizácie. Vzniknuté reprezentácie by mohli byť použité aj na predikciu fitness. Taktiež by mohli vzniknúť nové genetické operátory, ktoré by priamo pracovali s novo vzniknutými reprezentáciami.

Táto práca sa zameria na preskúmanie dvoch nových mutačných operátorov a predikciu fitness pomocou viacvrstvového perceptrónu. Prvá mutácia sa zameria na priame upravovanie vektorových reprezentácii génov a druhá na výber génov pre mutáciu na základe podobnosti génov vo vektorovej reprezentácii.

Mutácia priamou úpravou vektorovej reprezentácie génov

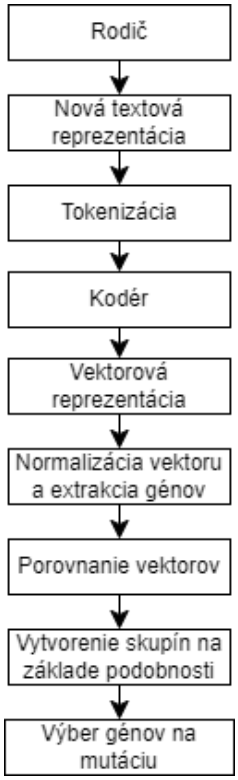
Počas evolučného procesu kartézskeho genetického programovania sa nahradí štandardná mutácia novou mutáciou, ktorá bude modifikovať vektorovú reprezentáciu génov vytvorenú pomocou kodéru transformátora. Pred použitím novej mutácie sa najskôr rodič transformuje do vektorovej reprezentácie podľa postupu v kapitole 6.3. Samotná mutácia modifikuje tieto vektory v každej z ich dimenzií náhodným malým číslom zo zvoleného intervalu. V poslednom kroku sú upravené vektorové reprezentácie génov pomocou dekodéru dekódované do číselnej reprezentácie, ktorá je v rámci tohto procesu pomocou gramatiky prevedená späť na jednotlivé tokeny v textovej reprezentácii kartézskeho genetického programu. Následne sa kontroluje validita vzniknutej textovej reprezentácie a zostavuje sa de novo vzniknutý jedinec. Metóda je znázornená na obr. 6.4.

Obr. 6.4: Na obrázku je znázornený proces nového mutačného operátora úpravou priamo vektorovej reprezentácie kartézskeho genetického programu.

Táto metóda bude porovnaná s náhodnou mutáciou a single.

Výber génov na základe podobnosti vektorov

Proces evolúcie kartézskeho genetického programovania prebieha štandardne až do bodu mutácie vybraného rodiča. Vybraný rodič sa najskôr prevedie na vektorovú reprezentáciu tokenov a to podľa postupu popísaného v kapitole 6.3. Pomocou masky sa z vektorovej reprezentácie tokenov extrahujú vektorové reprezentácie len samotných génov. Maska bude vytvorená automaticky a to tak, že na pozície tokenov, ktoré reprezentujú gény bude vložená hodnota `True` inak `False`. V kompaktnej reprezentácii sa jedná o pozície, kde sa nachádzajú tokeny `addr1`, `addr2` a `func` V rozšírenej textovej reprezentácii sa jedná o všetky pozície čísel v rámci uzlov. Výstupné gény boli opomenuté. Tieto masky budú vytvorené automaticky pre veľkosť kartézskej mriežky 8×8 a budú vložené do kódu ako konštanty, aby sa nemuseli znovu počítať. Pre inú veľkosť kartézskej mriežky by sa museli znovu vygenerovať. Na extrahované vektory sa použije normalizácia. Normalizované vektory sa porovnajú medzi sebou a sú vybrané dvojice na základe ich podobnosti. Z vybraných dvojíc sú vytvorené skupiny navzájom podobných vektorov. Vytvorené skupiny obsahujú už len indexy vektorov. V poslednom kroku sa z každej skupiny vyberie náhodne jeden index génu pre mutáciu. Ak náhodou je vybraných menej ako 5 génov pre mutáciu, tak sa zvyšné vyberú náhodne. Pri každej mutácii má výstupný gén 5 % pravdepodobnosť na mutáciu. Metóda výberu génov na základe podobnosti vektorov je znázornená na obr. 6.5



Obr. 6.5: Na obrázku je znázornený proces výberu génov pre mutáciu za použitia podobnosti vektorovej reprezentácie.

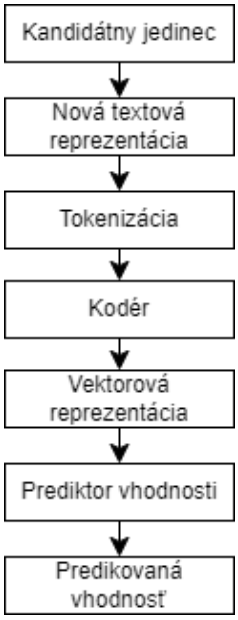
Táto metóda bude porovnaná s náhodnou mutáciou, single a mutáciou založenou na podobnosti hodnôt génov. Mutácia založená na podobnosti hodnôt génov zaznamenáva skupiny hodnôt génov pre jednotlivé vstupy uzlov a funkcie. Z každej z týchto skupín, ktoré majú veľkosť väčšiu ako 1, sú náhodne vybrané gény pre mutáciu. Podobne, ako pri mutácii založenej na podobnosti vektorov, sa pri nižšom počte génov zvyšné gény náhodne vyberú a pri každej mutácii má výstupný gén pravdepodobnosť 5 % na mutáciu.

Predikcia fitness kandidátneho jedinca

Prediktory fitness sa používajú na zrýchlenie evolučného procesu, pri ktorom sa vynechá často zdĺhavý výpočet fitness hodnoty a nahradí sa modelom, ktorý dokáže na základe príznakov kandidátnych riešení predikovať ich fitness hodnotu.

V tejto práci sa preskúma, či dokážu byť vytvorené reprezentácie kandidátnych riešení (pomocou procesu, ktorý je popísaný nižšie) použité ako príznaky pre tréning prediktoru fitness.

Proces evolúcie kartészkeho genetického programovania beží štandardne až do bodu evaluácie kandidátneho jedinca. Namiesto evaluácie sa kandidátny jedinec prevedie do vektorovej reprezentácie jednotlivých tokenov ako to bolo popísané v kapitole 6.3. Tieto vektory sa normalizujú a priemernou agregáciou agregujú. Agregovaná vektorová reprezentácia kandidátneho jedinca sa dá na vstup prediktoru, ktorý vráti predikovanú fitness hodnotu kandidátneho jedinca. Metóda predikcie fitness je znázornená na obr. 6.6. Ako prediktor fitness v tejto práci bude použitý viacvrstvový perceptrón.



Obr. 6.6: Na obrázku je znázornený proces predikcie fitness za použitia vektorovej reprezentácie kandidátneho jedinca.

Predikcia bude porovnaná naprieč viacerými metrikami ako je MSE (rovnica 6.3), MAE a Spearmanov korelačný koeficient.

Kapitola 7

Implementácia

V tejto kapitole je stručne popísaná implementácia návrhu riešenia, informácie o použitých technológiách a technikách.

7.1 Použité technológie

Tento projekt bol implementovaný v jazyku Python 3.10 [20]. Pre priebežné ukladanie výsledkov návrhu obrazových filtrov a tréningu hlbokých neurónových sietí bola použitá knižnica wandb [4]. Wandb je služba, ktorá je navrhnutá na priebežné sledovanie výsledkov experimentovania v strojovom učení. Počas trénovania môže služba zaznamenávať parametre sietí, výstupné metriky a dáta o využitých zdrojoch na výpočtovom systéme. Všetky zbierané dáta sa graficky zobrazujú, takže je jednoduchšie odhaliť chyby a vykonať potrebnú úpravu. Knižnica taktiež ponúka implementáciu pre hľadanie optimálnych hyperparametrov (wandb sweeps). Pre načítanie a predspracovanie obrázkov bola použitá knižnica OpenCV [7]. Pre návrh, trénovanie, validáciu a evaluáciu modelov bola použitá knižnica PyTorch [49]. Pre efektívnejšiu manipuláciu s dátami boli použité knižnice numpy [26] a pandas [42]. Na vizualizáciu dát boli použité knižnice seaborn [68] a matplotlib [30]. Medzi ďalšie použité knižnice patria skimage [67], scipy [66] a sklearn [50]. Ako verzovací systém bol použitý git [11].

7.2 Štruktúra projektu

Projekt je implementovaný čo najviac modulárne, aby sa dala do projektu jednoducho pridávať nová alebo upravovať stará funkcionalita. Projekt sa skladá z nasledujúcich súborov:

- `main.py` – Obsahuje implementáciu pre spustenie návrhu obrazových filtrov s detektormi šumu.
- `transfomer_training.py` – Implementuje algoritmus pre učenie jednotlivých úloch transformátoru.
- `predictor_training.py` – Implementuje algoritmus pre učenie prediktoru.
- `ImageProcessor.py` – Obsahuje implementáciu triedy, ktorá obsahuje implementácie funkcií na predspracovanie obrázkov pre návrh obrazových filtrov. Obsahuje dve statické funkcie. Jedna funkcia vytvorí všetky posuvné okná a druhá ich zorganizuje

takým spôsobom, aby sa dali všetky posuvné okná spracovať naraz kartézskym genetickým programom.

- `CartesianGeneticProgramming.py` – Implementácia triedy, ktorá implementuje algoritmus kartézskeho genetického programovania.
- `Individual.py` – Implementácia triedy, ktorá obsahuje implementáciu jedinca v populácii. Obsahuje funkcie na vytvorenie jedinca, mutáciu, kontrolu validity jedinca, evaluáciu, vyhodnotenie fitness hodnoty, nájdenie aktívnych uzlov a transformácie z iných a do iných textových reprezentácii.
- `Node.py` – Implementuje dátovú triedu, ktorá uchováva informácie o jednotlivých uzloch v jedincoch.
- `CoevolutionCartesianGeneticProgramming.py` – Implementuje triedu, ktorá implementuje algoritmus kompozičnej koevolúcie v kartézskom genetickom programovaní. Implementácia využíva už do teraz implementované moduly a rozširuje ich o interakciu medzi jedincami jednotlivých populácií.
- `ColumnInputRestrictions.py` – Implementuje dátovú triedu, ktorá obsahuje informácie o validných hodnotách, ktoré môže každý gén v jedincoch v populácii nadobúdať.
- `FitnessFunctions.py` – Implementácia statických funkcií, ktoré implementujú jednotlivé fitness funkcie, ktoré sa dajú priraďovať ku jednotlivým konfiguráciám jedincov.
- `Mutations.py` – Implementácia statických funkcií, ktoré implementujú jednotlivé mutácie. Tieto mutácie sa dajú následne priraďovať ku konfigurácii jedincov.
- `Tokenizer.py` – Implementácia triedy, ktorá vytvára gramatiku zo vstupnej textovej reprezentácie a kóduje resp. dekóduje ju do číselnej resp. textovej podoby.
- `MLP.py` – Implementácia viacvrstvového perceptrónu.
- `Transformer.py` – Implementácia transformátoru pre úlohu predikcie nasledujúceho tokenu.
- `TransformerAutoEncoder.py` – Implementácia transformátoru pre rekonštrukčnú úlohu.
- `utils.py` – Obsahuje implementáciu pomocných funkcií.

7.3 Argumenty programov

V tejto kapitole sú popísané argumenty jednotlivých skriptov. Argumenty skriptu `main.py`:

- `-runs` – Počet samostatných behov programu.
- `-runs_label` – Označenie behov programu.
- `-vocab_file` – Cesta ku uloženej gramatike.
- `-model_path` – Cesta ku natrénovanému modelu.

- `-noise_ratio` – Zvolená intenzita šumu. Len pre štatistické účely, hodnota sa bude objavovať v štatistikách.
- `-mutation` – Druh mutácie (random, embedding_similarity, value_similarity, single).
- `-generations` – Počet generácií.
- `-generate_dataset` – Príznak či sa má generovať dátová sada jedincov.
- `-interaction_interval` – Počet generácií, po ktorých sa má vykonávať interakcia medzi populáciami.
- `-result_dir` – Cesta k priečinku kde budú uložené výsledky evolúcie.
- `-save_interval` – Interval po koľkých generáciach sa budú ukladať jedinci z populácie. Používa sa ak je nastavený príznak generovania dátovej sady.
- `-use_wandb` – Príznak použitia wandb na zaznamenávanie dát
- `-wandb_project_name` – Názov wandb projektu
- `-wandb_entity_name` – Autor wandb projektu.
- `-wandb_run_name` – Názov behu.
- `-train_dataset_targets` – Cesta ku dátovej sade očakávaných výstupov.
- `-train_dataset_inputs` – Cesta ku dátovej sade poškodených vstupov.
- `-device` – Aké zariadenie používať pri používaní modelov.
- `-filter_predictor_path` – Cesta k prediktoru fitness pre filter.
- `-detector_predictor_path` – Cesta k prediktoru fitness pre detektor.
- `-experiment_name` – Názov experimentu. Bude použité vo wandb.

Argumenty skriptu `transformer_traning.py`:

- `-lr` – Learning rate.
- `-wd` – Weight decay.
- `-architecture` – Zvolená úloha.
- `-epochs` – Počet epoch.
- `-embed_dim` – Veľkosť vektorovej reprezentácie
- `-batch_size` – Veľkosť batchu pre trénovanie.
- `-num_heads` – Počet multi-head attention hláv.
- `-num_layers` – Počet vrstiev transformátoru
- `-ff_dim` – Veľkosť dimenzie plne prepojenej vrstvy
- `-dropout` – Pomer dropoutu.

- `-verbose` – Zobrazovanie procesu trénovania.
- `-path_to_dataset` – Cesta k dátovej sade.
- `-path_to_vocabulary` – Cesta ku gramatike.
- `-model_output_dir` – Cesta pre uloženie najlepšieho modelu.
- `-patience` – Trpezlivosť pri kontrole skorého ukončenia trénovania.
- `-use_wandb` – Použitie wandb.
- `-wandb_project_name` – Názov wandb projektu
- `-wandb_entity_name` – Autor wandb projektu.
- `-wandb_run_name` – Názov behu.
- `-experiment_name` – Názov experimentu v inštancii wandb.

Argumenty skriptu `predictor_training.py`:

- `-use_wandb` – Použitie wandb.
- `-wandb_project_name` – Názov wandb projektu
- `-wandb_entity_name` – Autor wandb projektu.
- `-wandb_run_name` – Názov behu.
- `-result_dir` – Priečinok pre uloženie výstupných modelov.
- `-experiment_name` – Názov experimentu v inštancii wandb.

7.4 Implementácia transformátoru

V rámci tejto práce bol transformátor implementovaný pomocou vstavaných implementovaných vrstiev transformátoru z knižnice PyTorch. Implementácia zahŕňa vytvorenie vlastného modelu pozostávajúceho z kódovacej a dekódovacej časti. Implementácia modelu pozostáva z nasledujúcich tried:

- PositionalEncoding – Jedná sa o implementáciu pozičného kódovania, ktoré sa používa pre pridanie pozičnej informácie do vstupných vektorov. Implementácia pozičného kódovania bola prevzatá z [63].
- TransformerEncoder – Jedná sa o kódovaciu časť transformátora a zahŕňa: embedding vrstvu (pre preklad slov do vektorov pevnej dĺžky), pozičné kódovanie, kódovaciu vrstvu transformátora, ktorá pozostáva z viacerých takýchto vrstiev (PyTorch implementácia `torch.nn.TransformerEncoderLayer`). Výstup je normalizovaný pomocou `torch.nn.LayerNorm`.

- TransformerDecoder – Jedná sa o dekódovaciu časť transformátora, ktorá spracováva výstupy z kodéru spolu s cieľovými dátami (pri úlohe rekonštrukcie pôvodného vstupu neboli cieľové dáta použité a boli použité informácie len z kódovacej časti transformátoru). Podobne ako kodér, obsahuje: embedding vrstvu a pozičné kódovanie. Ďalej obsahuje dekódovaciu vrstvu transformátora, ktorá tiež pozostáva z viac vrstiev (PyTorch implementácia `torch.nn.TransformerDecoderLayer`. Výstup je normalizovaný a následne transformovaný na predikcie slov pomocou lineárnej vrstvy.
- Transformer – táto trieda spája kodér a dekodér.

Transformátor bol trénovaný na všetkých úlohách od začiatku a bez predchádzajúceho predtrénovania na inej úlohe.

7.5 Vyhľadávanie optimálnych hyperparametrov

Na vyhľadávanie optimálnych hyperparametrov bola použitá funkcionalita knižnice wandb, ktorá ponúka funkcionalitu wandb sweeps. Wandb sweeps automatizuje vyhľadávanie najlepšej kombinácie hyperparametrov tým, že vykoná viacero tréningových experimentov s rôznymi nastaveniami hyperparametrov. Funguje to tak, že užívateľ si definuje buď priamo v kóde alebo pomocou konfiguračného súboru všetky hyperparametre, ktoré chce preskúmať a ich príslušné hodnoty. Zostáva nastaviť prehľadávací algoritmus. Je na výber z troch prehľadávacích algoritmov a to je náhodné, mriežkové alebo bayesovská optimalizácia. V tejto práci bolo použité náhodné prehľadávanie. Priebežné výsledky tréningových experimentov sú zbierané a vizualizované v reálnom čase.

7.6 Pomocné skripty

Mimo zmienené skripty boli implementované ďalšie dva pomocné skripty `experiments.py` a `experiments_transformer_model_training.py`. Prvý skript spúšťa experimenty pre návrh obrazových filtrov a druhý skript spúšťa experimenty na trénovanie transformátoru.

7.7 Jupyter notebooky pre experimentovanie

Jednotlivé experimenty popísané v kapitole 8, sú implementované v jupyter notebookoch (označených indexom experimentu) pre jednoduchšiu úpravu a vizualizáciu získaných dát.

7.8 Chybová funkcia cross–entropy

Počas učenia transformátoru bola použitá chybová funkcia cross–entropy. Cross–entropy počíta rozdiel medzi dvomi pravdepodobnostnými distribúciami: očakávaným výstupom a predikovanými výstupmi z modelu. Cieľom je minimalizovať rozdiel medzi týmito dvomi distribúciami a efektívne riadiť model aby produkoval čo najviac presné výstupy. Rovnica pre chybovú funkciu cross–entropy:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c}\log(\hat{y}_{i,c}) \tag{7.1}$$

kde C je počet tried, $y_{i,c}$ je binárny indikátor či je trieda c správna klasifikácia pre pozorovanie i a $\hat{y}_{i,c}$ je predikovaná pravdepodobnosť, že pozorovanie i patrí do triedy c.

7.9 Technika postupného znižovania miery učenia

Počas učenia transformátoru bola použitá technika postupného znižovania miery učenia. Bola použitá technika z knižnice PyTorch pod názvom `ReduceLROnPlateau`. Táto technika znižuje mieru učenia sledovaním validačnej chyby. Ak nebolo zaznamenané zlepšenie validačnej chyby určitý počet epoch, tak je miera učenia zmenšená o dopredu vybraný faktor.

7.10 Technika skorého ukončenia trénovania

Technika skorého ukončenia trénovania funguje na podobnom princípe ako znižovanie miery učenia. Ak validačná chyba je horšia ako v predchádzajúcej epoche, tak sa načíta do teraz najlepší model. Ak sa validačná chyba nezlepší určitý počet po sebe nasledujúcich epoch, tak je trénovanie modelu ukončené.

Kapitola 8

Experimenty

Táto kapitola overuje funkčnosť návrhu obrazových filtrov spolu s detektormi šumu a skúma automatické vytváranie reprezentácií a ich využitie pomocou sady experimentov. V rámci experimentálneho vyhodnotenia práce je zmienená fitness hodnota vždy vypočítaná kombináciou najlepšej kombinácie (kompozície) filtra a detektora z pohľadu PSNR.

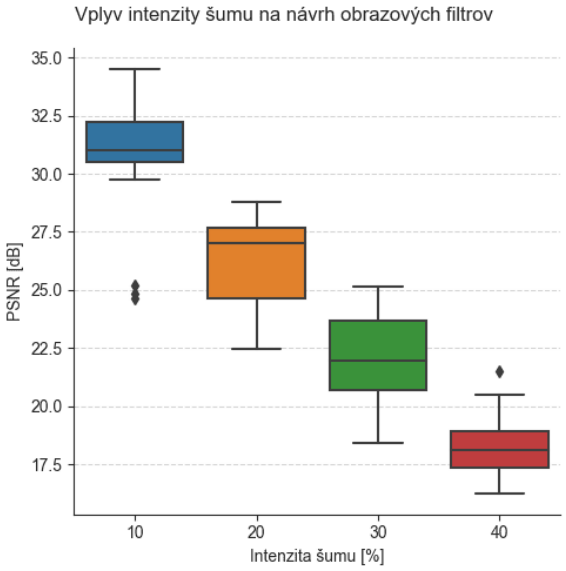
8.1 Overenie funkčnosti návrhu obrazových filtrov

Na vytvorenie dátovej sady pre učenie hlbokých modelov strojového učenia bola vybraná úloha návrhu obrazových filtrov spolu s detektormi šumu. V kapitole 4.2 je popísaný postup návrhu experimentu pre získanie dát z vybranej úlohy. Táto podkapitola sa zaoberá výsledkami spojenými s návrhom obrazových filtrov spolu s detektormi šumu a získanými jedincami pre vytvorenie dátových sád. Návrh obrazových filtrov spolu s detektormi šumu bol spustený 30 krát po 30000 generácií s parametrami, ktoré sú v tabuľke 6.1 pre obr. 6.1 poškodený 10, 20, 30 a 40 percentným šumom. V obr. 8.1 je znázornená výsledná fitness hodnota v závislosti na jednotlivých intenzitách šumu. Z výsledkov je zrejmé, že navrhované kompozície majú lepšiu fitness pri nižších intenzitách šumu. V tabuľke 8.1 sú vidieť špecifické hodnoty pre najlepšie, priemerné a najhoršie navrhnuté kompozície.

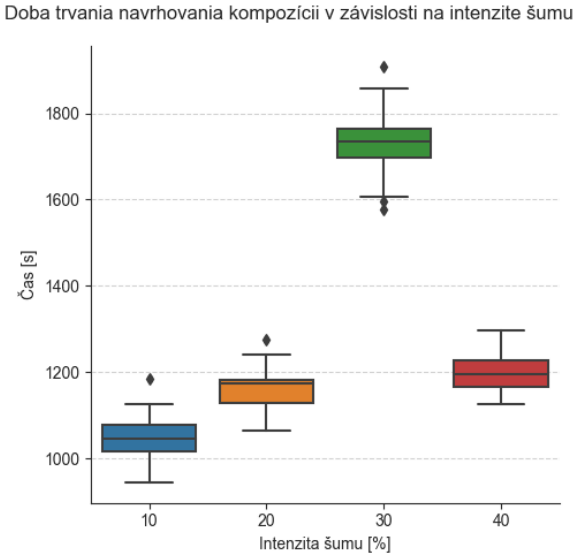
			Fitness [dB]	
Intenzita šumu	10 %	20 %	30 %	40 %
Minimum	24.655	22.479	18.413	16.253
Priemer	31.038	26.267	21.972	18.270
Maximum	34.500	28.808	25.123	21.503

Tabuľka 8.1: Minimálna, priemerná a maximálna fitness v závislosti na intenzite šumu.

Hypotéza: Doba navrhovania kompozícií závisí na intenzite šumu. Čím vyššia je intenzita šumu, tým dlhšie bude trvať návrh. Hypotéza vznikla na základe toho, že intenzita šumu ovplyvňuje počet tréningových dát filtra. Táto hypotéza sa ukázala ako pravdivá, a to je vidieť na nasledujúcom obrázku 8.2. Na obrázku sa pri 30 % intenzite šumu vyskytuje anomália z trendu doby trvania návrhu. Dôvod vzniku tejto anomálie nebol ďalej skúmaný, ale je možné, že v daný čas boli na zvolenom servery vyťažené výpočtové zdroje.

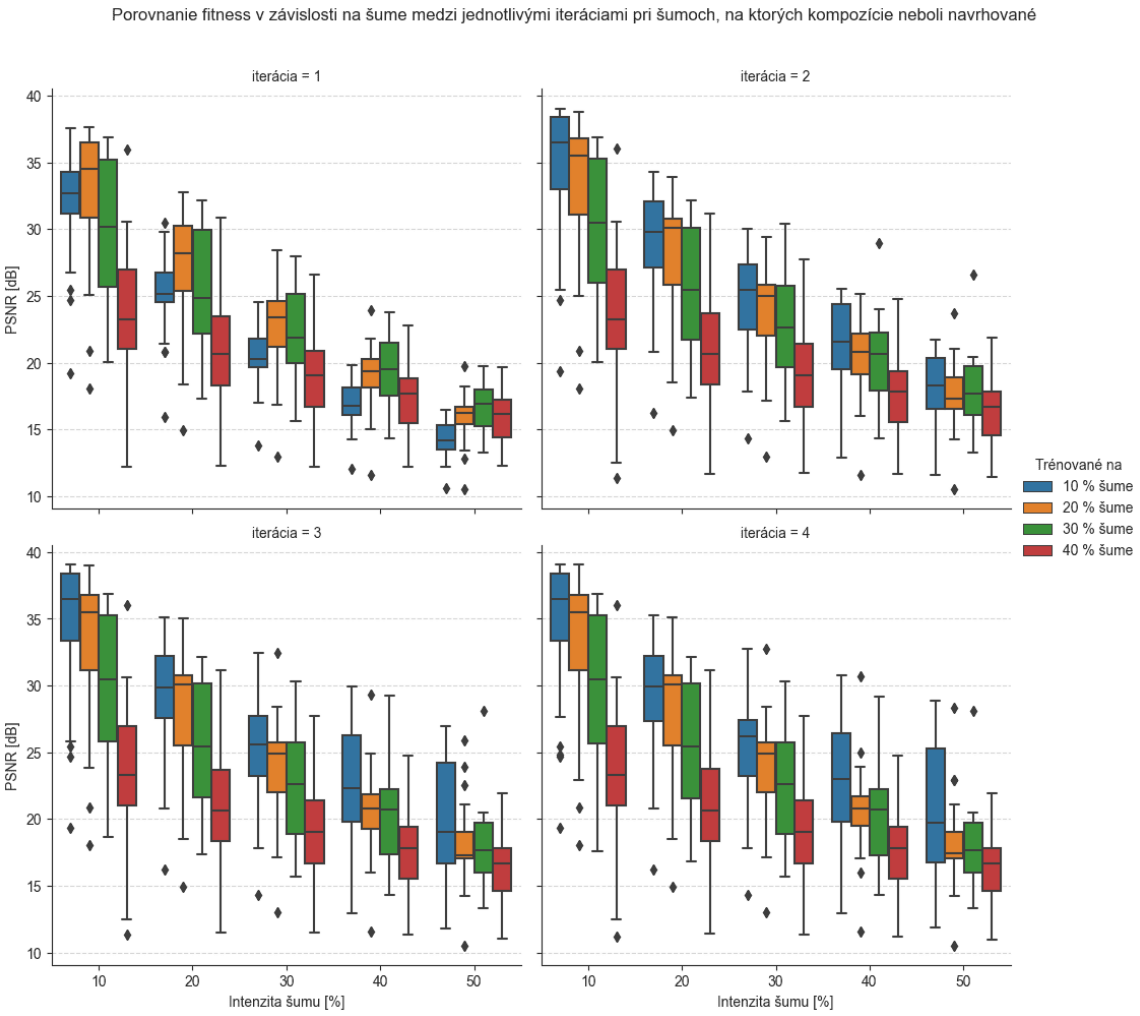


Obr. 8.1: Zobrazenie závislosti medzi fitness a intenzitou šumu pri návrhu obrazových filtrov spolu s detektormi šumu. Graf zobrazuje výslednú fitness hodnotu z pohľadu PSNR každého behu programu pre jednotlivé intenzity šumu.



Obr. 8.2: Doba navrhovania obrazových filtrov spolu s detektormi šumu v závislosti na intenzite šumu.

Jeden z ďalších experimentov je porovnanie fitness v závislosti na šume medzi iteráciami filtrovania pri šumoch, na ktorých kompozície neboli navrhované. Hypotéza predpokladá, že kompozície, ktoré boli navrhnuté na nižšej intenzite šumu, budú mať lepšie výsledky. Experiment bol spustený na piatich testovacích obrázkoch. Oproti šumom, na ktorých boli kompozície navrhnuté, bola pridaná 50 % intenzita šumu. Testovacie obrázky sú filtrované jednotlivými kompozíciami v štyroch iteráciách. Na obr. 8.3 sú znázornené výsledky tohto experimentu.



Obr. 8.3: Obrázok znázorňuje porovnanie fitness hodnoty v závislosti na šume medzi jednotlivými iteráciami pri šumoch, na ktorých kompozície neboli navrhované.

Po prvej iterácii je zrejmé, že kompozície ktoré boli navrhované na 20 % a 30 % intenzite šumu majú skoro v každom prípade prevahu pred ostatnými kompozíciami. V nasledujúcich iteráciách sa vyskytuje zlepšenie fitness naprieč všetkými kompozíciami, ktoré boli navrhnuté na nižšej intenzite šumu. Kompozície, ktoré boli navrhnuté na 40 % intenzite šumu, vykazujú veľmi malé zlepšenie fitness, naopak kompozície navrhnuté na 10 % intenzite šumu po štyroch iteráciách jasne prekonávajú výsledky všetkých ostatných kompozícií. Na obr. 8.4 sú vidieť výsledky filtrovania jedného z testovacích obrázkov po štyroch iteráciách najlepšou kompozíciou v danej šumovej kategórii.

Obr. 8.4: Na prvom riadku sú vidieť poškodené obrázky 10 % – 50 % intenzitou šumu. Na druhom riadku sú vyfiltrované obrázky po štyroch iteráciách najlepšími kompozíciami v danej šumovej kategórii. Je vidieť, že najlepšie kompozície filtrov a detektorov dokážu filtrovať aj 50 % intenzitu šumu na ktorej neboli navrhované.

8.2 Vytvorenie dátových sád

Počas návrhu obrazových filtrov s detektormi šumu v prvom experimente bola vygenerovaná dátová sada filtrov a detektorov. Počet dát v dátovej sade je popísaný v tabuľke 8.2.

	Počet filtrov	Počet detektorov	Celkový počet
Pred deduplikáciou	560092	572285	1132377
Po deduplikácii	559972	572165	1132137

Tabuľka 8.2: Počet filtrov a detektorov vo vygenerovanej dátovej sade

Z vytvorenej dátovej sady boli vytvorené 3 dátové sady pre kompaktnú a rozšírenú textovú reprezentáciu jedincov. Postupy pre obe textové reprezentácie boli rovnaké. Prvé dve dátové sady boli vytvorené pre trénovanie transformátoru. Tretia dátová sada bola vytvorená pre tréning prediktoru fitness. Postup pre vytvorenie prvej dátovej sady:

1. Dátová sada je rozdelená do sady filtrov a detektorov.
2. Načíta sa každý jedinec filtrov a detektorov.
3. Sú zistené aktívne gény každého jedinca.
4. Deduplikácia filtrov a detektorov na úrovni aktívnych génov.
5. Vypočítanie fitness filtrov a detektorov na jednom testovacom obrázku pre 10 - 50 % intenzitu šumu.
6. Extrakcia filtrov, ktoré majú fitness pri 10 % intenzite šumu väčšiu alebo rovnú 15.
7. Extrakcia detektorov, ktoré majú fitness pri 10 % intenzite šumu väčšiu alebo rovnú 0.5.

8. Vytvorenie zlúčenej dátovej sady filtrov a detektorov s pridanou hodnotou typu jedinca a priemernou fitness.

Prehľad počtu filtrov a detektorov pri jednotlivých krokoch tvorby dátovej sady sa nachádza v tabuľke 8.3.

	Počet filtrov	Počet detektorov	Celkový počet
Pred deduplikáciou	560092	572285	1132377
Po deduplikácii	559972	572165	1132137
Po deduplikácii podľa aktívnych génov	198636	194216	392852
Po extrakcii	96067	111026	207093

Tabuľka 8.3: Prehľad počtu filtrov a detektorov v jednotlivých krokoch prípravy prvej dátovej sady

Druhá dátová sada sa vytvárala podobne, s tým rozdielom, že sa nerobila deduplikácia podľa aktívnych génov a taktiež sa nerobila extrakcia podľa fitness. Špecifický postup vytvorenia druhej väčšej dátovej sady:

1. Z dátovej sady sa vyberie náhodne 500000 jedincov.
2. Náhodne vybrané dáta sú rozdelené do sady filtrov a detektorov.
3. Načíta sa každý jedinec filtrov a detektorov.
4. Vypočítanie fitness hodnoty filtrov a detektorov na jednom testovacom obrázku pre 10 - 50 % intenzitu šumu.
5. Vytvorenie zlúčenej dátovej sady filtrov a detektorov s pridanou hodnotou typu jedinca a priemernou fitness.

Prehľad počtu filtrov a detektorov pri jednotlivých krokoch tvorby dátovej sady sa nachádza v tabuľke 8.4.

	Počet filtrov	Počet detektorov	Celkový počet
Pred deduplikáciou	560092	572285	1132377
Po deduplikácii	559972	572165	1132137
Po náhodnom výbere	247066	252934	500000

Tabuľka 8.4: Prehľad počtu filtrov a detektorov v jednotlivých krokoch prípravy druhej dátovej sady

Posledná dátová sada je určená pre trénovanie prediktoru fitness. Postup pre vytvorenie tejto dátovej sady:

1. Z celkovej dátovej sady sú extrahované len jedince navrhované na 10 % a 20 % šume.
2. Deduplikácia filtrov a detektorov.

3. Náhodný výber 500000 jedincov.
4. Náhodne vybrané dáta sú rozdelené do sady filtrov a detektorov.
5. Načíta sa každý jedinec filtrov a detektorov.
6. Vypočítanie fitness hodnoty filtrov a detektorov na jednom tréningovom obrázku pre 10 % intenzitu šumu.
7. Vytvorenie zlúčenej dátovej sady filtrov a detektorov s pridanou hodnotou typu jedinca a fitness.

Prehľad počtu filtrov a detektorov pri jednotlivých krokoch tvorby dátovej sady sa nachádza v tabuľke 8.5.

	Počet filtrov	Počet detektorov	Celkový počet
Pred deduplikáciou	560092	572285	1132377
Po extrakcii	282979	288728	571587
Po deduplikácii	282919	288668	571587
Po náhodnom výbere	247296	252704	500000

Tabuľka 8.5: Prehľad počtu filtrov a detektorov v jednotlivých krokoch prípravy poslednej dátovej sady pre tréning prediktorou fitness

Počas tréningu prediktoru fitness sa kvôli výpočtovej náročnosti kódovania jedincov do vektorovej reprezentácie vytvorená dátová sada zredukovala na 170000 vzorkov po 85000 filtrov a 85000 detektorov.

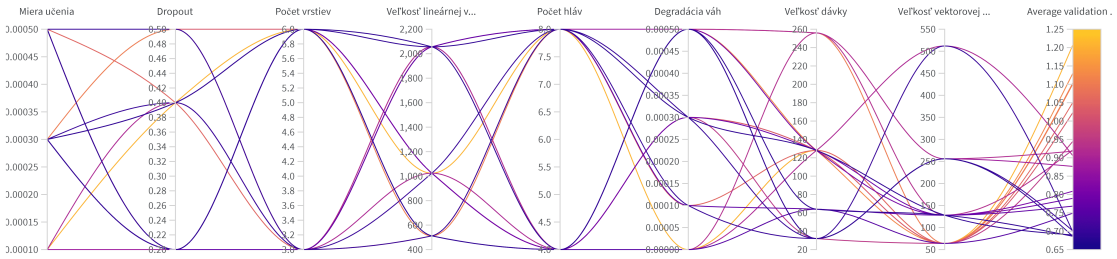
8.3 Prehľadávanie hodnôt hyperparametrov pre trénovanie transformátoru

Tento experiment sa zaoberá prehľadávaním hodnôt hyperparametrov pre trénovanie transformátoru na dvoch dátových sadách (kompaktnej a rozšírenej). V návrhu riešenia sú spomenuté dve úlohy, pomocou ktorých táto práca vytvára vektorové reprezentácie kartézskych genetických programov. Tento experiment je zameraný na druhú úlohu a to predikciu nasledujúceho tokenu. Keďže experiment je výpočtovo náročný bol spustený 20 krát po 1 epochu pre každú dátovú sadu. Chybová funkcia pre učenie transformátoru bola použitá cross entropy. Ako bolo spomenuté v návrhu pri prehľadávaní hyperparametrov bola použitá funkcionalita wandb sweeps. Výber hodnôt jednotlivých hyperparametrov bol náhodný a nezávislý na predchádzajúcich behoch. Tabuľka 8.6 obsahuje zoznam prehľadávaných parametrov pri rozšírenej dátovej sade.

Hyperparameter	Prehľadávacie hodnoty
Veľkosť dávky	32, 64, 128, 256
Veľkosť vektorovej reprezentácie	512, 256, 128, 64
Degradácia váh	0, 0.0001, 0.0003, 0.0005
Veľkosť lineárnej vrstvy	512, 1024, 2056
Miera učenia	0.0001, 0.0003, 0.0005
Dropout	0.2, 0.4, 0.5
Počet vrstiev	3, 6
Počet hláv	4, 8

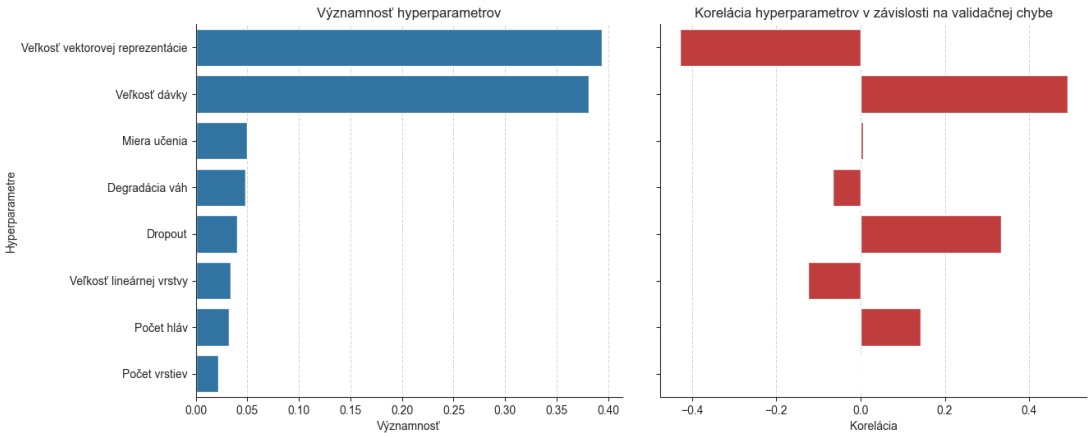
Tabuľka 8.6: Zoznam prehľadávaných hodnôt pre zvolené hyperparametre pri rozšírenej dátovej sade

Výsledky z tohto experimentu sú obsiahnuté v troch obrázkoch. Na obr. 8.5 sú zobrazené paralelné súradnice pre všetky hyperparametre a ich vplyv na priemernú validačnú chybu. Každá závislá os predstavuje jeden hyperparameter a krivky zobrazujú rôzne kombinácie hodnôt hyperparametrov a ich súvisiacu metriku validačnej chyby.



Obr. 8.5: Výsledná validačná chyba v závislosti na kombinácii parametrov pri prehľadávaní hodnôt hyperparametrov za použitia rozšírenej dátovej sady.

Pre lepšie pochopenie týchto dát bol natrénovaný model náhodného lesa (anglicky – RandomForest), kde hyperparametre boli použité ako príznaky a validačná chyba ako očakávaný výstup. Po naučení náhodného lesa sa extrahovala významnosť jednotlivých príznakov (hyperparametrov) a premietla do obr. 8.6. Tento obrázok obsahuje taktiež aj koreláciu parametrov v závislosti na validačnej chybe. Z prvého grafu sa dá vyčítať, že najviac k výsledku validačnej chyby prispievajú hyperparametre veľkosť dávky a veľkosť vektorovej reprezentácie. Korelačná analýza zobrazuje negatívnu koreláciu pre veľkosť vektorovej reprezentácie, pozitívnu pre veľkosť dávky a dropout. To znamená, že čím je veľkosť vektorovej reprezentácie vyššia, tým je validačná chyba nižšia a naopak pri veľkosti dávky a dropout je validačná chyba väčšia.



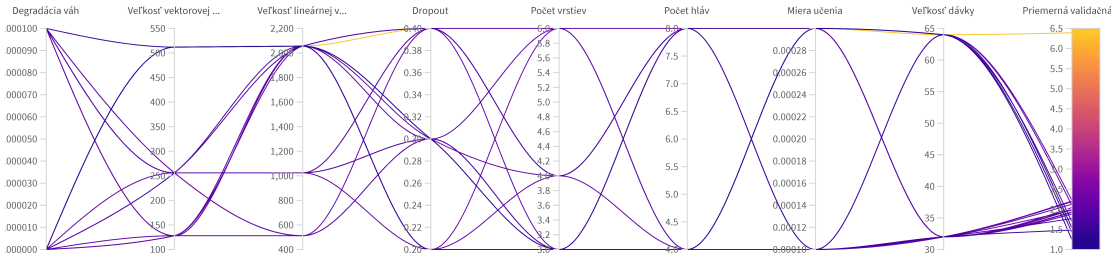
Obr. 8.6: Na pravej časti obrázku je vidieť významnosť jednotlivých parametrov v závislosti na validačnej chybe a na pravej časti je vidieť koreláciu hyperparametrov v závislosti na validačnej chybe.

Pre trénovanie transformátoru na kompaktnej dátovej sade boli na základe výsledkov z trénovania transformátoru na rozšírenej dátovej sade vybrané odlišné hodnoty na prehľadávanie (tabuľka 8.7).

Hyperparameter	Prehľadávacie hodnoty
Veľkosť dávky	32, 64
Veľkosť vektorovej reprezentácie	512, 256, 128
Degradácia váh	0, 0.0001
Veľkosť lineárnej vrstvy	512, 1024, 2056
Miera učenia	0.0001, 0.0003
Dropout	0.2, 0.3, 0.4
Počet vrstiev	3, 4, 6
Počet hláv	4, 8

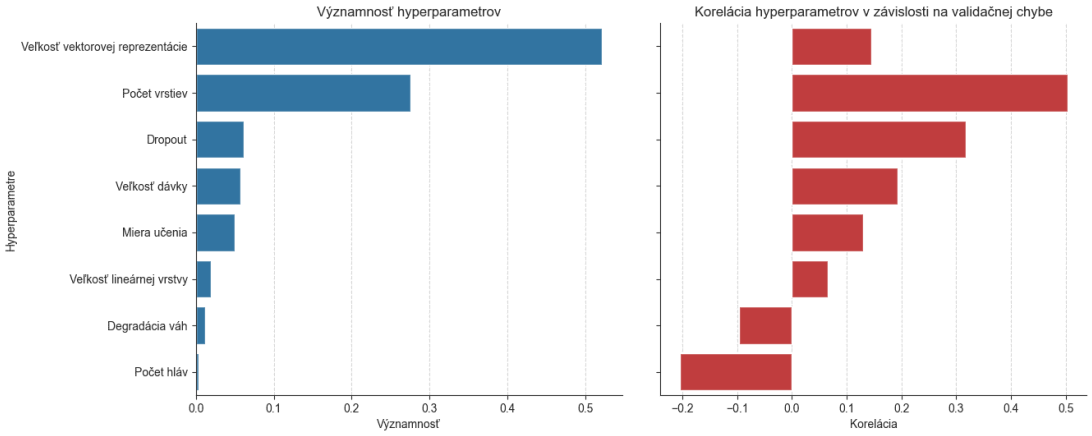
Tabuľka 8.7: Zoznam prehľadávaných hodnôt hyperparametrov pri kompaktnej dátovej sade

Výsledky tejto časti experimentu sú vyhodnotené rovnako ako pri rozšírenej dátovej sade. Podobne ako pri rozšírenej dátovej sade na obr. 8.7 sú zobrazené paralelné súradnice pre všetky hyperparametre a ich vplyv na priemernú validačnú chybu. Každá závislá os predstavuje jeden hyperparameter a krivky zobrazujú rôzne kombinácie hodnôt hyperparametrov a ich súvisiacu metriku validačnej chyby.



Obr. 8.7: Výsledná validačnej chyba v závislosti na kombinácii parametrov pri prehľadávaní vhodných hodnôt hyperparametrov za použitia kompaktnej dátovej sady.

Z prvého grafu na obr.8.8 je možné pozorovať najväčší príspevok hyperparametrov veľkosť vektorovej reprezentácie a počet vrstiev k výslednej validačnej chybe. V korelačnej analýze je významná pozitívna korelácia v závislosti na validačnej chybe pozorovateľná pri hyperparametroch veľkosti vektorovej reprezentácie, počte vrstiev, dropout, veľkosti dávky a negatívna pri počte hláv.



Obr. 8.8: Na pravej časti obrázku je vidieť významnosť jednotlivých hyperparametrov v závislosti na validačnej chybe a na pravej časti je vidieť koreláciu hyperparametrov v závislosti na validačnej chybe.

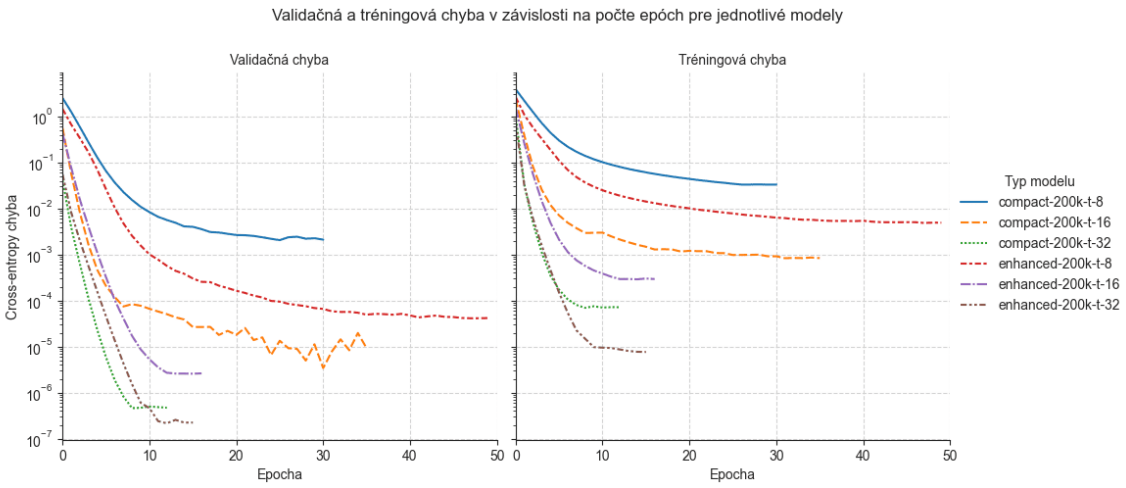
8.4 Trénovanie transformátoru

Na základe výsledkov z predchádzajúceho experimentu boli natrénované nasledujúce modely (tabuľka 8.8).

Názov modelu Hyperparameter	compact-200k-t, enhanced-200k-t	compact-200k-tp	enhanced-200k-tp	compact-500k-tp	enhanced-500k-tp
Veľkosť vektorovej reprezentácie	32, 16, 8	512, 256, 128	512, 256, 128	512	512
Veľkosť lineárnej vrstvy	512	2048	512	512	512
Miera učenia	0.0001	0.0003	0.0003	0.0003	0.0003
Veľkosť dávky	64	256	64	256	64
Degradácia váh	0	0	0	0	0
Dropout	0.2	0.2	0.2	0.2	0.2
Počet vrstiev	4	3	4	4	4
Počet hláv	8	8	8	8	8
Počet epoch	50	60	30	30	10

Tabuľka 8.8: Prehľad nastavení hyperparametrov pre jednotlivé modely. Označenia `t` a `tp` značia úlohy, na ktorých sú modely trénované. Značka `t` značí rekonštrukčnú úlohu. Značka `tp` značí predikčnú úlohu nasledujúceho tokenu. Značky 200k a 500k značia veľkosť dátovej sady.

Po počiatočnom trénovaní boli vybrané dve najlepšie riešenia z predikčnej úlohy nasledujúceho tokenu (`compact-200k-tp-512` a `enhanced-200k-tp-512`) pre trénovanie na väčšej dátovej sade. Nastavenia pre tieto dodatočné trénovania sú obsiahnuté v tabuľke 8.8 a výsledky z týchto dodatočných trénovaní sú v obr. 8.10.

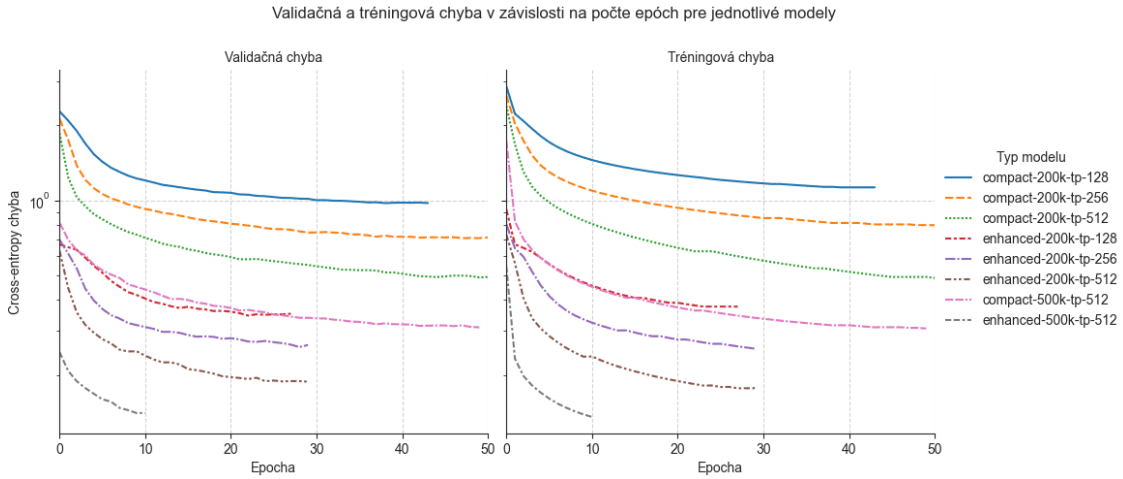


Obr. 8.9: Na obrázku je zobrazená závislosť medzi dobou trénovania a tréningovou a validačnou chybou pre jednotlivé modely v úlohe rekonštrukcie vstupu.

Dátová sada bola rozdelená na tréningovú (70%), validačnú (15%) a testovaciu sadu (15%). Bolo implementované skoré ukončenie trénovania v závislosti na validačnej chybe. Vždy keď model začal horšie generalizovať (čo naznačovalo zvýšenie validačnej chyby) sa

načítal doteraz najlepší model a zvýšila sa trpezlivosť. Ak bola dosiahnutá hranica trpezlivosti, čo v tomto prípade bolo číslo 5, tak sa tréning zastavil.

Z výsledkov učenia na úlohe rekonštrukcie vstupu (obr. 8.9) je vidieť, že čím je veľkosť vektorovej reprezentácie väčšia, tým je výsledok lepší. Ďalej sa dá povedať, že sa model lepšie učí na rozšírenej dátovej sade. Pri úlohe predikcie (obr. 8.10) je vidieť podobný trend čím je veľkosť vektorovej reprezentácie väčšia, tým je chyba nižšia. Podobne aj tu sa model lepšie učí na rozšírenej dátovej sade. Pre zaujímavosť bola pri tomto experimente vytvorená aj väčšia dátová sada s očakávaním, že sa tréningová aj validačná chyba zníži. Tento predpoklad sa potvrdil a tréningová a aj validačná chyba sa znížili, ale kvôli veľkej výpočtovej náročnosti nebolo možné trénovať pri kompaktnej dátovej sade dlhšie ako 30 epoch a pri rozšírenej 10 epoch.



Obr. 8.10: Na obrázku je zobrazená závislosť medzi dobou trénovania a tréningovou a validačnou chybou pre jednotlivé modely v úlohe predikcie nasledujúceho tokenu.

V tabuľke 8.9 sa nachádzajú výsledky metrík pre validačnú a testovaciu dátovú sadu.

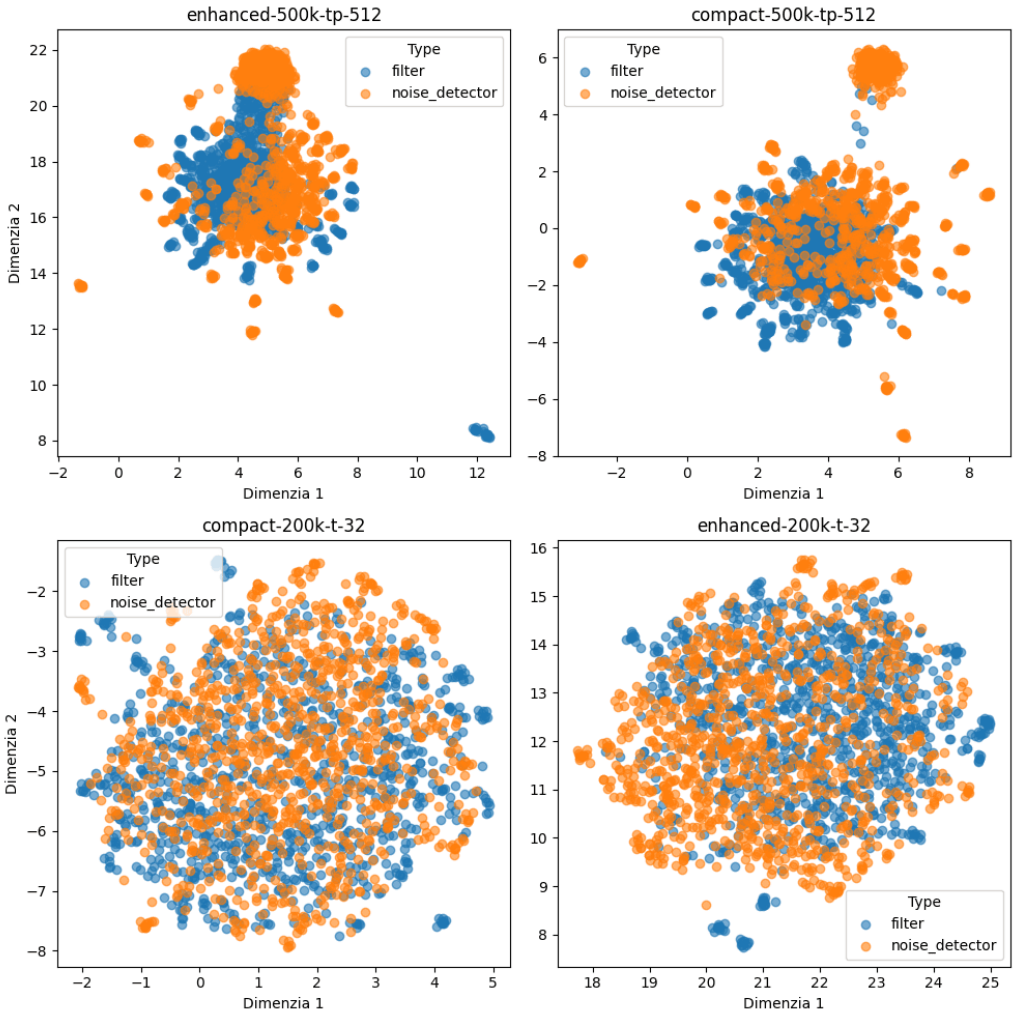
Dátová sada	Presnosť$_{\text{Mikro}}$		Presnosť$_{\text{Makro}}$		Presnosť$_{\text{Vážený}}$	
Model	Test	Valid	Test	Valid	Test	Valid
enhanced-200k-t-16	1.000	1.000	0.973	0.973	1.000	1.000
enhanced-200k-t-32	1.000	1.000	0.973	0.973	1.000	1.000
enhanced-200k-t-8	1.000	1.000	0.973	0.973	1.000	1.000
compact-200k-t-16	1.000	1.000	0.987	0.987	1.000	1.000
compact-200k-t-32	1.000	1.000	0.987	0.987	1.000	1.000
compact-200k-t-8	1.000	1.000	0.987	0.987	1.000	1.000
enhanced-200k-tp-128	0.913	0.914	0.811	0.812	0.916	0.917
enhanced-200k-tp-256	0.940	0.940	0.856	0.855	0.941	0.941
enhanced-200k-tp-512	0.958	0.958	0.889	0.888	0.958	0.958
enhanced-500k-tp-512	0.967	0.967	0.905	0.905	0.967	0.967
compact-200k-tp-128	0.749	0.749	0.746	0.745	0.754	0.754
compact-200k-tp-256	0.832	0.831	0.835	0.835	0.833	0.832
compact-200k-tp-512	0.892	0.891	0.888	0.888	0.892	0.892
compact-500k-tp-512	0.919	0.919	0.915	0.915	0.919	0.919

Tabuľka 8.9: Porovnanie modelov v závislosti na zvolenej dátovej sade a metrike.

8.5 Vizualizácia vektorových reprezentácií

Vytvorenie vizualizácie vektorových reprezentácií prebehlo na všetkých modeloch. V tomto experimente sú zobrazené len vizualizácie najlepších štyroch modelov z pohľadu chybovej funkcie a presnosti rekonštrukcie. Vizualizácia všetkých modelov sa nachádza v prílohe A na obr. A.1.

Vybrané modely pre vizualizáciu sú `compact-500k-tp-512`, `enhanced-500k-tp-512`, `compact-200k-t-32` a `enhanced-200k-t-32`. Návrh tohto experimentu je popísaný v kapitole 6.4. Tu len v stručnosti. Z validačnej dátovej sady po tokenizácii a následnom zakódovaní jednotlivých tokenov pomocou kódovacej vrstvy transformátoru sa jednotlivé vektory jedincov mohli pomocou priemernej agregácie agregovať. Agregované vektorové reprezentácie jedincov sa po normalizácii vložili na vstup metódy pre redukciu dimenzinality UMAP. Z tejto metódy vznikli body v 2D priestore, ktoré sú vizualizované v závislosti na type jedinca a podľa priemernej fitness naprieč všetkými intenzitami šumu. Vizualizácia podľa typu jedinca je zobrazená na obr. 8.11.



Obr. 8.11: Vizualizácia de novo vzniknutých vektorových reprezentácií kartézskych genetických programov. Na ose X je zobrazená prvá dimenzia z výsledku metódy UMAP a na ose Y sa nachádza druhá dimenzia.

Z vizualizácie sa dá vyčítať, že pri metóde predikcie nasledujúceho tokenu sa modely úspešne dokážu naučiť rozdeliť filtre a detektory do samostatných zhlukov. Pri druhej metóde tento jav nie je možné pozorovať. Vizualizácia podľa priemernej fitness hodnoty sa kvôli nevýznamným výsledkom nachádza v prílohe A na obr. A.2. Tu sa dá podotknúť, že keby modely boli učené na inej úlohe alebo dátovej sade, tak by bolo možné vytvoriť reprezentácie, ktoré by pri vizualizácii lepšie rozdeľovali jedincov aj v závislosti na fitness.

8.6 Mutácia úpravou vektorovej reprezentácie génov

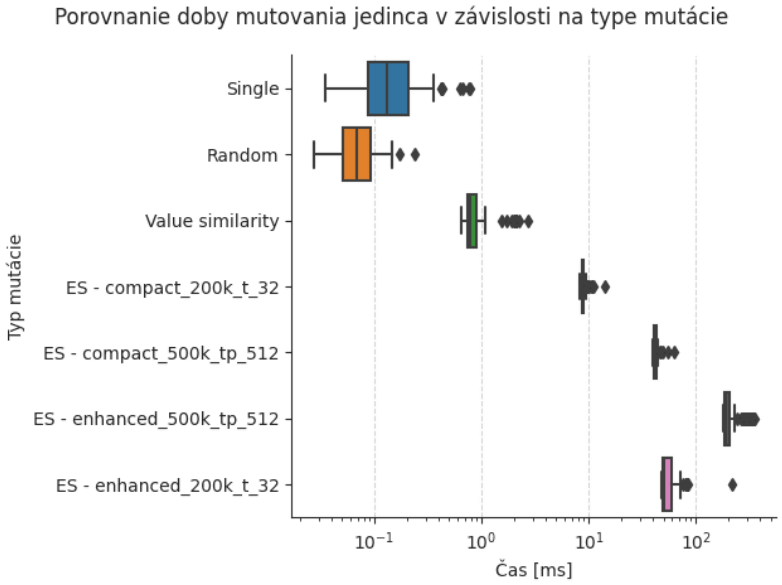
V kapitole 6.4 bol vytvorený návrh mutácie, ktorá dokáže priamo upravovať vektorovú reprezentáciu génov počas behu evolúcie. Treba podotknúť že evolúcia kartészkého genetického programovania prebieha štandardne až do momentu mutácie. Pred samotnou mutáciou sa kandidátný jedinec s jeho génmi prevedie pomocou metódy popísanej v návrhu riešenia na vektorovú reprezentáciu. Táto metóda je v tomto experimente preskúmaná pomocou viacerých modelov. Tento experiment bol prvotne zameraný na to, či dokáže model z upravených reprezentácií dekódovať validný CGP program. Na otestovanie tohto prístupu bolo náhodne vybraných 1000 filtrov a 1000 detektorov z vytvorenej dátovej sady. Filtre a detektory boli následne zakódované do vektorovej reprezentácie, kde každý gén je reprezentovaný ako jeden vektor. Následne ku každej dimenzii každého vektoru bola pripočítaná 0, aby sa zistilo, ako sa modelu darí správne dekódovať CGP programy a potom následne boli pripočítavané hodnoty z intervalov $\langle -0.2, 0.2\rangle$, $\langle -0.4, 0.4\rangle$, $\langle -0.6, 0.6\rangle$, $\langle -0.8, 0.8\rangle$, $\langle -1, 1\rangle$. Zaznamenával sa počet dekódovaných originálnych, nových – validných a nových – invalidných CGP programov. Výsledky tohto experimentu sa nachádzajú v prílohe B na obr. B.1.

Z výsledkov sa dá pozorovať, že tento prístup nedokáže dekódovať validné CGP programy. Už z metrík pri validačnej a testovacej dátovej sade sa dalo pozorovať, že výsledky pre modely ktoré predikujú nasledujúci token nebudú mať uspokojivé výsledky pri rekonštrukcii vstupu, keďže nedokážu dekódovať ani jeden vstup správne aj bez úpravy vektorovej reprezentácie. Pri úlohe rekonštrukcie je výsledok viac uspokojivý. Modely dokážu rekonštruovať originálny CGP program, ale majú problémy zrekonštruovať upravené vektorové reprezentácie. Výsledok je väčšinou invalidný. Validné výsledky sa vyskytli pri jednom intervale, ale pre ďalšie pokračovanie tohto experimetu a otestovanie tejto mutácie pri návrhu obrazových filtrov to nestačí. Preto sa táto mutácia už ďalej v tejto práci neskúma.

8.7 Porovnanie vybraných mutácií s mutáciou založenou na podobnosti vektorov

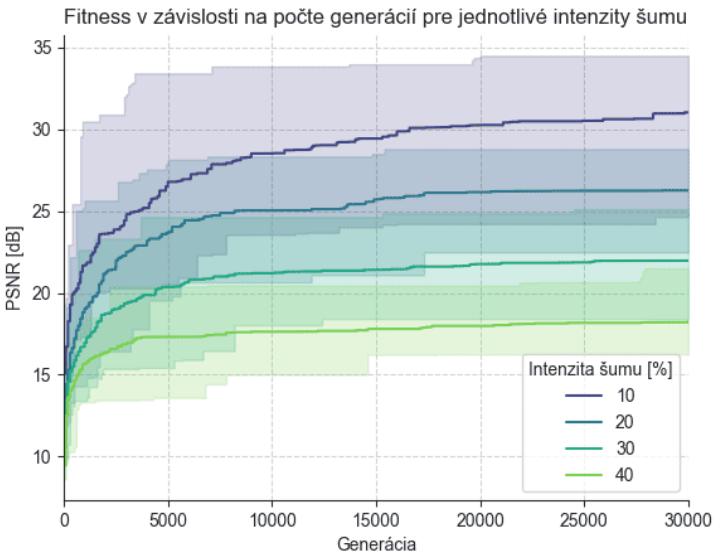
Tento experiment sa zaoberá druhou navrhnutou mutáciou v kapitole 6.4. Podobne ako v predchádzajúcej mutácii treba podotknúť že evolúcia v kartézskom genetickom programovaní prebieha štandardne, až do momentu vykonávania mutácie. Pred samotnou mutáciou sa kandidátný jedinec s jeho génmi prevedie pomocou metódy popísanej v návrhu riešenia na vektorovú reprezentáciu. Mutácia je založená na lokalizácii a extrahovaní vektorov, ktoré reprezentujú gény v CGP programe. Tieto vektory sú normalizované a porovnané medzi sebou. Indexy vektorov sú vybrané, ak prekročia prah podobnosti, ktorý je nastavený na 0.8. Vybrané indexy sú podľa podobnosti zaradené do skupín navzájom podobných génov. Z každej skupiny je následne vybraný jeden náhodný index génu pre mutáciu. Na konci každej mutácie je s pravdepodobnosťou 0.05 mutovaný aj výstupný gén. Ak by sa

stalo, že by neboli žiadne gény pre mutáciu, tak je vybraných 5 náhodných génov. Táto metóda je následne porovnaná s metódou single, náhodnou mutáciou a metódou založenou na podobnosti hodnôt génov, ktorá bola tiež popísaná v kapitole 6.4.



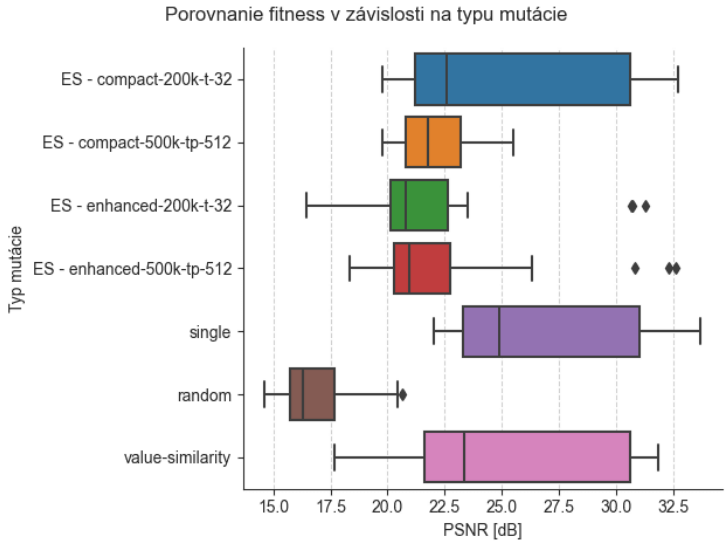
Obr. 8.12: Porovnanie doby trvania mutácie v závislosti na použitej metóde.

Mutácie boli porovnané z pohľadu doby mutovania jedinca. Každá mutácia na rovnako vygenerovanom počiatočnom jedincovi vykonala 100 mutácií. Časy mutovania boli zaznamenané a zobrazené na obr. 8.12. Z grafu sa dá pozorovať, že náhodná mutácia a metóda single sú v súvislosti s časom mutovania najrýchlejšie. Metóda založená na podobnosti vektorov bola otestovaná na štyroch modeloch a trvala zo všetkých najdlhšie. Modely, ktoré používajú rozšírenú dátovú sadu a väčšiu veľkosť vektorovej reprezentácie sú pomalšie. Metóda založená na podobnosti hodnôt génov je pomalšia ako náhodná mutácia a metóda single, ale je rýchlejšia ako mutácia založená na podobnosti vektorov.



Obr. 8.13: Na tomto obrázku je zobrazený priebeh fitness v závislosti na počte generácií pre jednotlivé intenzity šumu v štandardnom evolučnom procese bez použitia vzniknutých modelov.

Mutácie sú ďalej porovnané pri návrhu obrazových filtrov. Nastaviteľné parametre boli použité rovnaké ako pri vytváraní dátovej sady (tabuľka 6.1). Rozdiel je len v počte generácii, vybranej metóde mutovania jedincov a navrhuje sa len na 10 % šume vzhľadom na výsledky dosiahnuté pri overovaní funkčnosti návrhu obrazových filtrov. Počet generácii bol kvôli výpočtovej náročnosti znížený na 5000.



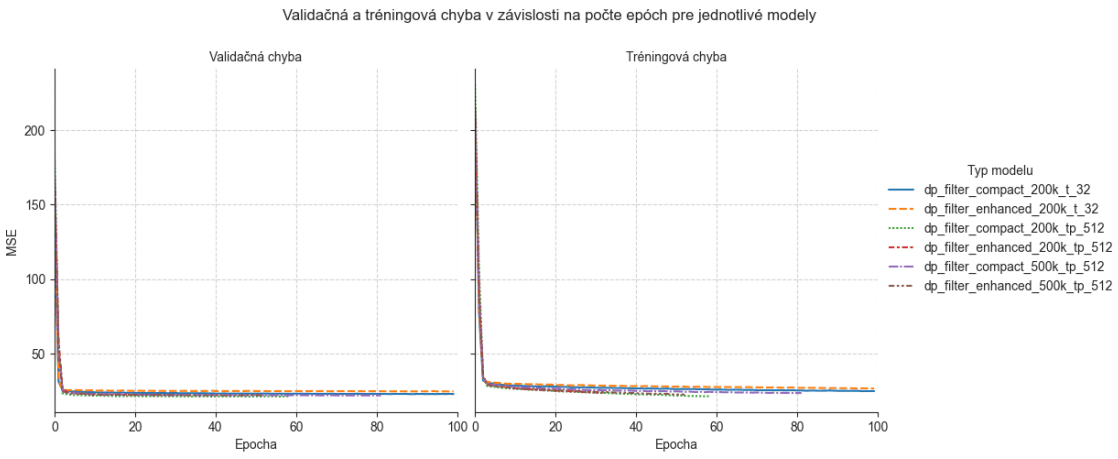
Obr. 8.14: Porovnanie fitness z každého behu evolúcie v závislosti na použitej mutácii pri návrhu obrazových filtrov spolu s detektormi šumu.

Počet generácii bol vybraný na základe pozorovania vývoja fitness v čase (obr. 8.13), kde to vyzerá tak, že uspokojivé výsledky fitness dokážu byť dosiahnuté už aj pri 5000 ge-

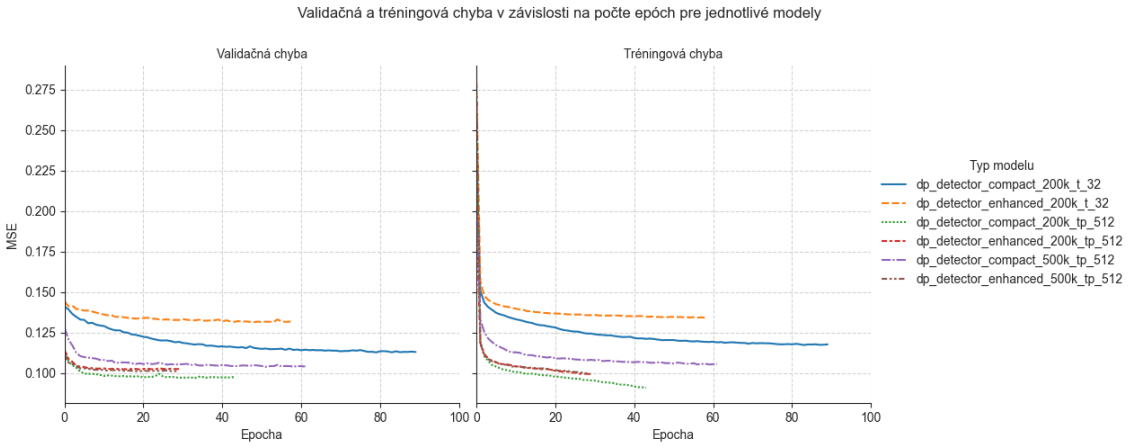
neráciách. Z výsledkov tejto časti experimentu (obr. 8.14) je jasne vidieť, že všetky metódy sú lepšie ako náhodná mutácia. Najzaujímavejšie výsledky je možné pozorovať pri typoch mutácie single, value-similarity a ES-compact-200K-t-32. Na týchto troch distribúciach boli vykonané testy na normalitu a následne Mann Whitney U-test. Distribúcia metódy single nie je štatisticky významne podobná ostatným dvom distribúciam. Na druhú stranu, distribúcie podobnosti vektorov a podobnosti hodnôt si sú podobné a metóda založená na podobnosti hodnôt génov je lepšia. Z týchto výsledkom je zrejmé, že metóda založená na podobnosti vektorov je horšia jednak časovo, tak aj výslednými kompozíciami ako metóda založená na podobnosti hodnôt génov. Aj keď táto metóda dosahuje podobných výsledkov ako štandardný prístup tak sa ju kvôli časovej náročnosti aktuálne neoplatí používať dokým sa nenájde menší model, ktorý by bol menej časovo náročný.

8.8 Trénovanie prediktora fitness kandidátnych riešení

Ako prediktor fitness bol zvolený viacvrstvový perceptrón. Bolo vytvorených 12 prediktorov fitness, 6 pre filtre a 6 pre detektory. Pre trénovanie, validáciu a testovanie bola použitá dátová sada o veľkosti 170 tisíc vzoriek, kde sa nachádza 85 tisíc filtrov a 85 tisíc detektorov. Dátové sady filtrov a detektorov boli rozdelené do tréningovej (70%), validačnej (20%) a testovacej (10%) dátovej sady. Bolo použitých 6 najlepších modelov pre prevod jedinca do vektorovej reprezentácie (`compact-500k-tp-512`, `enhanced-500k-tp-512`, `compact-200k-tp-512`, `enhanced-200k-tp-512`, `compact-200k-t-32` a `enhanced-200k-t-32`. Architektúra prediktoru sa skladá zo vstupnej vrstvy, troch skrytých vrstiev a jednej výstupnej vrstvy. Vstupná vrstva obsahuje 32 alebo 512 neurónov na základe použitého modelu pre kódovanie. Skryté vrstvy obsahujú 256, 128 a 64 neurónov. Výstupná vrstva obsahuje jeden neurón. Po každej skrytej vrstve nasledovala 1D batch normalizácia, aktivačná funkcia ReLU a dropout s pravdepodobnosťou 0.4. Pri učení bola použitá MSE funkcia ako chybová funkcia. Výsledky z trénovania prediktoru fitness pre filtre sa nachádzajú na obr. 8.15 a pre detektory na obr. 8.16.



Obr. 8.15: Na obrázku je zobrazený proces učenia prediktoru fitness pre filter.



Obr. 8.16: Na obrázku je zobrazený proces učenia prediktoru fitness pre detektor.

Pre lepšie porovnanie výsledkov bol na validačnej a testovacej dátovej sade po predikcii vypočítaný Spearmanov korelačný koeficienta, MSE a MAE medzi očakávanými a predikovanými distribúciami. Výsledky sa nachádzajú v tabuľke 8.10 a v tabuľke 8.11.

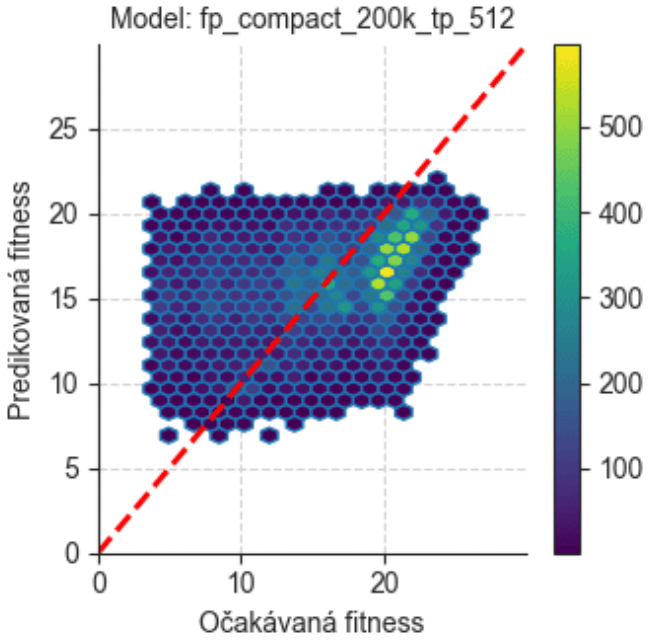
Dátová sada Model	MSE Test	 Valid	MAE Test	 Valid	Spearman Test	 Valid
fp_compact_200k_t_32	22.996	22.886	3.973	3.964	0.353	0.350
fp_compact_200k_tp_512	20.884	21.208	3.678	3.704	0.471	0.460
fp_compact_500k_tp_512	21.834	21.941	3.811	3.822	0.419	0.409
fp_enhanced_200k_t_32	24.650	24.649	4.187	4.182	0.224	0.209
fp_enhanced_200k_tp_512	22.254	22.285	3.880	3.883	0.402	0.397
fp_enhanced_500k_tp_512	21.910	22.011	3.801	3.805	0.422	0.417

Tabuľka 8.10: Porovnanie prediktorov pre filtre v závislosti na zvolenej dátovej sade a metrike.

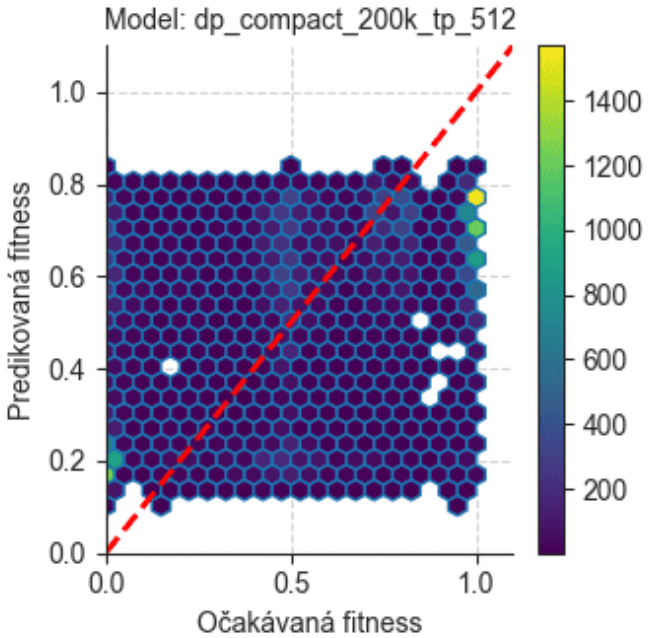
Dátová sada Model	MSE Test	 Valid	MAE Test	 Valid	Spearman Test	 Valid
dp_compact_200k_t_32	0.115	0.113	0.288	0.285	0.415	0.418
dp_compact_200k_tp_512	0.098	0.097	0.260	0.258	0.520	0.518
dp_compact_500k_tp_512	0.105	0.104	0.273	0.270	0.480	0.478
dp_enhanced_200k_t_32	0.133	0.132	0.313	0.309	0.208	0.208
dp_enhanced_200k_tp_512	0.103	0.102	0.271	0.269	0.487	0.477
dp_enhanced_500k_tp_512	0.103	0.101	0.269	0.266	0.489	0.485

Tabuľka 8.11: Porovnanie prediktorov pre detektory v závislosti na zvolenej dátovej sade a metrike.

Kompletná vizualizácia očakávaných a predikovaných hodnôt sa nachádzajú v prílohe C na obr. C.1 a C.2. V tejto sekcii je zobrazený len najlepší model prediktoru fitness pre filter na obr. 8.17. a detektor na obr. 8.18.



Obr. 8.17: Obrázok zobrazuje predikcie najlepšieho prediktoru pre filter



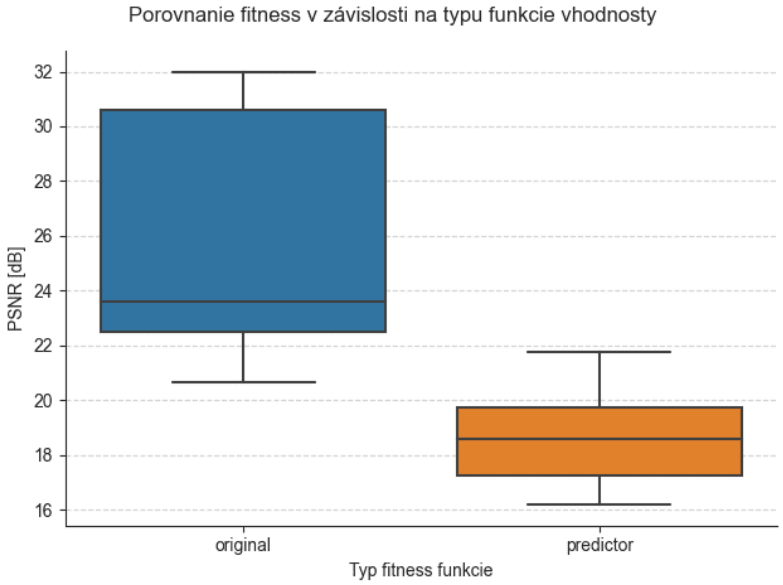
Obr. 8.18: Obrázok zobrazuje predikcie najlepšieho prediktoru pre detektor

Z výsledkov sa dá pozorovať, že pri prediktore fitness hodnoty filtrov dochádza k menšej korelácii. Tento fenomén je pozorovateľný jak na obrázku, tak aj v tabuľke výsledkov. Pri prediktore fitness hodnoty detektorov to už také zrejmé nie je. Na obrázku vyzerá, že žiadna korelácia tam nie je, ale v tabuľke výsledkov je vidieť miernu pozitívnu koreláciu.

8.9 Porovnanie navrhovania kompozícii s a bez prediktora fitness

Tento experiment sa zaoberá návrhom kompozícií s pomocou prediktora fitness. Ako porovnanie bol použitý štandardný prístup, ktorý bol použitý na vytvorenie dátovej sady v rámci tejto práce. Aj napriek nie príliš dobrým výsledkom predikcie fitness boli vybrané dva prediktory na ďalšie preskúmanie. Jeden pre predikciu fitness hodnoty filtrov a druhý pre predikciu fitness hodnoty detektorov.

Program na návrh obrazových filtrov bol spustený 30 krát po 5000 generácií na poškodených obrázkoch s 10 % intenzitou šumu. Ostatné parametre boli nastavené rovnako aj pri pôvodnom postupe. Hlavný rozdiel spočíva vo výpočte fitness hodnoty. Namiesto evaluácie jedincov nad tréningovými dátami sa jedinec zakóduje do vektorovej reprezentácie, ktorá sa znormalizuje, agreguje a predá do prediktora, ktorý vráti predikovanú fitness. Evaluácia či už filtra alebo detektoru prebieha vždy, keď je treba vykonať kompozíciu.



Obr. 8.19: Porovnanie medzi pôvodnou metódou a metódou za použitia prediktorou fitness.

Výsledky fitness v závislosti na použitej metóde je možné pozorovať na nasledujúcom obr. 8.19, kde je jasne vidieť, že štandardná metóda dokáže produkovať lepšie výsledky. K neuspokojivým výsledkom z veľkej časti prispieva nepresnosť prediktorov. Z časového hľadiska sa tieto dve metódy neporovnávali a to z dôvodu väčšej výpočtovej náročnosti natrénovaných transformátorov, ktorá bola zdôraznená na obr. 8.12.

Kapitola 9

Diskusia

Táto kapitola je zameraná na diskusiu k výsledkom experimentu a budúce pokračovanie práce.

Pri overovaní funkčnosti návrhu obrazových filtrov bola overená funkčnosť filtrov na štyroch intenzitách šumu. Výsledky boli porovnateľné s výsledkami v práci [16]. Navyše boli filtre zrovnané v ich fitness aj na šumoch, na ktorých neboli navrhované. Z týchto výsledkov je jasné, že viac sa oplatí navrhovať filtre na nižších šumoch vzhľadom na to, že dokážu po viac iteráciách dosiahnuť lepšie výsledky.

Z výsledkov vizualizácie vektorovej reprezentácie je vidieť, že predikčná úloha dokáže navrhovať lepšie výsledné reprezentácie, ale za cenu väčšej výpočtovej náročnosti. V tejto oblasti by bolo vhodné preskúmať aj iné architektúry, modely, úlohy prípadne aj formát dát. V tejto práci bola využitá len architektúra autoenkóderu, ale zaujímavé by mohlo byť preskúmať aj architektúru variačného autoenkóderu. Ako model bol využitý transformátor, ale vzhľadom na formát dát, kde CGP programy sú orientované acyklický grafy, by bolo vhodné otestovať aj grafové neurónové siete. Vzhľadom na veľkú špecifickosť dát neboli využité ani predtrénované transformátory, čo by možno tiež stálo za preskúmanie, ale v tejto práci neboli obsiahnuté. Pre získavanie vhodných reprezentácii môže byť využitá aj predikčná úloha maskovaných niektorých častí vstupu.

Preskúmané dva nové mutačné operátory sú aktuálne veľmi obmedzené. Prvý aktuálne nedokáže generovať validné CGP programy a druhý dokázal vytvárať porovnateľné výsledky, ale je výpočtovo náročný. Pre budúce pokračovanie v tejto oblasti by bolo možné vyskúšať predikčnú úlohu maskovaných častí vstupu, kde by sa následne maskovali náhodné časti CGP programov a model by sa snažil doplniť chýbajúce gény. Ďalej by bolo zaujímavé preskúmať aj iné varianty CGP, a to napríklad Rekurzívne CGP, kde sú medzi uzlami povolené cykly. Prípadne pri trénovaní modelov obmedziť predikciu podľa pozície na aktuálne validné tokeny z gramatiky a nie na celú gramatiku.

Prediktor fitness bol v tejto práci preskúmaný len veľmi obmedzene. Výsledky prediktorov nie sú vôbec významné, a pre budúce pokračovanie v tejto oblasti je nutné nájsť modely pre vytváranie reprezentácii, ktoré nebudú tak výpočtovo náročné. Ale stále je možné preskúmať aj ďalšie prediktory ako sú náhodné lesy, metódy podporných vektorov a ďalšie.

Kapitola 10

Záver

V tejto práci boli overené doterajšie výsledky pri návrhu obrazových filtrov s detektorom šumu pomocou kompozičnej koevolúcie v kartézskom genetickom programovaní. Výsledky ukázali, že kompozície ktoré sú navrhnuté na nižšej intenzite šumu si v závislosti na počte iterácií vedú lepšie ako ostatné. Pomocou návrhu obrazových filtrov bola vygenerovaná rozsiahla dátová sada filtrov a detektorov, ktoré boli predspracované do nových vhodnejších textových reprezentácii.

Novo vzniknuté textové reprezentácie boli použité pri preskúmaní novej metódy pre automatické vytváranie vektorových reprezentácií pre kartézske genetické programovanie za pomoci transformátoru. Reprezentácie boli vytvárané na dvoch úlohách: rekonštrukcii pôvodného vstupu a predikcii nasledujúceho tokenu. De novo vzniknuté reprezentácie boli vizualizované, použité v novo vzniknutých mutačných operátoroch a na predikciu fitness.

Reprezentácie vytvorené pomocou úlohy predikcie nasledujúceho tokenu dokázali od seba oddeliť filtre a detektory vo vektorovom priestore. Ako bolo spomenuté, boli preskúmané dva mutačné operátory: mutačný operátor založený na priamej zmene vektorovej reprezentácie a mutačný operátor založený na výbere génov na základe podobnosti génov vo vektorovej reprezentácii. Prvý mutačný operátor nedokázal generovať validné kartézske genetické programy, z toho dôvodu bol zavrhnutý. Druhý dokázal dosiahnuť dobré výsledky porovnateľné so štandardným prístupom, ale pre vyššiu výpočtovú náročnosť sa aktuálne neoplatí používať.

Ako posledné bolo preskúmané vytvorenie prediktoru fitness s využitím vzniknutých reprezentácií. Vytvorené prediktory na základe výsledkov mali menšiu koreláciu medzi očakávanými a predikovanými fitness hodnotami jedincov, ale pri overení použitia prediktoru oproti pôvodnej metóde neboli dosiahnuté lepšie výsledky.

Možností pokračovania tejto práce je veľa. Mohlo by byť prínosné preskúmať ďalšie úlohy pomocou ktorých by možno vznikli zaujímavejšie reprezentácie. Jednou z takýchto úloh by mohla byť predikcia maskovaných častí vstupu. Taktiež by mohlo byť zaujímavé preskúmať aj iné modely než transformátor. Vzhľadom na štruktúru kartézskych genetických programov, by mohlo byť zaujímavé preskúmať grafové neuronové siete.

Cenným prínosom tejto práce je vygenerovaná rozsiahla dátová sada filtrov a detektorov, predspracované dátové sady do novo vzniknutých textových reprezentácii a samozrejme natrénované transformátory a prediktory, ktoré môžu byť použité pre ďalší výskum vo výskumnej skupine Evolvable Hardware.

Literatúra

[1] Aggarwal, C. C. Convolutional Neural Networks. In: *Neural Networks and Deep Learning: A Textbook*. Cham: Springer International Publishing, 2018, s. 315–371. DOI: 10.1007/978-3-319-94463-0_8. ISBN 978-3-319-94463-0. Dostupné z: https://doi.org/10.1007/978-3-319-94463-0_8.

[2] Alpaydin, E. *Introduction to Machine Learning*. 2nd. The MIT Press, 2010. ISBN 026201243X.

[3] Baker, J. E. Reducing Bias and Inefficienry in the Selection Algorithm. In: *International Conference on Genetic Algorithms*. 1987. Dostupné z: https://api.semanticscholar.org/CorpusID:33236432.

[4] Biewald, L. *Experiment Tracking with Weights and Biases*. 2020. Software available from wandb.com. Dostupné z: https://www.wandb.com/.

[5] Bishop, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN 0387310738.

[6] Box, G. E. P. Evolutionary Operation: a Method for Increasing Industrial Productivity. *Journal of The Royal Statistical Society Series C-applied Statistics*. 1957, zv. 6, s. 81–101. Dostupné z: https://api.semanticscholar.org/CorpusID:67695486.

[7] Bradski, G. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*. 2000.

[8] Buduma, N. a Locascio, N. *Fundamentals of Deep Learning: Designing Next-Generation Machine Intelligence Algorithms*. 1st. O'Reilly Media, Inc., 2017. ISBN 1491925612.

[9] Burkov, A. B. *The Hundred-Page Machine Learning Book*. Andriy Burkov, 2019. ISBN 978-1999579500.

[10] Caetano, V., Teixeira, M. C. a Pappa, G. L. Symbolic Regression Trees as Embedded Representations. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. New York, NY, USA: Association for Computing Machinery, 2023, s. 411–419. GECCO '23. DOI: 10.1145/3583131.3590423. ISBN 9798400701191. Dostupné z: https://doi.org/10.1145/3583131.3590423.

[11] Chacon, S. a Straub, B. *Pro Git*. 2. vyd. Apress, 2014. Dostupné z: https://git-scm.com/book/en/v2.

[12] Cho, K., Merrienboer, B., Gulcehre, C., Bougares, F., Schwenk, H. et al. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. Jún 2014. DOI: 10.3115/v1/D14-1179.

[13] Cramer, N. L. A representation for the Adaptive Generation of Simple Sequential Programs. In: Grefenstette, J. J., ed. *Proceedings of an International Conference on Genetic Algorithms and the Applications*. Carnegie-Mellon University, Pittsburgh, PA, USA: [b.n.], 24-26 júl 1985, s. 183–187. Dostupné z: https://dl.acm.org/doi/10.5555/645511.657085.

[14] De Jong, K. Generalized Evolutionary Algorithms. In: Rozenberg, G., Bä, T. a Kok, J. N., ed. *Handbook of Natural Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, s. 625–635. DOI: 10.1007/978-3-540-92910-9_20. Dostupné z: https://doi.org/10.1007/978-3-540-92910-9_20.

[15] Devlin, J., Chang, M.-W., Lee, K. a Toutanova, K. {BERT}: Pre-training of Deep Bidirectional Transformers for Language Understanding. In: Burstein, J., Doran, C. a Solorio, T., ed. *Proceedings of the 2019 Conference of the North {A}merican Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, Jún 2019, s. 4171–4186. DOI: 10.18653/v1/N19-1423. Dostupné z: https://aclanthology.org/N19-1423.

[16] Drahošová, M., Komjáthy, G. a Sekanina, L. Towards Compositional Coevolution in Evolutionary Circuit Design. In: *2014 IEEE International Conference on Evolvable Systems Proceedings*. Institute of Electrical and Electronics Engineers, 2014, s. 157–164. DOI: 10.1109/ICES.2014.7008735. ISBN 978-1-4799-4479-8. Dostupné z: https://www.fit.vut.cz/research/publication/10655.

[17] Eiben, A. E. a Smith, J. E. Fitness, Selection, and Population Management. In: *Introduction to Evolutionary Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, s. 79–98. DOI: 10.1007/978-3-662-44874-8_5. Dostupné z: https://doi.org/10.1007/978-3-662-44874-8_5.

[18] Eiben, A. E. a Smith, J. E. Representation, Mutation, and Recombination. In: *Introduction to Evolutionary Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, s. 49–78. DOI: 10.1007/978-3-662-44874-8_4. Dostupné z: https://doi.org/10.1007/978-3-662-44874-8_4.

[19] Forsyth, R. {BEAGLE} A {Darwinian} Approach to Pattern Recognition. *Kybernetes*. 1981, zv. 10, č. 3, s. 159–166. DOI: doi:10.1108/eb005587. ISSN 0368-492X.

[20] Foundation, P. S. *Python Programming Language*. 2020. Python Software Foundation. Python Language Reference, version 3.10. Dostupné z: https://www.python.org/.

[21] Friedberg, R. M. A Learning Machine: Part I. *IBM Journal of Research and Development*. 1958, zv. 2, č. 1, s. 2–13. DOI: 10.1147/rd.21.0002.

[22] Goldman, B. W. a Punch, W. F. Reducing Wasted Evaluations in Cartesian Genetic Programming. In: Krawiec, K., Moraglio, A., Hu, T., Etaner Uyar,

A. Ş. a Hu, B., ed. *Genetic Programming*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, s. 61–72. ISBN 978-3-642-37207-0.

[23] Goldman, B. W. a Punch, W. F. Analysis of Cartesian Genetic Programming's Evolutionary Mechanisms. *IEEE Transactions on Evolutionary Computation*. 2015, zv. 19, č. 3, s. 359–373. DOI: 10.1109/TEVC.2014.2324539.

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

[25] Hadsell, R., Chopra, S. a LeCun, Y. Dimensionality Reduction by Learning an Invariant Mapping. In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*. 2006, sv. 2, s. 1735–1742. DOI: 10.1109/CVPR.2006.100.

[26] Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P. et al. Array programming with NumPy. *Nature*. 2020, zv. 585, s. 357–362. Dostupné z: https://www.nature.com/articles/s41586-020-2649-2.

[27] Hinton, G. E. a Salakhutdinov, R. R. Reducing the Dimensionality of Data with Neural Networks. *Science*. 2006, zv. 313, č. 5786, s. 504–507. DOI: 10.1126/science.1127647. Dostupné z: https://www.science.org/doi/abs/10.1126/science.1127647.

[28] Hochreiter, S. a Schmidhuber, J. Long Short-Term Memory. *Neural Comput.* Cambridge, MA, USA: MIT Press. nov 1997, zv. 9, č. 8, s. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735. ISSN 0899-7667. Dostupné z: https://doi.org/10.1162/neco.1997.9.8.1735.

[29] Holland, J. H. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1992. ISBN 0262082136.

[30] Hunter, J. D. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*. IEEE Computer Society. 2007, zv. 9, č. 3, s. 90–95.

[31] Jing, L. a Tian, Y. Self-Supervised Visual Feature Learning With Deep Neural Networks: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2019, zv. 43, s. 4037–4058. Dostupné z: https://api.semanticscholar.org/CorpusID:62841734.

[32] Kingma, D. P. a Welling, M. Auto-Encoding Variational Bayes. *CoRR*. 2013, abs/1312.6114. Dostupné z: https://api.semanticscholar.org/CorpusID:216078090.

[33] Kipf, T. N. a Welling, M. Semi-Supervised Classification with Graph Convolutional Networks. *ArXiv preprint arXiv:1609.02907*. 2016.

[34] Kočí, M. *Evoluční návrh klasifikátoru obrazů*. Brno, CZ, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.fit.vut.cz/study/thesis/23727/.

[35] Koza, J. R. *Genetic programming: on the programming of computers by means of natural selection*. Cambridge, MA, USA: MIT Press, 1992. ISBN 0262111705.

[36] Koza, J. R. a Poli, R. Genetic Programming. In: Burke, E. K. a Kendall, G., ed. *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*. Springer, 2005, kap. 5, s. 127–164. DOI: doi:10.1007/978-1-4614-6940-7_6. ISBN 0-387-23460-8. Dostupné z: http://cswww.essex.ac.uk/staff/poli/papers/KozaPoli2005.pdf.

[37] Kumar, A. Different Types of CNN Architectures Explained: Examples. *Vitalflux*. 2023. Dostupné z: https://vitalflux.com/different-types-of-cnn-architectures-explained-examples/.

[38] MACH, M. *Evolučné algoritmy: prvky a princípy*. Elfa, 2009. 250 s. ISBN 978-80-8086-123-0.

[39] Martin, D., Fowlkes, C., Tal, D. a Malik, J. A Database of Human Segmented Natural Images and its Application to Evaluating Segmentation Algorithms and Measuring Ecological Statistics. In: *Proc. 8th Int'l Conf. Computer Vision*. July 2001, sv. 2, s. 416–423.

[40] McCulloch, W. S. a Pitts, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*. Dec 1943, zv. 5, č. 4, s. 115–133. DOI: 10.1007/BF02478259. ISSN 1522-9602. Dostupné z: https://doi.org/10.1007/BF02478259.

[41] McInnes, L., Healy, J., Saul, N. a Grossberger, L. UMAP: Uniform Manifold Approximation and Projection. *Journal of Open Source Software*. The Open Journal. 2018, zv. 3, č. 29, s. 861. DOI: 10.21105/joss.00861. Dostupné z: https://doi.org/10.21105/joss.00861.

[42] McKinney, W. Data Structures for Statistical Computing in Python. In: *Proceedings of the 9th Python in Science Conference*. 2010, s. 51–56.

[43] Miller, J., Thomson, P., Fogarty, T. a Ntroduction, I. Designing Electronic Circuits Using Evolutionary Algorithms. Arithmetic Circuits: A Case Study. *Genetic Algorithms and Evolution Strategies in Engineering and Computer Science*. Október 1998.

[44] Miller, J. F. An empirical study of the efficiency of learning boolean functions using a Cartesian Genetic Programming approach. In:. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, s. 1135–1142. GECCO'99. ISBN 1558606114.

[45] Miller, J. F. Cartesian Genetic Programming. In: Miller, J. F., ed. *Cartesian Genetic Programming*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, s. 17–34. DOI: 10.1007/978-3-642-17310-3_2. Dostupné z: https://doi.org/10.1007/978-3-642-17310-3_2.

[46] Miller, J. F. Introduction to Evolutionary Computation and Genetic Programming. In: Miller, J. F., ed. *Cartesian Genetic Programming*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, s. 1–16. DOI: 10.1007/978-3-642-17310-3_1. ISBN 978-3-642-17310-3. Dostupné z: https://doi.org/10.1007/978-3-642-17310-3_1.

[47] Miller, J. F. a Thomson, P. Cartesian Genetic Programming. In: Poli, R., Banzhaf, W., Langdon, W. B., Miller, J., Nordin, P. et al., ed. *Genetic*

Programming. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, s. 121–132. ISBN 978-3-540-46239-2.

[48] Miller, J. F. Cartesian genetic programming: its status and future. *Genetic Programming and Evolvable Machines*. Jun 2020, zv. 21, č. 1, s. 129–168. DOI: 10.1007/s10710-019-09360-6. ISSN 1573-7632. Dostupné z: https://doi.org/10.1007/s10710-019-09360-6.

[49] Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E. et al. Automatic differentiation in PyTorch. In: *Advances in Neural Information Processing Systems*. 2017.

[50] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B. et al. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*. 2011, zv. 12, s. 2825–2830.

[51] Poli, R. a Langdon, W. Genetic Programming with One-Point Crossover and Point Mutation. Apríl 1997. DOI: 10.1007/978-1-4471-0427-8_20.

[52] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D. et al. Language Models are Unsupervised Multitask Learners. In:. 2019. Dostupné z: https://api.semanticscholar.org/CorpusID:160025533.

[53] Restak, R. *The Secret Life of the Brain*. Joseph Henry Press, 2001. ISBN 9780309074353. Dostupné z: https://books.google.cz/books?id=jjlx8b2p3sMC.

[54] Rosenblatt, F. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan Books, 1962. Cornell Aeronautical Laboratory. Report no. VG-1196-G-8. Dostupné z: https://books.google.cz/books?id=7FhRAAAAMAAJ.

[55] Rumelhart, D. E., Hinton, G. E. a Williams, R. J. Learning representations by back-propagating errors. *Nature*. Oct 1986, zv. 323, č. 6088, s. 533–536. DOI: 10.1038/323533a0. ISSN 1476-4687. Dostupné z: https://doi.org/10.1038/323533a0.

[56] Sanchez Lengeling, B., Reif, E., Pearce, A. a Wiltschko, A. B. A Gentle Introduction to Graph Neural Networks. *Distill*. 2021. DOI: 10.23915/distill.00033. https://distill.pub/2021/gnn-intro.

[57] Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M. a Monfardini, G. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*. 2009, zv. 20, č. 1, s. 61–80. DOI: 10.1109/TNN.2008.2005605.

[58] Sikulova, M. a Sekanina, L. Acceleration of Evolutionary Image Filter Design Using Coevolution in Cartesian GP. In: Coello, C. A. C., Cutello, V., Deb, K., Forrest, S., Nicosia, G. et al., ed. *Parallel Problem Solving from Nature - PPSN XII*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, s. 163–172. ISBN 978-3-642-32937-1.

[59] Šikulová, M. a Sekanina, L. Coevolution in Cartesian Genetic Programming. In: Moraglio, A., Silva, S., Krawiec, K., Machado, P. a Cotta, C., ed. *Genetic Programming*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, s. 182–193. ISBN 978-3-642-29139-5.

[60] Smith, S. F. *A learning system based on genetic adaptive algorithms.* USA, 1980. Dizertačná práca.

[61] Sutton, R. S. a Barto, A. G. *Reinforcement Learning: An Introduction.* Cambridge, MA, USA: A Bradford Book, 2018. ISBN 0262039249.

[62] Turing, A. M. Computing Machinery and Intelligence. *Mind.* [Oxford University Press, Mind Association]. 1950, zv. 59, č. 236, s. 433–460. ISSN 00264423, 14602113. Dostupné z: http://www.jstor.org/stable/2251299.

[63] Tutorials, P. *Language Modeling with nn.Transformer and torchtext.* 2023. Dostupné z: https://pytorch.org/tutorials/beginner/transformer_tutorial.html.

[64] Vanneschi, L. a Poli, R. Genetic Programming — Introduction, Applications, Theory and Open Issues. In: Rozenberg, G., Bäck, T. a Kok, J. N., ed. *Handbook of Natural Computing.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, s. 709–739. DOI: 10.1007/978-3-540-92910-9_24. ISBN 978-3-540-92910-9. Dostupné z: https://doi.org/10.1007/978-3-540-92910-9_24.

[65] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L. et al. Attention is all you need. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems.* Red Hook, NY, USA: Curran Associates Inc., 2017, s. 6000–6010. NIPS'17. ISBN 9781510860964.

[66] Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T. et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods.* 2020, zv. 17, s. 261–272. Dostupné z: https://www.nature.com/articles/s41592-019-0686-2.

[67] Walt, S. van der, Schönberger, J. L., Nunez Iglesias, J., Boulogne, F., Warner, J. D. et al. Scikit-image: Image processing in Python. *PeerJ.* 2014, zv. 2, s. e453. DOI: 10.7717/peerj.453.

[68] Waskom, M. *Seaborn: statistical data visualization.* 2021. Dostupné z: https://seaborn.pydata.org/.

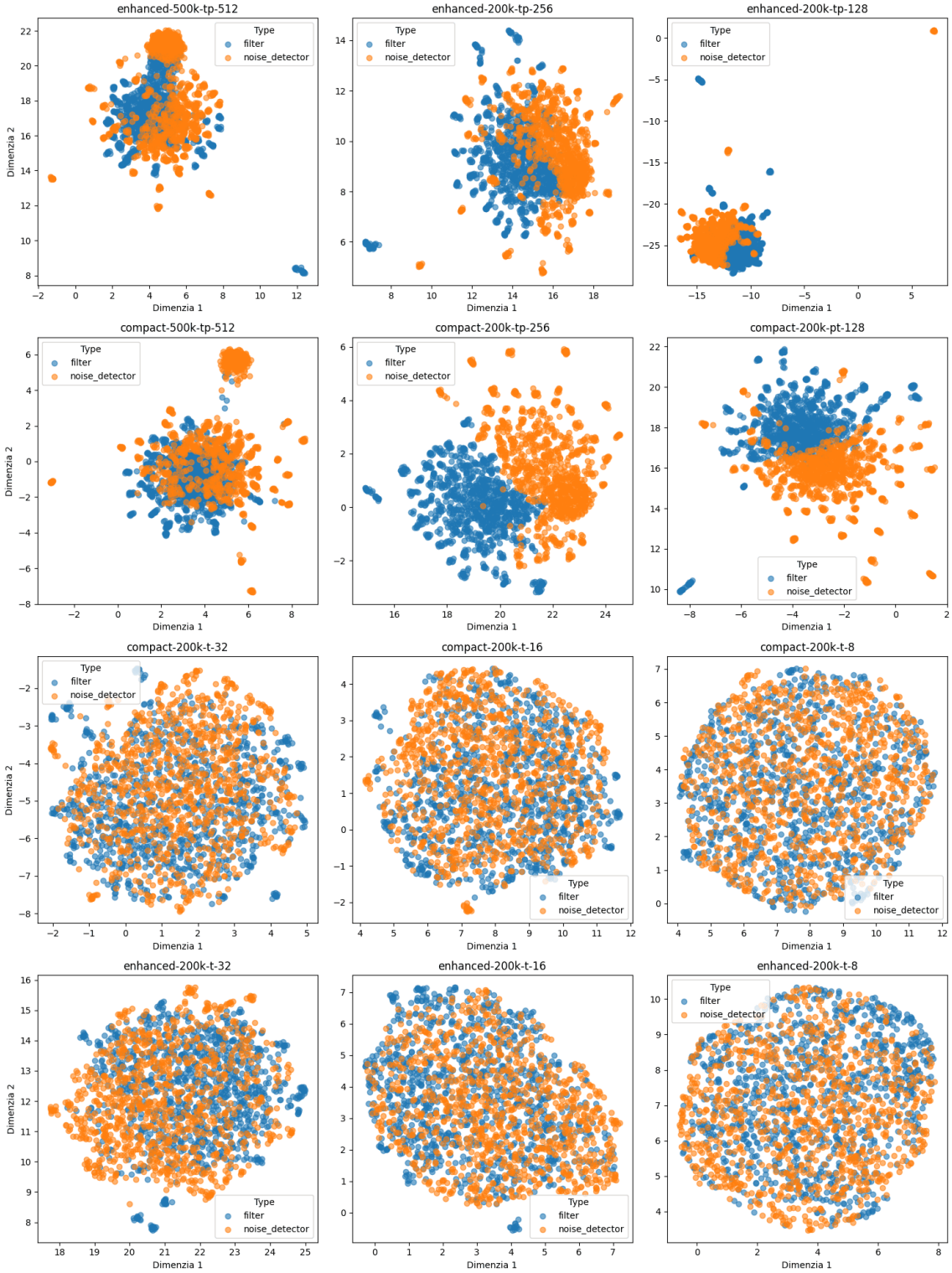
[69] Weiss, K., Khoshgoftaar, T. M. a Wang, D. A survey of transfer learning. *Journal of Big Data.* May 2016, zv. 3, č. 1, s. 9. DOI: 10.1186/s40537-016-0043-6. ISSN 2196-1115. Dostupné z: https://doi.org/10.1186/s40537-016-0043-6.

[70] Zhu, X., Goldberg, A. B., Brachman, R. a Dietterich, T. *Introduction to Semi-Supervised Learning.* Morgan and Claypool Publishers, 2009. ISBN 1598295470.

Príloha A

Vizualizácia vektorových reprezentácii

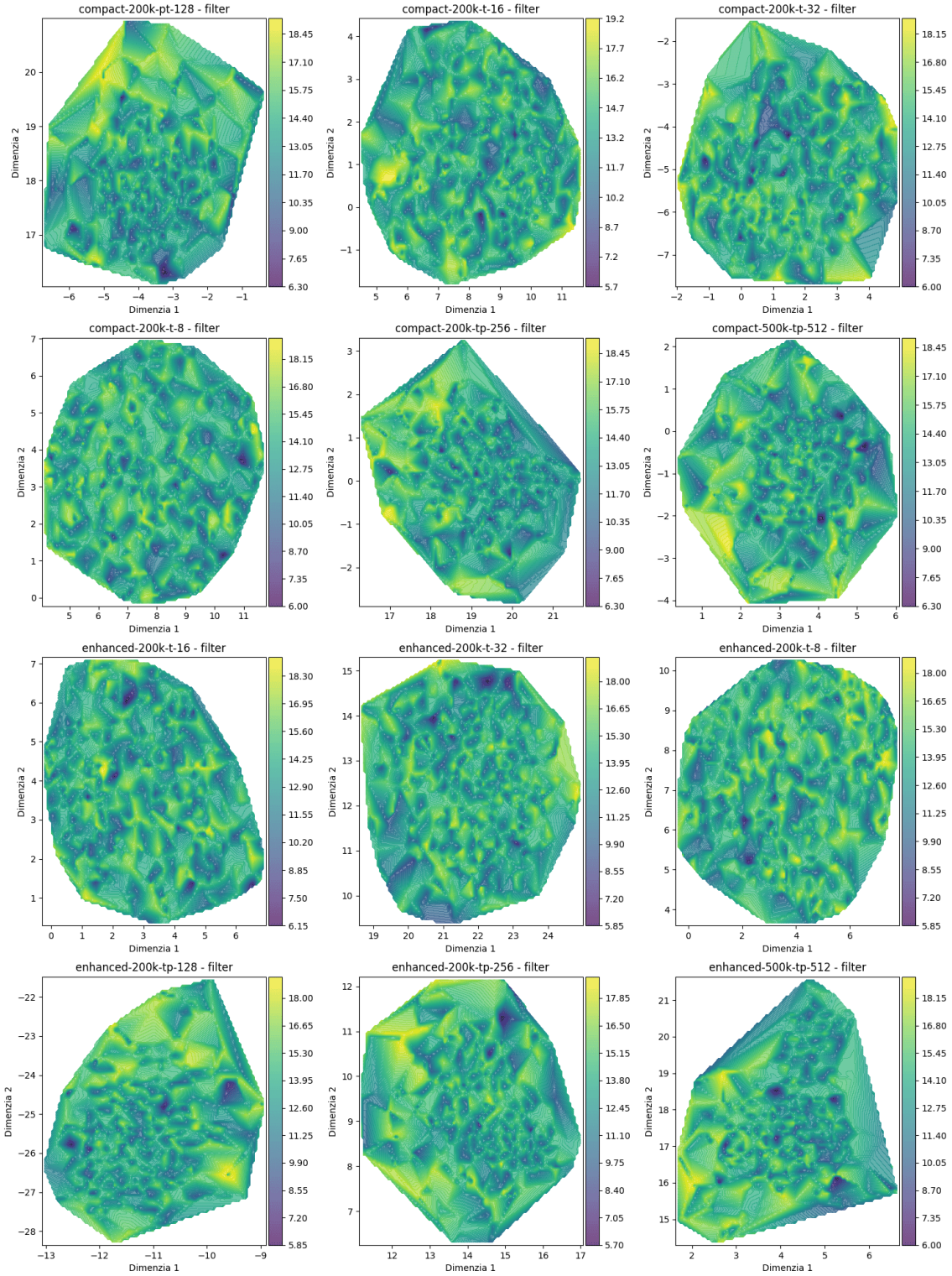
Na obr. A.1 je zobrazený kompletný výsledok vizualizácie vektorových reprezentácii kandidátných jedincov. Výsledok bol získaný procesom, ktorý je popísaný v kapitole 6.4. Bolo náhodne vybraných 1000 filtrov a 1000 detektorov z validačnej dátovej sady kandidátnych jedincov. Kandidátny jedinci boli tokenizovaní, zakódovaní do vektorovej reprezentácie pomocou kódovacej vrstvy transformátoru, agregovaní a normalizovaní. Normalizované vektory kandidátnych jedincov boli pomocou metódy UMAP, ktorá slúži na redukciu dimenzinality, premietnutí do 2D priestoru. Na ose X je zobrazená prvá dimenzia a na ose Y druhá dimenzia.



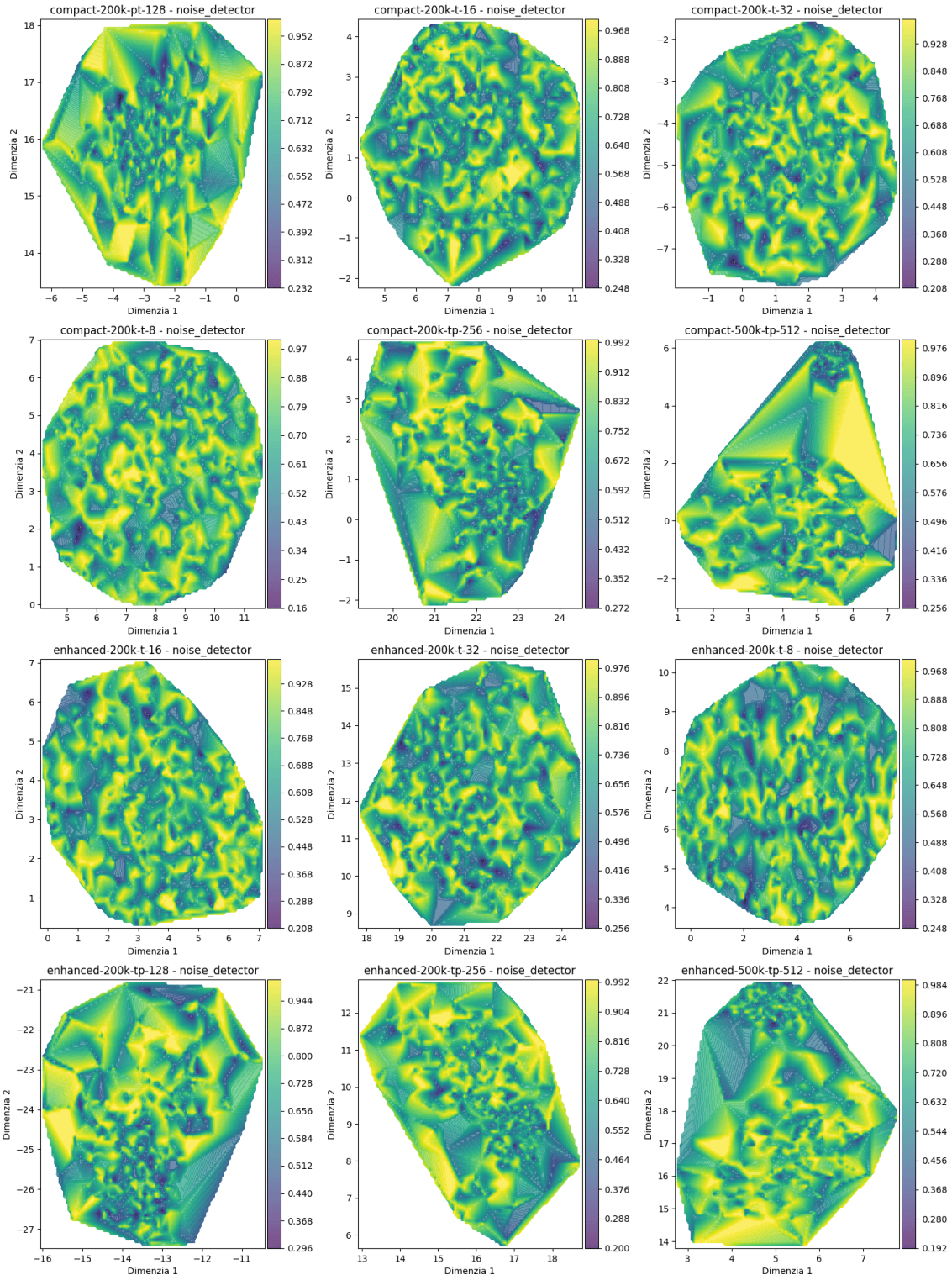
Obr. A.1: Kompletná vizualizácia novo vzniknutých reprezentácií kartézskych genetických programov v závislosti na type kandidátneho jedinca.

Na obr. A.2 a na obr. A.3 je zobrazená vizualizácia priemernej fitness hodnoty, ktorá bola vypočítaná spriemerovaním filtrácie na testovacom obrázku poškodenom na piatich

rôznych intenzitách šumu. Vektorové reprezentácie boli získané rovnakým spôsobom ako pri obr. A.1 s tým, že boli odstránené odľahlé hodnoty pre lepšiu vizualizáciu.



Obr. A.2: Vizualizácia vzniknutých reprezentácií filtrov v závislosti na priemernej fitness.

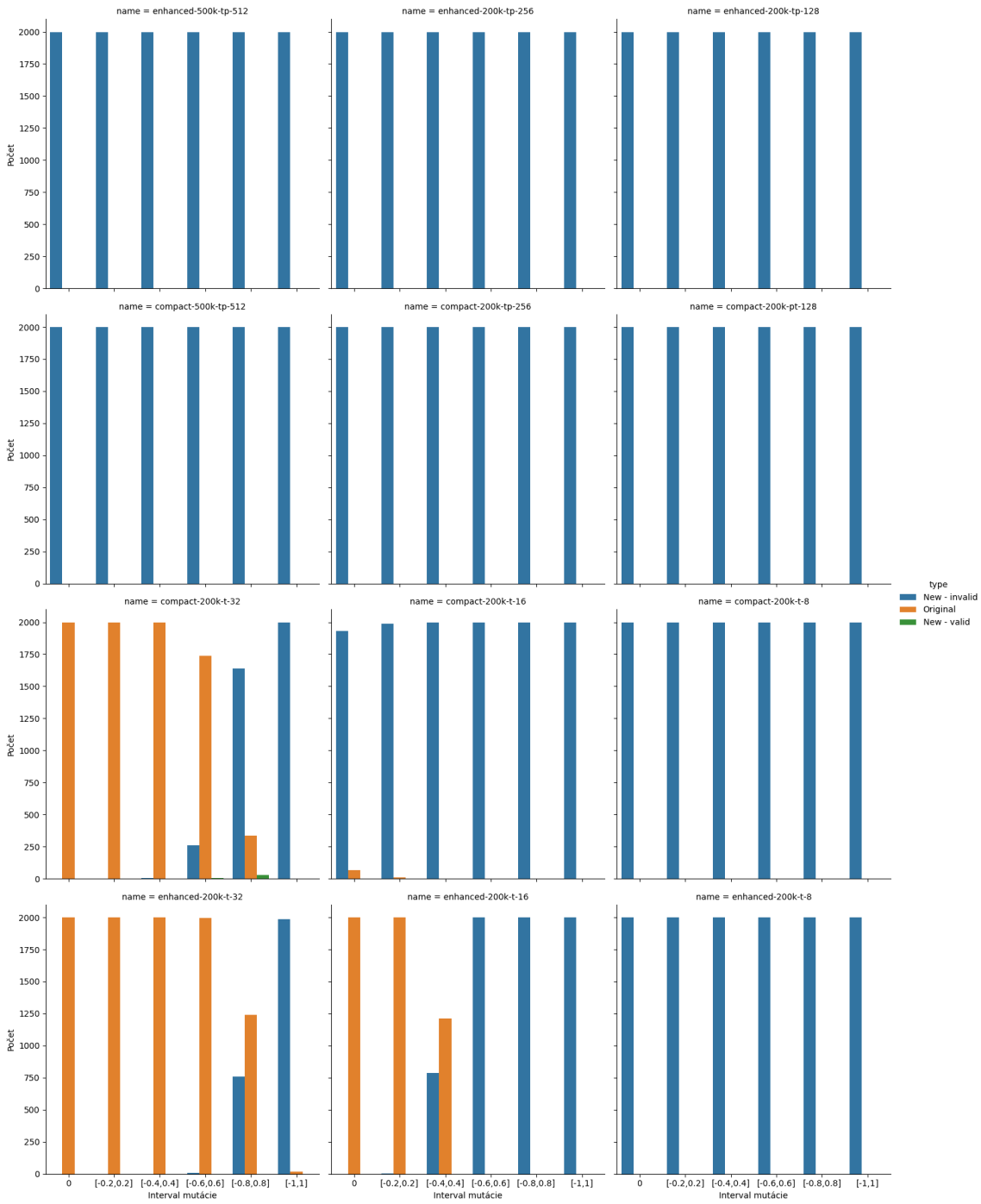


Obr. A.3: Vizualizácia vzniknutých reprezentácií detektorov v závislosti na priemernej fitness.

Príloha B

Mutácia založená na priamej úprave vektorovej reprezentácie génov

Na obr. B.1, sú zobrazené výsledky experimentu pre mutáciu založenú na priamej úprave vektorovej reprezentácie génov. Pri tomto experimente bolo vybraných 1000 filtrov a 1000 detektorov z validačnej dátovej sady kandidátnych riešení. Kandidátni jedinci boli prevedení do novej textovej reprezentácie, tokenizovaní, zakódovaní, upravení a znovu dekódovaní. Následne sa sledovalo, či vytvorené nové kandidátne riešenia sú validné, invalidné alebo originálne s pôvodným kandidátnym riešením. Na obrázku sú vidieť výsledky pre jednotlivé natrénované modely. Pri predikčnej úlohe nasledujúceho tokenu sa nedarilo zrekonštruovať ani pôvodný vstup. Pri úlohe rekonštrukcie sa to darilo, ale akonáhle sa začal pridávať šum do vektorovej reprezentácie génov, tak dekóder nedokázal dekódovať validné CGP programy.

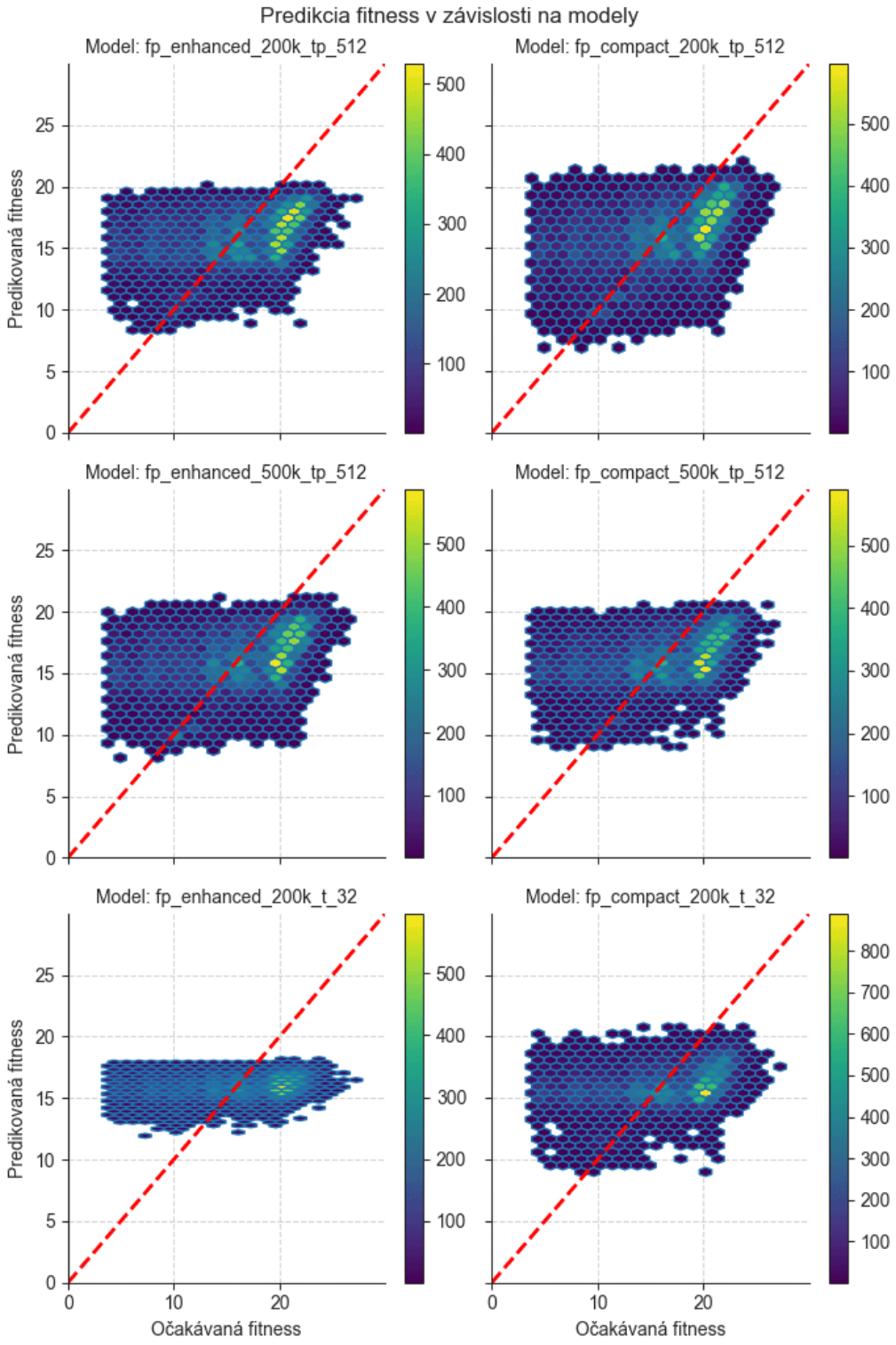


Obr. B.1: Obrázok zobrazuje mutáciu založenú na priamej úprave vektorovej reprezentácie

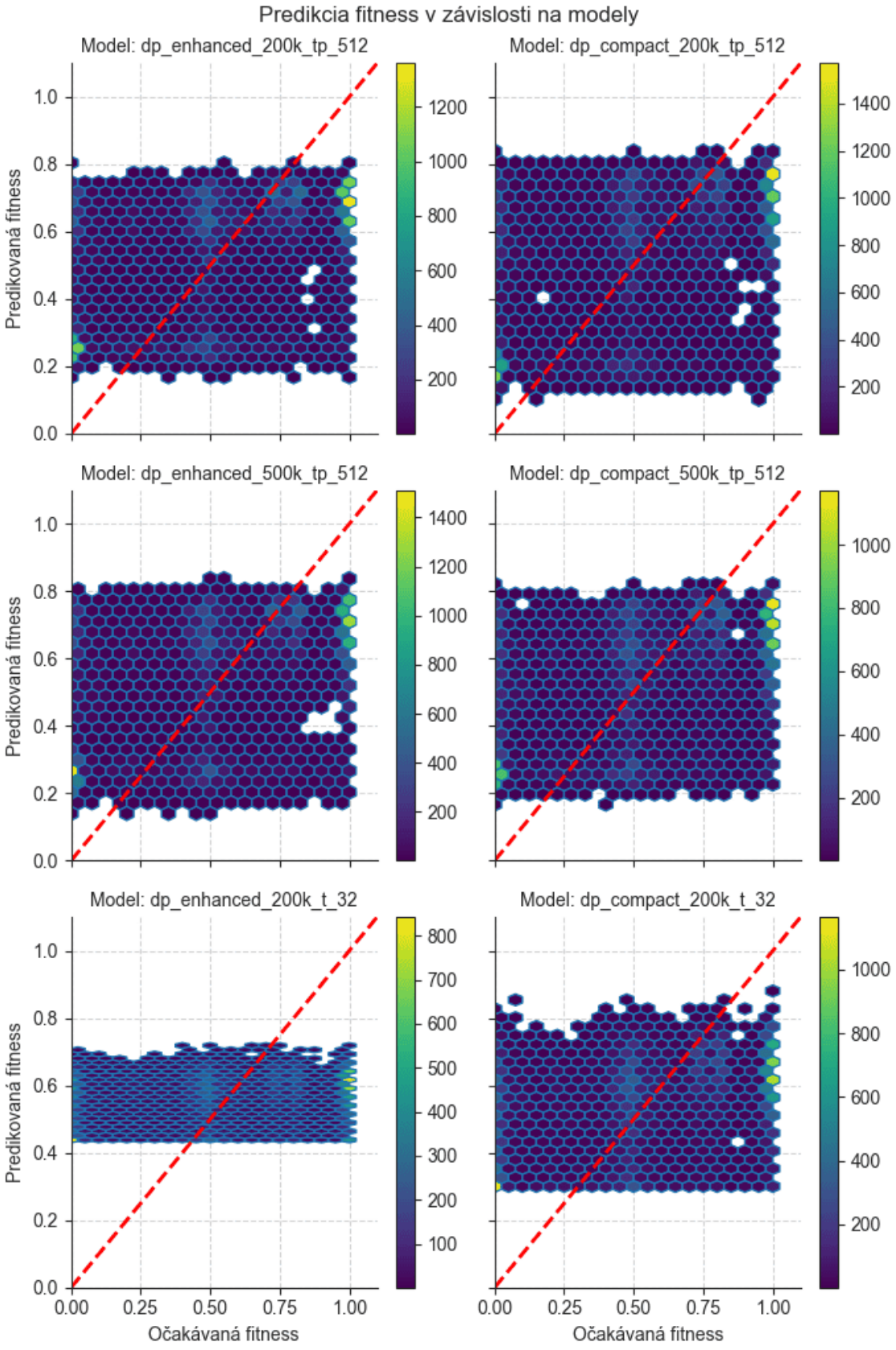
Príloha C

Trénovanie prediktora fitness

Na obr. C.1 je graf zobrazujúci predikovanú fitness v závislosti na očakávanej fitness pre prediktory fitness hodnoty filtrov. Z grafu je možné pozorovať menšiu koreláciu. Na nasledujúcom obr. C.2 je možné vidieť rovnaký typ grafu, ale pre prediktory fitness hodnoty detektorov. Na tomto obrázku nie je na prvý pohľad vidieť korelácia, ale z tabuľky 8.11 je vidieť menšiu koreláciu medzi predikovanými a očakávanými hodnotami fitness.



Obr. C.1: Obrázok zobrazuje predikciu fitness pre filtre



Obr. C.2: Obrázok zobrazuje predikciu fitness pre detektory