
**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**
**BRNO UNIVERSITY OF TECHNOLOGY**

**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ**
**FACULTY OF INFORMATION TECHNOLOGY**

**ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ**
**DEPARTMENT OF COMPUTER SYSTEMS**


# STROJOVÉ UČENÍ REPREZENTACE PRO GENETICKÉ PROGRAMOVÁNÍ

**MACHINE LEARNING OF REPRESENTATIONS IN GENETIC PROGRAMMING**

**DIPLOMOVÁ PRÁCE**
**MASTER'S THESIS**


**AUTOR PRÁCE** **ŠIMON POMYKAL**
**AUTHOR**

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



**BRNO 2024**

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

# Zadání diplomové práce

154385

| | |
|---|---|
| Ústav: | Ústav počítačových systémů (UPSY) |
| Student: | **Pomykal Šimon, Bc.** |
| Program: | Informační technologie a umělá inteligence |
| Specializace: | Strojové učení |
| Název: | **Strojové učení reprezentace pro genetické programování** |
| Kategorie: | Umělá inteligence |
| Akademický rok: | 2023/24 |

Zadání:

1. Seznamte se s metodami strojového učení, které se používají v oblasti automatického návrhu reprezentace pro danou úlohu. Zaměřte se na využití hlubokých neuronových sítí pro potřeby reprezentace stromových struktur v genetickém programování.
2. S využitím neuronové sítě navrhněte metodu pro automatizované vytvoření vhodné reprezentace pro genetické programování. Navrhněte využití takto vytvořené reprezentace pro aplikace v oblasti zpracování obrazu.
3. Vygenerujte nebo jinak získejte vhodnou datovou sadu pro potřeby automatického vytváření reprezentace.
4. Implementujte metodu popsanou 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 evolučního návrhu pomocí genetického programování.
6. Ověřte funkčnost a chování vytvořené reprezentace ve zvolené úloze z oblasti zpracování obrazu.
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

Cílem této práce je seznámit se s metodami strojového učení, které se využívají pro automatický návrh reprezentace. Speciálně se poté práce zaměřuje na hluboké učení v oblasti genetického programování (GP). Jako případová studie je zvoleno zpracování obrazu, a to zejména metody odstranění šumu. Spojením získaných poznatků je navržena nová reprezentace, jejímž účelem je nahradit syntaktický strom v algoritmu GP. Tato metoda je získána pomocí neuronové sítě typu transformer. Na závěr je vytvořena upravená varianta GP, která pracuje s novou reprezentací. Tato varianta je v několika experimentech porovnána s GP, který používá původní reprezentaci.



# Abstract

The aim of this thesis is to become acquainted with machine learning methods that are used for the automatic design of representations. Specifically, the work focuses on deep learning in the field of genetic programming (GP). Image processing is chosen as a case study, particularly noise reduction methods. By combining the acquired knowledge, a new representation is proposed, intended to replace the syntactic tree in the GP algorithm. This method is obtained using a transformer-type neural network. In conclusion, a modified version of GP that works with the new representation is created. This variant is compared with the original GP using the traditional representation in several experiments.



# Klíčová slova

genetické programování, neuronové sítě, hluboké učení, učení reprezentací, zpracování obrazu

# Keywords

genetic programming, neural networks, deep learning, representation learning, image processing


# Citace

POMYKAL, Šimon. *Strojové učení reprezentace pro genetické programování.* Brno, 2024. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Lukáš Sekanina, Ph.D.

# Strojové učení reprezentace pro genetické programování

## Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana prof. Ing. Lukáše Sekaniny Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Šimon Pomykal
15. května 2024


## Poděkování

Rád bych poděkoval vedoucímu práce prof. Ing. Lukáši Sekaninovi Ph.D. za odbornou pomoc při řešení této práce.

# Obsah


**1 Úvod** **6**

**2 Umělé neuronové sítě** **7**
2.1 Neuron . . . 8
2.2 Učení umělých neuronových sítí pomocí algoritmu back-propagation . . . 10
2.3 Učení reprezentace . . . 11
2.3.1 Obecné aprioritní předpoklady . . . 11
2.4 Hluboké neuronové sítě a hluboké učení . . . 13
2.5 Konvoluční neuronové sítě . . . 14
2.6 Transformery . . . 15
2.6.1 Mechanismus attention . . . 16
2.6.2 Enkodér . . . 17
2.6.3 Dekodér . . . 18

**3 Evoluční algoritmy** **19**
3.1 Pojmy . . . 20
3.1.1 Kandidátní řešení . . . 20
3.1.2 Reprezentace . . . 20
3.1.3 Populace . . . 21
3.1.4 Generace . . . 21
3.1.5 Fitness a fitness funkce . . . 21
3.1.6 Genetické operátory . . . 21
3.2 Evoluční programování . . . 21
3.3 Evoluční strategie . . . 22
3.4 Genetické algoritmy . . . 24
3.5 Genetické programování . . . 26
3.5.1 Reprezentace . . . 27
3.5.2 Inicializace populace . . . 27
3.5.3 Selekce . . . 29
3.5.4 Křížení a mutace . . . 29
3.6 Kartézské genetické programování . . . 31

**4 Filtrace obrazu** **32**
4.1 Problematika šumu obrazu . . . 32
4.1.1 Gaussovský šum . . . 33
4.1.2 Impulsní šum . . . 35
4.2 Genetické programování pro evoluční návrh obrazových filtrů . . . 36
4.2.1 Funkce posuvného okna . . . 37

4.2.2 Konvenční filtry . . . 37
4.2.3 Evoluční návrh filtrů . . . 38

**5 Návrh řešení** **39**
5.1 Navržená reprezentace - Distributed embedding . . . 39
5.2 Předzpracování syntaktického stromu . . . 40
5.3 Transformer . . . 40
5.4 Získání trénovacích dat . . . 42
5.5 Algoritmus genetického programování . . . 43

**6 Implementace** **44**
6.1 Algoritmus genetického programování . . . 44
6.1.1 Reprezentace syntaktického stromu . . . 45
6.1.2 Inicializace populace . . . 47
6.1.3 Evoluce . . . 48
6.2 Transformer . . . 48

**7 Experimenty** **51**
7.1 Tvorba datasetu . . . 52
7.2 Trénování modelu transformeru . . . 53
7.3 Přehled modelů . . . 55
7.4 Vyhodnocení enkodéru a dekodéru modelu . . . 56
7.5 Inicializace, mutace a křížení v nové reprezentaci . . . 58
7.6 Evoluce s využitím nové reprezentace . . . 59
7.7 Využití nové reprezentace v GP pro detekci hran . . . 62

**8 Závěr** **67**

**Literatura** **68**

**A Obsah paměťového média** **71**

# Seznam obrázků


2.1 Typický příklad dopředné plně propojené vícevrstvé sítě. Převzato z [23]. . 8
2.2 Zjednodušené schéma biologického neuronu. Převzato z [26]. . . . . 9
2.3 Schéma umělého neuronu. Převzato z [9]. . . . . 10
2.4 Ilustrace hlubokého učení ve vztahu ke strojovému učení a umělé inteligenci, převzato z [20]. . . . . 13
2.5 Aplikace jádra na část vstupu v konvoluční vrstvě. Centrální bod zpracovávané části je nahrazen váženým součtem celé části. Váhy jednotlivých bodů jsou určeny pomocí jádra. Převzato z [16] . . . . . 14
2.6 Architektura jednoduché konvoluční sítě s pěti vrstvami. Převzato z [16] . . 15
2.7 Architektura modelu transformeru typu enkodér-dekodér. Převzato z [25]. . 16
2.8 Scaled Dot-Product Attention (vlevo) a Multi-Head Attention (vpravo). Převzato z [25]. . . . . 17

3.1 Porovnání idealizované darwinovské evoluce a digitální teorie [18]. . . . . 20
3.2 Schéma základního algoritmu evolučního programování. . . . . 22
3.3 Schéma základního algoritmu evoluční strategie. . . . . 23
3.4 Příklad reprezentace 4 chromozomů genetického algoritmu v binárním kódování délky 8. . . . . 25
3.5 Příklad ohodnocení jedinců z populace pro vytvoření ruletového kola. . . . . 25
3.6 Demonstrace tvorby potomků pomocí jednobodového křížení (nahoře) a více bodového křížení (dole). . . . . 26
3.7 Vytvoření potomků pomocí genetického operátoru mutace. . . . . 26
3.8 GP syntaktický strom reprezentující program $\max(x + x, x + 3y)$, převzato z [17]. . . . . 27
3.9 Tvorba syntaktického stromu pomocí metody full s maximální hloubkou 2, převzato z [17]. . . . . 28
3.10 Tvorba syntaktického stromu pomocí metody growth s maximální hloubkou 2, převzato z [17]. . . . . 29
3.11 Příklad křížení podstromů v GP, převzato z [17]. . . . . 30
3.12 Na obrázku je zobrazeno pole uzlů reprezentujících výpočetní funkce z look-up tabulky. Pole má $n_c$ sloupců a $n_r$ řádků. Počet vstupů programu je roven $n_i$, počet výstupů programu je roven $n_o$. Každý uzel má počet vstupů roven maximální aritě funkce $a$. Každá vstup má unikátní označení. Převzato z [11]. 31

4.1 Porovnání originálního obrázku (vlevo) a obrázku s Gaussovským šumem se střední hodnotou 0 a směrodatnou odchylkou 30 (vpravo)[5]. . . . . 33
4.2 Příklad grafu jednorozměrného Gaussova rozložení. Převzato z [14]. . . . . 34
4.3 Příklad grafu vícerozměrného Gaussova rozložení. Převzato z [6]. . . . . 35

4.4 Porovnání originálního obrázku (vpravo) a obrázku s 15% šumem sůl a pepř (vlevo) [5]. . . . . 36
4.5 Demonstrace využití funkce posuvného okna, převzato z [13]. . . . . 37
4.6 Mediánový filtr, převzato z [4]. . . . . 37
4.7 Seznam funkcí které lze použít v uzlech, převzato z [21]. . . . . 38

5.1 Proces převodu syntaktického stromu na tokenizovaný strom, převzato z [7]. 40
5.2 Proces trénování transformeru, převzato z [7]. . . . . 41
5.3 Využití seq-to-seq modelu Transformeru pro trénování převodu syntaktického stromu do nové reprezentace a zpět do původní podoby. Jednotlivé vstupy a výstupy jsou rozšířeny o speciální znaky $S$ - začátek sekvence, $E$ - konec sekvence. . . . . 42

6.1 Na obrázku lze vidět syntaktické strom reprezentující filtr pomocí jednotlivých uzlů (vpravo) a odpovídající pole hodnot z look-up tabulky, které je využito v algoritmu GP (vlevo). Reprezentace je v obou případech doplněna do maximální velikosti v dané hloubce o speciální uzly s hodnotou padding. V případě syntaktického stromu se jedná o šedé uzly s textem "pad", zatímco v případě pole se jedné o prvky s hodnotou 0. . . . . 45
6.2 Importování modelu Transformeru do v prostředí Kaggle jupiter notebooku. 49
6.3 Upravení prostředí Kaggle pro zajištění kompatibility s imporotvaným modelem transformeru. . . . . 49

7.1 Dvojice obrázků o velikosti $64 \times 64$ používaná při běžích GP v rámci experimentů. Referenční obrázek je vlevo a obrázek s 15% impulsním šumem vpravo. . . . . 51
7.2 Histogram zobrazující rozdělení velikostí jedinců v datasetu 192_96-48-48. . 52
7.3 Histogram zobrazující rozdělení velikostí jedinců v datasetu 192_96-48-48. . 53
7.4 Průběh trénování prvního modelu. . . . . 54
7.5 Průběh trénování prvního modelu s upraveným slovníkem, kde speciální symbol padd je brán jako běžné slovo. . . . . 54
7.6 Rozdělení velikostí a fitness funkcí korektních jedinců vygenerovaných inicializací matice náhodnými hodnotami z intervalu (-3 , 3). . . . . 58
7.7 Porovnání hodnot fitness funkcí v jednotlivých generacích 40 běhů GP využívajícího stromovou reprezentaci a 40 běhů GP upraveného pro běh s novou reprezentací. . . . . 60
7.8 Porovnání hodnot fitness funkcí nejlepších jedinců v jednotlivých generacích 40 běhů GP využívajícího stromovou reprezentaci a 40 běhů GP upraveného pro běh s novou reprezentací. . . . . 61
7.9 Porovnání doby trvání jednotlivých generacích 40 běhů GP využívajícího stromovou reprezentaci a 40 běhů GP upraveného pro běh s novou reprezentací. 62
7.10 Porovnání hodnot fitness funkcí v jednotlivých generacích 10 běhů GP využívajícího stromovou reprezentaci a 10 běhů GP upraveného pro běh s novou reprezentací v úloze detekce hran. . . . . 63
7.11 Porovnání hodnot fitness funkcí nejlepších jedinců v jednotlivých generacích 10 běhů GP využívajícího stromovou reprezentaci a 10 běhů GP upraveného pro běh s novou reprezentací v úloze detekce hran. . . . . 64

7.12 Porovnání doby trvání jednotlivých generacích 10 běhů GP využívajícího stromovou reprezentaci a 10 běhů GP upraveného pro běh s novou reprezentací v úloze detekce hran. . . . . 65
7.13 Jeden z testovací obrázků o velikosti $256 \times 256$ zpracovaný pomocí filtru z nejlepších běhů GP-Emb a GP-Tree. . . . . 66

# Kapitola 1

# Úvod

Kvalita metod strojového učení je výrazně závislá na výběru reprezentace dat, se kterými pracují. Z tohoto důvodu je velká část úsilí při implementaci algoritmů strojového učení věnována návrhu procesů předzpracování a transformace těchto vstupních dat. V současnosti dosahují v tomto oboru nejlepších výsledků modely využívající hluboké učení. Správně zvolená reprezentace umožňuje algoritmu strojového učení lépe získávat informace ze vstupních dat a dosáhnout lepších výsledků [3].

Reprezentace získaná pomocí hlubokého učení by mohla například nahradit syntaktické stromy, používané v genetickém programování (GP), které řeší problematiku evoluce filtrů pro odstranění šumu, detekci hran atd. Cílem této práce je seznámit se oblastmi učení reprezentace, hlubokého učení, zpracování obrazu a genetického programování a spojením znalostí z těchto oblastí navrhnout automatizovaný způsob pro získání reprezentace využitelné v úlohách genetického programování pro zpracování obrazu na základě neuronové sítě včetně datové sady pro její učení.

Předlohou pro tuto práci je článek „Symbolic Regression Trees as Embedded Representations“ z roku 2023 [7], který popisuje model určený pro zpracování přirozeného jazyka jako nástroj pro tvorbu nové reprezentace, která má potenciál zachytit sémantické informace reprezentovaného jedince. Reprezentace však postrádá schopnost zpětného převedení, a proto je její potenciální využití omezené. Jedním z možných využití je například nahrazení metriky editační vzdálenosti stromu, která se používá v GP.

Reprezentace navržená v této práci vychází ze stejných principů jako [7], avšak je opravena tak, aby ji bylo možné transformovat zpět do původní stromové reprezentace. Nová reprezentace je vyvíjena v GP pro odstranění impulsního šumu, což je jeden z úkolů v kontextu zpracování obrazu.

V experimentální části práce bude reprezentace vyhodnocena a využita v upravené verzi GP, který bude schopný s touto reprezentací pracovat. Struktura diplomové práce je popsána v následujícím odstavci.

Kapitola 2 nejprve obecně popisuje neuronové sítě a algoritmus back-propagation pro jejich učení. Poté je popsána tématika učení reprezentace se zaměřením na hluboké neuronové sítě. V kapitole 3 jsou popsány evolučních algoritmy včetně jednotlivých typů těchto algoritmů. Speciální pozornost je věnována genetickému programování. Problematika šumu obrazu a evoluční návrh filtrů pro odstranění tohoto šumu je popsán v kapitole 4. Kapitoly 5 a 6 popisují návrh a implementaci veškerých částí potřebných během procesu tvorby nové reprezentace. Samotný proces tvorby reprezentace a její následné vyhodnocení je v kapitole 7.

# Kapitola 2

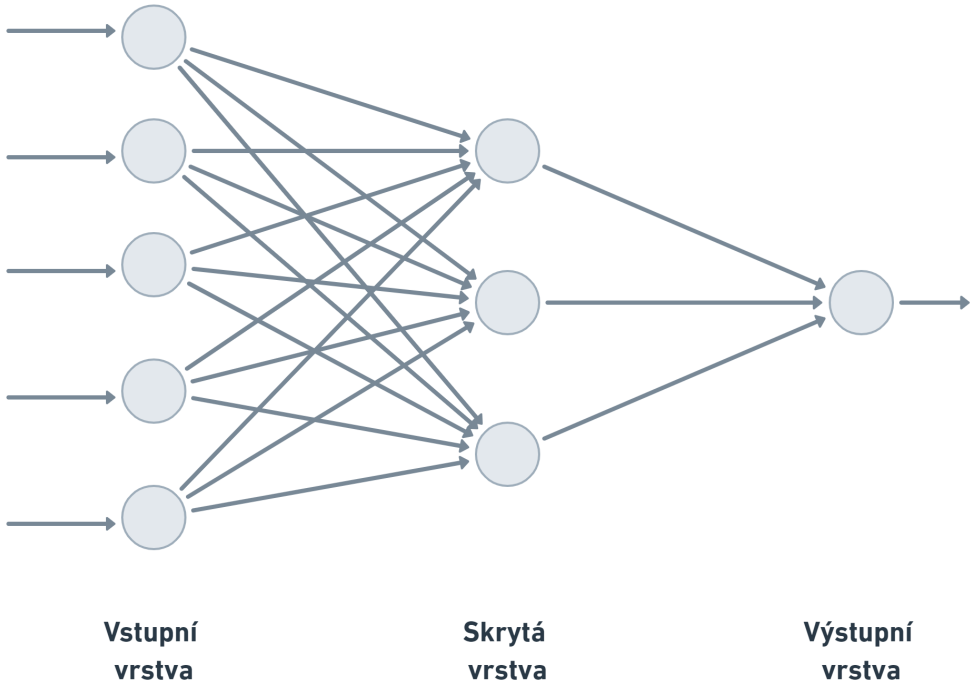
# Umělé neuronové sítě

Umělá neuronová síť (ANN) je síť jednoduchých výpočetních prvků - neuronů, které mezi sebou komunikují. Koncept ANN vychází ze struktury a fungování mozku [23]. Z biologického hlediska je základním požadavkem na ANN, aby se snažila co nejpřesněji napodobit fungování a chování reálné sítě - mozku [8].

Motivací pro rozvoj tohoto oboru bylo zjištění, že lidský mozek zpracovává informace zcela jiným způsobem než klasický digitální počítač. Mozek je velmi komplexní, nelineární a paralelní výpočetní systém, který má schopnost měnit své výpočetní prvky (neurony) za účelem provedení specifických výpočtů mnohonásobně rychleji. [9].

Je zřejmé, že výpočetní síla ANN je odvozena jednak z její distribuované a paralelní struktury a také z její schopnosti učit se a generalizovat. Generalizace je schopnost ANN dobře fungovat i při práci s dosud neviděnými daty. Díky těmto vlastnostem je ANN schopna řešit poměrně komplexní problémy. V praxi však nejsou komplexní úlohy řešeny pouze jednou samostatnou ANN, nýbrž jsou rozděleny na jednotlivé podúlohy, které jsou poté řešeny systémem, do něhož je ANN (nebo i více ANN) zakomponována [9].

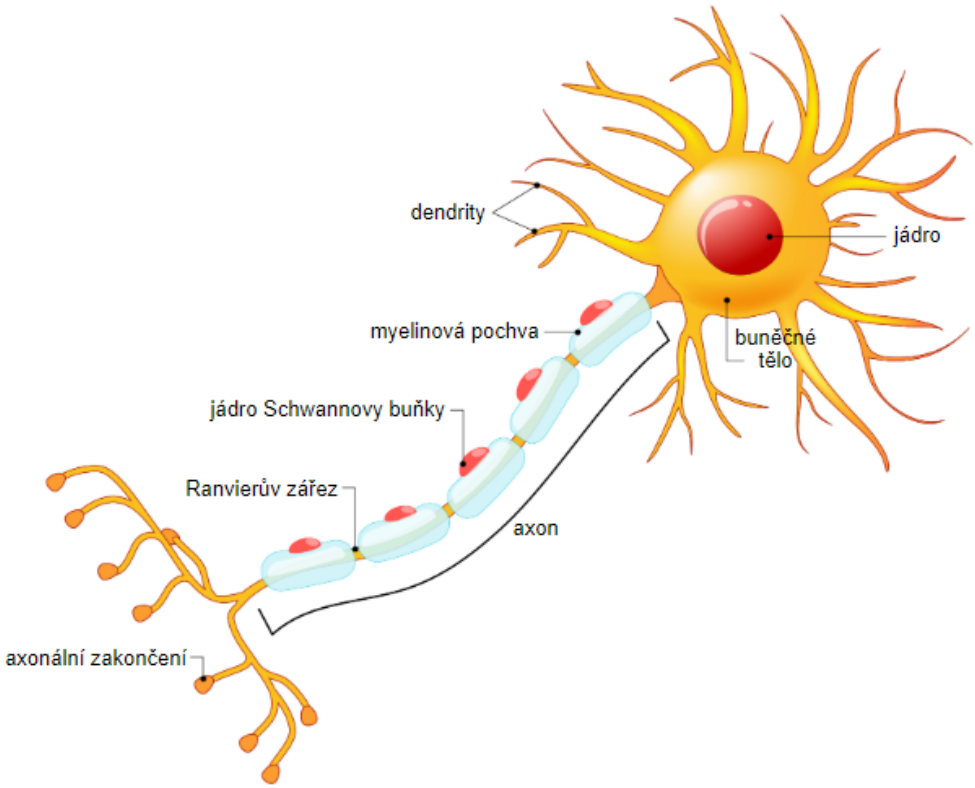
Nejrozšířenějším typem ANN jsou vícevrstvé dopředné (MLF) neuronové sítě, kterou lze vidět na obrázku 2.1 . Neurony v těchto sítích jsou organizovány do vrstev. První vrstva se nazývá vstupní, poslední vrstva se nazývá výstupní. Ostatní vrstvy se nazývají skryté. Síť se navíc označuje jako plně propojená, je-li každý neuron jedné vrstvy jednosměrně propojen s každým neuronem následující vrstvy. Směr toku informací v takovéto sítě je pouze jednosměrný a to ve směru od vstupní vrstvy k výstupní. ANN s obousměrným tokem informací se nazývají rekurentní neuronové sítě [23].




Obrázek 2.1: Typický příklad dopředné plně propojené vícevrstvé sítě. Převzato z [23].

## 2.1 Neuron

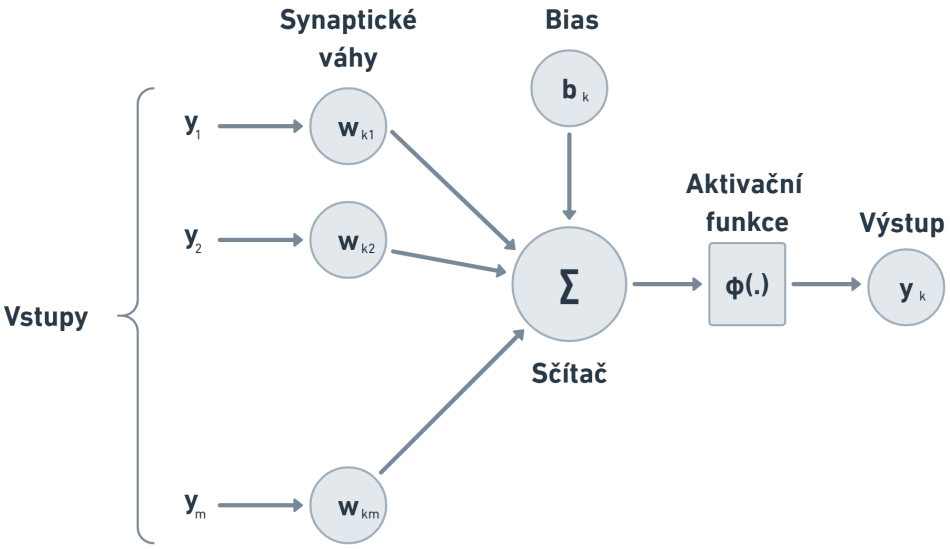
Fungování bilogických neuronů je mimořádně složité. Nicméně, zjednodušený proces zpracování informací lze shrnout takto: Neuron přijímá vstupní signály z ostatních neuronů. Tyto signály jsou modulovány synaptickým mechanismem na postsynaptický potenciál (PSP ). Jednotlivé PSP jsou integrovány dendritickými výběžky v prostoru a čase. PSP mohou být excitační nebo inhibiční a jejich kombinovaný výsledek je změnou membránového potenciálu, který může buď excitovat nebo inhibovat neuron. Dynamika membrány v důsledku těchto změn je složitá, ale v mnoha případech lze předpokládat, že existuje prahová hodnota membránového potenciálu, po jejímž překročení je generován akční potenciál a pod kterou k takové události nedochází. Řetězec akčních potenciálů tvoří výstup neuronu. Ten putuje od buňky po axonu až k axonálním zakončením, kde se celý tento postup opakuje. Informace jsou v neuronech kódovány různými způsoby, ale běžnou metodou je využití frekvence nebo rychlosti produkce akčních potenciálů [8].




Obrázek 2.2: Zjednodušené schéma biologického neuronu. Převzato z [26].

Umělý neuron je základní výpočetní jednotka neuronové sítě. Zjednodušené schéma biologického neuronu, ze kterého vychází neuron používaný v ANN, lze vidět na obrázku 2.2. Umělý neuron lze rozdělit na tři základní části:

- **Synapse:** jedná se o množinu vstupů neuronu, z nichž každý je navíc charakterizován vahou. Vstupní signál na synapsi je vždy touto vahou vynásoben před vstupem do další části neuronu. Na rozdíl od reálných synapsí v mozku mohou mít synapse umělého neuronu nastaveny váhy v kladných i záporných hodnotách [9].
- **Sčítač:** slouží k sečtení vstupů ze všech synapsí po vynásobení jejich příslušnými vahami. Součástí sčítače může být i tzv. bias, což je hodnota uložená v neuronu, která je přičtena k celkovému součtu sčítače [9].
- **Aktivační funkce:** převádí celkový vstupní signál na výstup neuronu. Může být jak lineární tak nelineární. Zároveň určuje výstupní formát neuronu [9].




Obrázek 2.3: Schéma umělého neuronu. Převzato z [9].

Matematicky lze umělý neuron zachycený na obrázku 2.3 popsat rovnicemi:

$$u_k = \sum_{j=1}^{m} w_{kj} x_j$$

$$y_k = \phi(u_k + b_k),$$

kde $x_1, x_2, \ldots, x_m$ jsou vstupní signály neuronu, $w_{k1}, w_{k2}, \ldots, w_{km}$ jsou synaptické váhy, $u_k$ je výstup sčítače neuronu, $b_k$ je bias, $\phi$ je aktivační funkce a $y_k$ je výstupní signál neuronu.

## 2.2 Učení umělých neuronových sítí pomocí algoritmu back-propagation

Back-propagation (BP) iterativní algoritmus, který je využíván při učení dopředných neuronových sítí pomocí optimalizační metody gradientního sestupu. Cílem tohoto algoritmu je minimalizovat objektivní funkci vyjadřující odchylku odezvy sítě od požadovaných hodnot [15].

Derivace BP je přímá aplikace metody gradientního sestupu pro optimalizaci a je závislá na definici chybové funkce (loss funkce) sítě. Mezi běžně používané chybové funkce patčí například entropy error (CE) a summed squared error (SSE) [15].

$$E^{SSE} = \sum_{i \in O_{out}} \sum_{s \in S_{Train}} (T_i(s) - r_i(s))^2$$

$$E^{CE} = \sum_{i \in O_{out}} \sum_{s \in S_{Train}} \left[ T_i(s) \ln(r_i(s)) - (1 - T_i(s)) \ln(1 - r_i(s)) \right],$$

kde $O_{\text{out}}$ jsou výstupy neuronové sítě, $S_{\text{Train}}$ jsou vzorky z trénovací sady $S_{\text{Train}}$, $T_i(s)$ je skutečná hodnota pro výstup $i$ a vzorek trénovací sady $s$, $r_i(s)$ je predikovaná hodnota pro výstup $i$ a vzorek trénovací sady $s$ produkovaná modelem [15].

Každý váhový parametr $w_{ji}$ (váha spojení od neuronu $j$ do neuronu $i$) je aktualizován o množství úměrné negativnímu gradientu chyby vzhledem k E:

$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}},$$

kde velikost kroku $\eta$ reflektuje kompromis mezi hladkým průběhem konvergence a rychlostí konvergence. Parametr $\eta$ se nazývá learning rate.

## 2.3 Učení reprezentace

Kvalita metod strojového učení je výrazně závislá na výběru reprezentace dat, se kterou pracuje. Z tohoto důvodu je velká část úsilí při implementaci algoritmů strojového učení věnována návrhu procesů předzpracování a transformace těchto vstupních dat. Cílem tohoto úsilí je najít vhodnou reprezentaci, která umožňuje efektivní strojové učení. Tento proces je sice důležitý, ale vyžaduje hodně práce a poukazuje na slabiny současných učících algoritmů: jejich neschopnost extrahovat a organizovat diskriminační informace z dat samotných. Tvorba příznaků je způsob, jak využít lidskou důmyslnost a předchozí znalosti o problému ke kompenzaci této slabiny. Aby bylo možné rozšířit možnosti využití strojového učení a zjednodušit jeho aplikaci, je velmi žádoucí, aby algoritmy strojového učení byly méně závislé na této činnosti, která vyžaduje práci člověka s expertní znalostí o daném problému [3].

Jedná se tedy o učení se reprezentovat data takovým způsobem, který usnadňuje extrakci užitečných informací při vytváření klasifikátorů nebo jiných prediktorů. V případě pravděpodobnostních modelů je dobrá reprezentace často taková, která odhaluje posteriorní rozdělení základních vysvětlujících faktorů pro pozorovaný vstup. Dobrá reprezentace by měla být také užitečná jako vstup pro supervizovaný prediktor [3].

Učení reprezentace se stalo samostatným oborem v komunitě strojového učení, který je pravidelně předmětem workshopů na předních konferencích jako NIPS[1] a ICML[2], a dalších konferencích zaměřených speciálně na tento obor, jako je ICLR[3]. Často je tato oblast spojována s hlubokým učením nebo učením funkcí. Rychlý vzestup vědeckých aktivit v oblasti učení reprezentace byl podpořen a posílen řadou pozoruhodných empirických úspěchů, a to jak v akademickém prostředí, tak v průmyslu [3].

### 2.3.1 Obecné aprioritní předpoklady

Správně zvolené reprezentace mohou vyjadřovat obecné aprioritní předpoklady o světě kolem nás, tedy předpoklady takové, které nejsou specifické pro danou úlohu, ale pravděpodobně by mohly být užitečné pro učící se algoritmus. Příklady obecně použitelných aprioritních předpokladů: [3]:

- **Hladkost:** základní apriorní předpoklad ve strojovém učení, který předpokládá, že pokud jsou dva vstupy, $x$ a $y$, podobné, pak by měly být podobné i odpovídající výstupy, $f(x)$ a $f(y)$.

[1] Conference on Neural Information Processing Systems
[2] International Conference on Machine Learning
[3] International Conference on Learning Representations

Platí tedy:

$$x \approx y \Rightarrow f(x) \approx f(y)$$

Ačkoli je tento předpoklad běžně používán, sám o sobě nedokáže překonat problémy spojené se zpracováním dat s vysokou dimenzionalitou. K účinnému řešení tohoto problému jsou potřeba další techniky a přístupy [3].

- **Více vysvětlujících faktorů:** distribuce generující data je tvořena různými základními faktory a ve většině případů se to, co se naučíme o jednom faktoru, generalizuje v mnoha konfiguracích dalších faktorů. Cílem je získat nebo alespoň rozlišit tyto základní faktory variace. Tento předpoklad stojí za myšlenkou distribuovaných reprezentací [3].

- **Hierarchická organizace vysvětlujících faktorů:** koncepty, které jsou užitečné pro popis světa kolem nás, mohou být definovány v termínech dalších konceptů v hierarchii, přičemž abstraktnější koncepty jsou umístěny výše v hierarchii a jsou definovány pomocí méně abstraktních konceptů. Toto předpokládání je využíváno u hlubokých reprezentací [3].

- **Semi-supervised learning:** se vstupy $X$ a cílovou proměnnou $Y$, kterou je třeba předpovědět, podmnožina faktorů vysvětlujících distribuci $X$ vysvětluje značnou část $Y$ za předpokladu $X$. Proto reprezentace, které jsou užitečné pro $P(X)$, mají tendenci být užitečné i při učení $P(Y|X)$ [3].

- **Faktory sdílené napříč úlohami:** s mnoha cílovými proměnnými $Y$, které nás zajímají, nebo obecně mnoha úlohami učení, úlohy (např. odpovídající $P(Y|X, uloha)$) jsou vysvětleny faktory, které jsou sdílené s jinými úlohami [3].

- **Podprostory:** Pravděpodobnostní distribuce se soustředí v oblastech, které mají mnohem menší dimenzionalitu než původní prostor, v němž data existují. Tento princip je explicitně využíván v některých algoritmech autoenkodérů a dalších algoritmech inspirovaných podprostory [3].

- **Přirozené shlukování:** jev, kdy jsou různé hodnoty kategorických proměnných, jako jsou třídy objektů, spojeny s odlišnými oblastmi prostoru. Tyto podprostory vykazují lokální variace, které mají tendenci zachovat kategorii a lineární interpolace mezi příklady různých tříd obecně zahrnuje průchod oblastí s nízkou hustotou pravděpodobnostní funkce, což vede k dobře odděleným oblastem pro různé kategorie. Tento koncept se využívá v úlohách strojového učení k předpovídání kategorických proměnných využitím statistické struktury pozorované v kategoriích a třídách pojmenovaných lidmi [3].

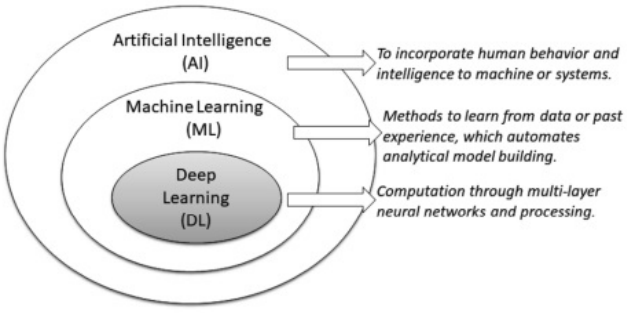
- **Temporální a prostorová koherence:** týká se pozorování, kdy po sobě jdoucí nebo blízké datové body často sdílí podobné hodnoty pro relevantní kategorické koncepty nebo vykazují malé změny na hustě osídleném podprostoru. Tento apriorní předpoklad naznačuje, že různé faktory se mění v různých měřítkách, a že zachycení takových kategorických proměnných může být dosaženo trestáním významných změn hodnot v čase nebo prostoru. Tento koncept byl představen Beckerem a Hintonem v roce 1992 [3].

- **Řídkost:** pro jakékoli dané pozorování $x$ je relevantní pouze malý zlomek možných faktorů. Z hlediska reprezentace by to mohlo být reprezentováno vlastnostmi, které jsou často nulové, nebo tím, že většina extrahovaných vlastností je nevnímavá k malým variacím x. Toho lze dosáhnout určitými formami apriorních předpokladů pro latentní proměnné, nebo použitím nelinearity [3].
- **Jednoduchost závislostí faktorů:** v dobrých reprezentacích vyšší úrovně jsou faktory navzájem spojeny prostřednictvím jednoduchých, typicky lineárních závislostí. To lze vidět v mnoha zákonech fyziky [3].

## 2.4 Hluboké neuronové sítě a hluboké učení

Díky vývoji efektivních architektur a algoritmů pro jejich učení jsou neuronové sítě důležitou součástí oblastí strojového učení a umělé inteligence již desítky let, vznikly ale dříve. Avšak navzdory jejich využití zájem o jejich další rozvoj poněkud opadl. To se změnilo po roce 2006, kdy byl představen koncept hlubokého učení (deep learning) a hlubokých neuronových sítí založený na umělých neuronových sítích. Představení hlubokého učení odstartovalo novou vlnu zájmu o oblast umělých neuronových sítí. Proto se také hluboké sítě někdy označují jako "neuronové sítě nové generace". Tomuto zájmu se hluboké učení dostalo zejména díky tomu, že modely založené na této metodě dosahovaly oproti tradičním neuronovým sítím a dalším metodám výborných výsledků v celé škále klasifikačních a regresních úloh [20].

V dnešní době je termín hluboké učení často zaměnitelně používán spolu se strojovým učením a umělou inteligencí k popisu systémů, jenž vyjadřuje známky inteligentního chování. Na obrázku 2.4 je vyjádřena pozice hlubokého učení ve srovnáním se strojovým učením a umělé inteligencí. Obecně umělá inteligence začleňuje lidské chování a inteligenci do strojů, zatímco strojové učení je metoda učení se z dat, která automatizuje vytváření analytických modelů. Hluboké učení je poté případ této metody, kdy výpočet probíhá prostřednictvím vícevrstvých neuronových sítí. Termín "hluboké"tedy odkazuje na koncept několika úrovní, kterými jsou data zpracována [20].




Obrázek 2.4: Ilustrace hlubokého učení ve vztahu ke strojovému učení a umělé inteligenci, převzato z [20].

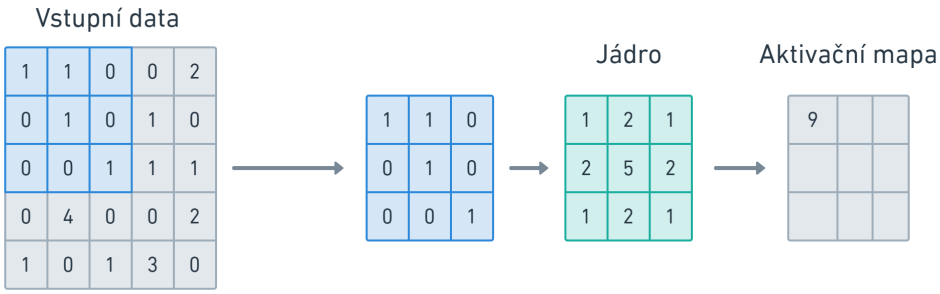
Při implementaci modelu, který využívá hluboké učení pro řešení problémů reálného světa, následujeme proces, který se skládá ze tří fází. V první fázi dochází k analýze dat a jejich přípravě, což zahrnuje čištění, normalizaci a transformaci dat pro další zpracování.

Druhá fáze se zaměřuje na navržení struktury modelu hlubokého učení a jeho trénování pomocí dostupných dat. Ve třetí a závěrečné fázi pak probíhá ověřování účinnosti modelu a interpretace jeho výsledků s cílem pochopit, jak model funguje a jak jsou jeho výstupy relevantní pro daný problém. Tento proces je podobný jako při implementaci modelu strojového učení, avšak na rozdíl od modelování strojového učení je extrakce příznaků v modelu hlubokého učení automatizovaná, nikoli manuální [20].

Největší rozdíl mezi hlubokým učením a klasickým strojovým učením je v případě kdy množství dat exponenciálně roste, kdy hluboké neuronové sítě dokonce fungují lépe s větším počtem dat. Tyto modely jsou ideální pro práci s velkým objemem dat díky jejich schopnosti zpracovat velké množství příznaků a vytvořit efektivní model založený na datech. Pokud jde o implementaci hlubokých neuronových sítí, pracuje se s paralelizovanými operacemi s maticemi a tenzory a výpočtem gradientů. Tuto funkcionalitu poskytuje několik rozhraní, které jsou přímo navržena pro vývoj hlubokých neuronových sítí jako například PyTorch a Keras. Tato rozhraní navíc poskytují předtrénované modely a další užitečnou funkcionalitu [20].

## 2.5 Konvoluční neuronové sítě

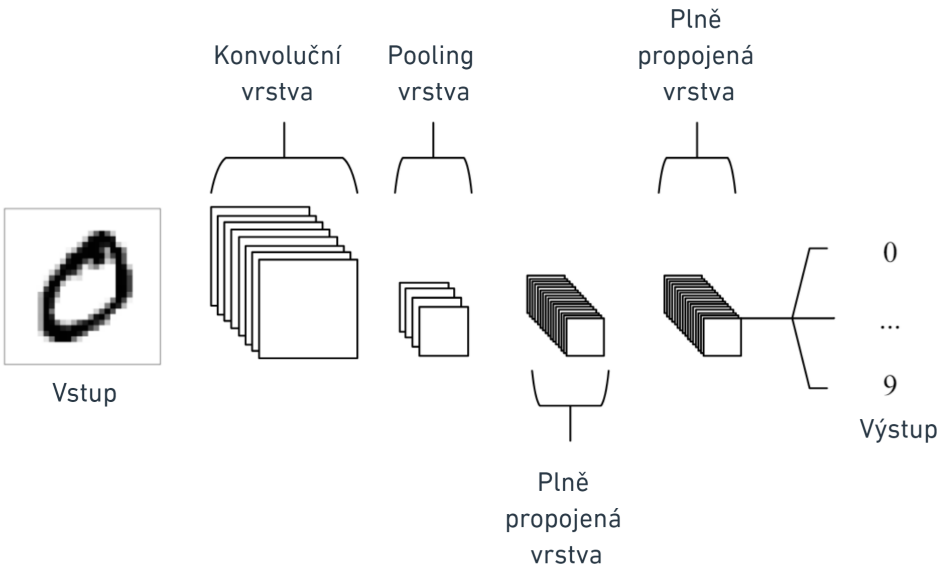
Konvoluční sítě (CNN) obsahují tři typy vrstev - konvoluční vrstvy, pooling vrstvy a plně propojené vrstvy. Klíčovou roli v CNN hrají konvoluční vrstvy. Tento typ vrstvy využívá proces zvaný konvoluce (viz. obrázek 2.5), který na vstupní data aplikuje různé filtry (tzv. jádra). Tyto jádra jsou předmětem učení konvolučních vrstev. Cílem vrstvy je exrahovat informace ze vstupních dat. Výstup každého jádra v konvoluční vrstvě jsou je 2D aktivační mapa. Konvoluční vrstva obsahuje několik jader. Vzdálenost o kterou se okno definující část vstupu zpracovanou pomocí konvoluce posune je určena pomocí parametru stride [16].




Obrázek 2.5: Aplikace jádra na část vstupu v konvoluční vrstvě. Centrální bod zpracovávané části je nahrazen váženým součtem celé části. Váhy jednotlivých bodů jsou určeny pomocí jádra. Převzato z [16]

.

Pooling vrstvy jsou v konvolučních neuronových sítích (CNN) umístěny za konvolučními vrstvami. Jejich hlavním účelem je redukovat velikost aktivačních map, čímž snižují výpočetní náročnost modelu. Pooling vrstva zpracovává aktivační mapu pomocí agregační funkce. Velikost částí, které jsou agregovány, je určena dimenzí jádra. Běžně se používá jádro o velikosti $2 \times 2$ s agregační funkcí MAX. Vrstva s takovou agregační funkcí se nazývá max-pooling. Pokud například použijeme tuto pooling vrstvu s parametrem stride $= 2$, velikost vstupní aktivační mapy je redukována na čtvrtinu její původní velikosti [16].

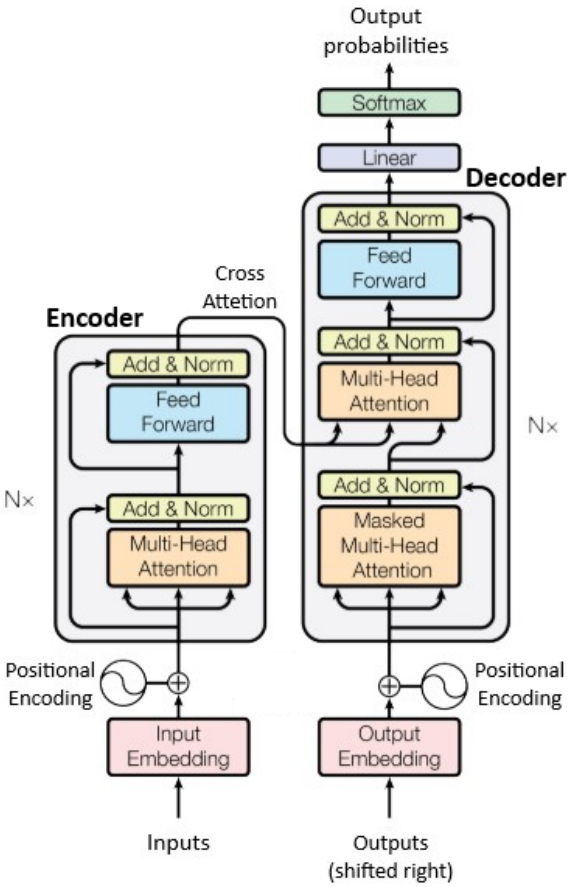



Obrázek 2.6: Architektura jednoduché konvoluční sítě s pěti vrstvami. Převzato z [16]
.

CNN se primárně využívají v úlohách pro zpracování obrazu [16], nicméně některé CNN modely jako například Extended Neural GPU, ByteNet a ConvS2S tvoří součást modelů pro řešení úloh přirozeného zpracování jazyka [25].

## 2.6 Transformery

Jedná se architekturu neuronové sítě skládající se ze dvou částí - enkodéru a dekodéru založenou na mechanismus attention zcela bez použití rekurentních a konvolučních vrstev. Úloha enkodéru je mapovat vstupní sekvence symbolů $(x_1, \ldots, x_n)$ z diskrétní množiny na reprezentaci v podobě sekvencí reálných čísel $\mathbf{z} = (z_1, \ldots, z_m)$. dekodér je schopen z reprezentace $\mathbf{z}$ postupně generovat sekvenci symbolů $(y_1, \ldots, y_n)$ z původní diskrétní množiny. Generování výstupní sekvence symbolů je autoregresevní - v každém kroku je na vstupu dekodéru symbol vygenerovaný v předešlém kroku [25].

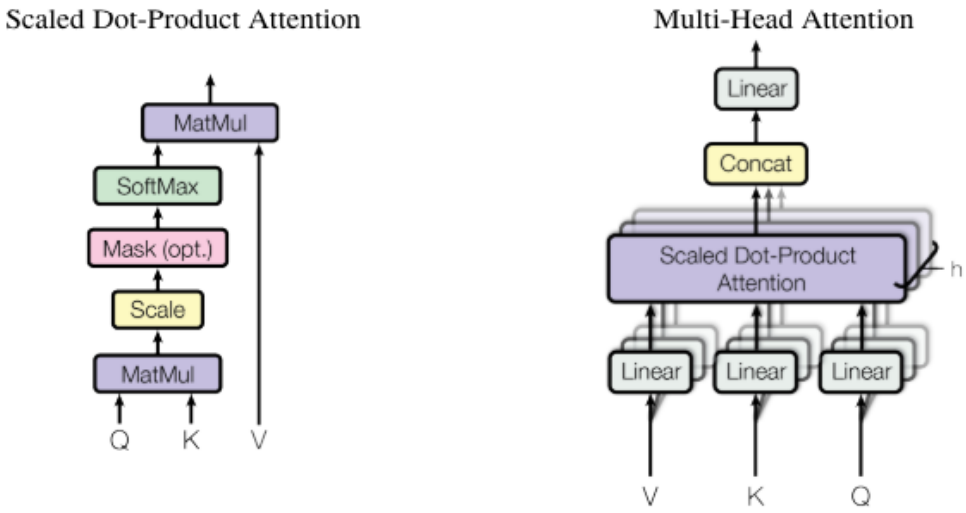



Obrázek 2.7: Architektura modelu transformeru typu enkodér-dekodér. Převzato z [25].

Transformer se řídí architekturou zachycenou na obrázku 2.7, která se skládá z attention mechanismu a plně propojených vrstev v částech enkodéru i dekodéru. Dekodérová část je navíc rozšířena o cross-attention [25].

### 2.6.1 Mechanismus attention

Mechanismus attention lze popsat jako mapovaní dotazu a páru klíč-hodnota na výstup, kde dotaz, klíč, hodnota i výstup jsou vektory. Výstupní vektor je vypočítán jako vážena suma hodnot. Váha každé hodnoty je spočtena pomocí funkce kompatibility dotazu a příslušného klíče [25].




Obrázek 2.8: Scaled Dot-Product Attention (vlevo) a Multi-Head Attention (vpravo). Převzato z [25].

V článku "Attention is all you need"[25] je použita implementace attention mechanismu zvaná "Scaled Dot-Product Attention", kterou lze vidět na obrázku 2.8. Vstup do Scaled Dot-Product Attention je tvořen dotazy a klíči o dimenzi $d_k$ a hodnotami o dimenzi $d_v$. Vypočítá se skalární součin dotazu se všemi klíči, který se podělí $\sqrt{d_k}$ a aplikuje se na něj softmax funkce. Tím postupem získáme váhy hodnot [25].

V praxi je tento výpočet proveden zároveň na několika dotazech, ty tvoří matici $Q$. Z klíčů a hodnot jsou taktéž sestaveny matice $K$ a $v$. Výpočet tedy lze zapsat jako:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Multi-Head Attention provede paralelně $h$ lineárních projekcí. Výstupy z těchto projekcí jsou zřetězeny a nakonec zpracovány pomocí finální projekce. Tento proces je zobrazen na obrázku 2.8 a matematicky lze zapsat jako [25]:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$
$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V),$$

kde:
$W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_k}$, a $W^O \in \mathbb{R}^{h_d \times d_{\text{model}}}$ [25].

### 2.6.2 Enkodér

Enkodér (viz obrázek 2.7 ) je složen z několika na sebe navazujících identických bloků. Každý takový blok je tvořena z multi-head self-attention mechanismu a plně propojené vrstvy. Tyto dvě vrstvy jsou doplněný reziduálním propojením. Celý blok je ukončen normalizační vrstvou [25].

### 2.6.3 Dekodér

Dekodér (viz obrázek 2.7 ) je tvořen z podobným bloků jako enkodér, ty jsou však rozšířeny o multi-head attention nad výstupem z enkodéru. V dekodéru je navíc self-attention vrstva doplněna o maskování, které zajišťuje aby attention mechanismus nebral ohled na nadcházející pozice [25].

# Kapitola 3

# Evoluční algoritmy

Evoluční algoritmus (EA) je nedeterministický prohledávací algoritmus inspirovaný přírodou, založených na Darwinově teorii přirozeného výběru a dalších teorií neodarwinismu. Hlavními zakladateli evolučních algoritmů byli John Holland, Ingo Rechenberg, Hans-Paul Schwefel a Lawrence Fogel. Holland navrhl genetické algoritmy a popsal je ve své knize z roku 1975. Myšlenku umělé evoluce navrhl již Alan Turing v roce 1948, při práci na konstrukci elektronického počítače ACE v UK, pod dohledem Sira Charlese Darwina, vnuka Charlese Darwina [12].

EA lze neformálně popsat jako algoritmus náhodného prohledávání prostoru, který pracuje s populací jedinců a snaží se svým chováním napodobit přírodní evoluci. Body ve prohledávaném prostoru reprezentují jednotlivá řešení, ze kterých je tvořena populace algoritmu. Kvalitu jedinců v populaci vzhledem ke zvolené úloze, kterou EA řeší, lze vyhodnotit pomocí fitness funkce. Další nezbytnou součástí EA jsou genetické operátory jako selekce, křížení nebo mutace, které se využívají pro tvorbu nové generace. Algoritmus je ukončen po dosažení předem stanovených cílů, jako např. dosažení stanovené hodnoty fitness funkce, dosažení určitého počtu řešení atd. [2].

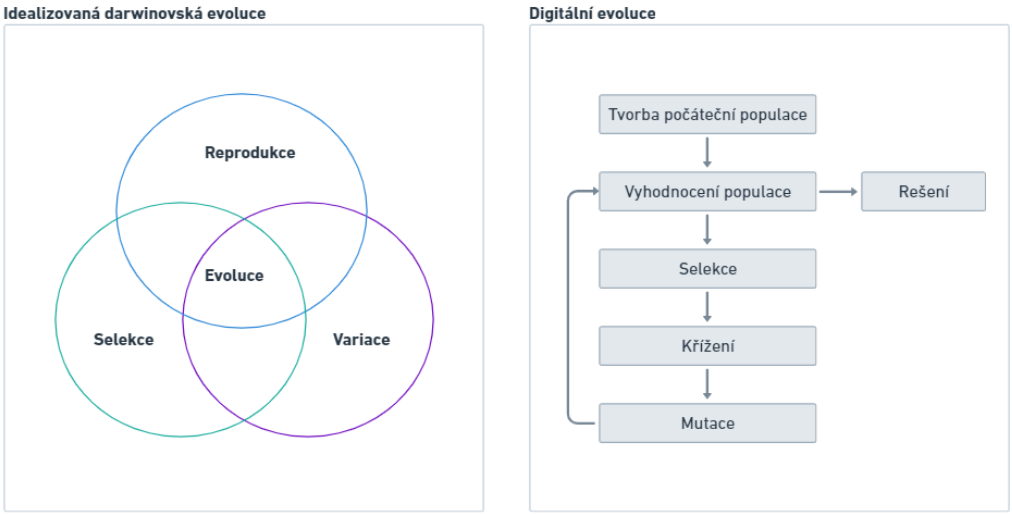
Společně s fuzzy systémy, neuronovými sítěmi, strojovým učením a pravděpodobnostním uvažováním patří EA do oboru zvaného Soft computing (SC). Předlohou pro SC je lidská mysl. Jeho hlavním rysem je využití tolerance k nepřesnostem a aproximacím k dosažení proveditelnosti úlohy, či snížení robustnosti a nákladů na řešení této úlohy. Vyjmenované disciplíny SC se často navzájem doplňují v řešení komplexních problémů

Základním konceptem EA je evoluce. Evoluce je dynamický mechanismus, který aplikuje selekci, variaci a replikaci na populaci jedinců. Toto je náhodný proces se snahou přibližovat ke stanovenému cíli [18].

- **Reprodukce** - Tvorba nové populace jedinců nebo změna jednotlivých jedinců ve stejné populaci [18].
- **Variace** - Snaha o diverzifikaci populace. Toho lze dosáhnout pomocí křížení, což je vytvoření nového jedince (potomka) kombinací částí rodičů vybraných z populace, a mutace, která představuje náhodnou změnu části jedince [18].
- **Selekce** - Výběr jedinců pro tvorbu nové populace na základě jejich kvality založený na Darwinově přirozeném výběru [18].

V kontextu evolučních algoritmů existuje také pojem digitální evoluce, která vychází ze stejných principů jako klasická evoluce. V digitální evoluci je populace jedinců měněna v

průběhu generací. Každá změna začíná selekcí z předchozí populace, vybraní jedinci jsou ohodnoceni vzhledem k požadovaným vlastnostem. Poté následuje reprodukce s určitým stupněm variace ve formě křížení nebo mutace [18].




Obrázek 3.1: Porovnání idealizované darwinovské evoluce a digitální teorie [18].

EA nachází uplatnění při řešení problémů, kde tradiční analytické, deterministické nebo stochastické algoritmy mají problém nalézt řešení. Tento fakt může být způsoben například omezeným výpočetním výkonem, komplexní funkcionalitou programu nebo daty s velkým počtem dimenzí. Řešení takových problémů by vyžadovalo překročení limitů dostupných výpočetních zdrojů. V takovém případě je vhodné použít EA, který může být schopen najít alespoň dostačující řešení. EA není vhodné používat pro jednoduché úlohy, kde lze snadno využít tradiční algoritmy [22].

## 3.1 Pojmy

### 3.1.1 Kandidátní řešení

Kandidátní řešení (fenotyp) je objekt, na kterém je možné uplatnit křížení a mutaci. Fenotyp můžeme volně označit jako program, který je možné chápat nejen ve smyslu počítačových jazyků, ale také například jako popis biologických procesů nebo matematickou funkci [22].

### 3.1.2 Reprezentace

Reprezentace, také zvaná genotyp nebo chromozom, je struktura, která zastupuje kandidátní řešení v EA. Může mít podobu řetězce, grafu, pole numerických hodnot atd. Každá reprezentace má své výhody a nevýhody, které navíc závisí na doméně řešeného problému. Volba reprezentace přímo ovlivňuje, s jakými částmi struktury lze manipulovat při křížení a mutaci. Na základě reprezentace jsou také definována pravidla, která určují korektní manipulaci s genotypem [22].

### 3.1.3 Populace

Populace je množina kandidátních řešení, která je tvořena dočasně v každém cyklu evoluce. Počáteční populace je vytvořena buď náhodně nebo jako podmnožina z množiny již známých řešení [22].

Velikost populace může být buď fixní nebo se může během evoluce dynamicky měnit. Pokud se velikost populace dynamický mění v průběhu evoluce je výhodné na začátku evoluce začínat s velkou populací a tu během evoluce postupně zmenšovat [22].

### 3.1.4 Generace

Jedna iterace EA, během které je vytvořena nová populace pomocí genetických operátorů, se nazývá generace. EA jsou často omezeny maximálním počtem generací, po kterém se se evoluce ukončí. Počet generací potřebný pro dosažená požadovaného cíle během evoluce je závislý na zvolené úloze, kterou EA řeší. Využívají se jak kombinace malé populace a velkého počtu generací, tak velké populace a malého počtu generací. Při volbě velikosti populace a maximálního počtu generací je také nutnou přihlédnout k výpočetní náročnosti vyhodnocení populace pomocí objektivní funkce [22].

### 3.1.5 Fitness a fitness funkce

Hodnota fitness vyjadřuje kvalitu jedince vzhledem k požadovaným vlastnostem. K výpočtu této hodnoty se používá fitness funkce zvaná také účelová funkce. Ohodnocení populace pomocí fitness funkce je výpočetně jednou z nejnáročnějších částí evoluce. Jedinci s lepší hodnotou fitness funkce mají větší pravděpodobnost být zvoleni jako rodiče pro tvorbu další generace [22].

### 3.1.6 Genetické operátory

Selekce je metoda výběru jedinců ze současné populace, ze kterých je tvořena následující populace. V digitální evoluci není počet rodičů, z nichž vznikne jeden potomek, omezen pouze na dva. Jev, kdy jsou v nové populaci zahrnuti i jedinci zvolení pomocí selekce, nazýváme elitismus. Mezi metody selekce patří například ranked selection, při které je populace seřazena podle hodnoty fitness a jedinci jsou vybíráni od nejlepšího. Další možností je tournament selection, kde mezi náhodně zvolenými jedinci z populace je uspořádán turnaj a výherce každého turnaje je zvolen pro další evoluci [22].

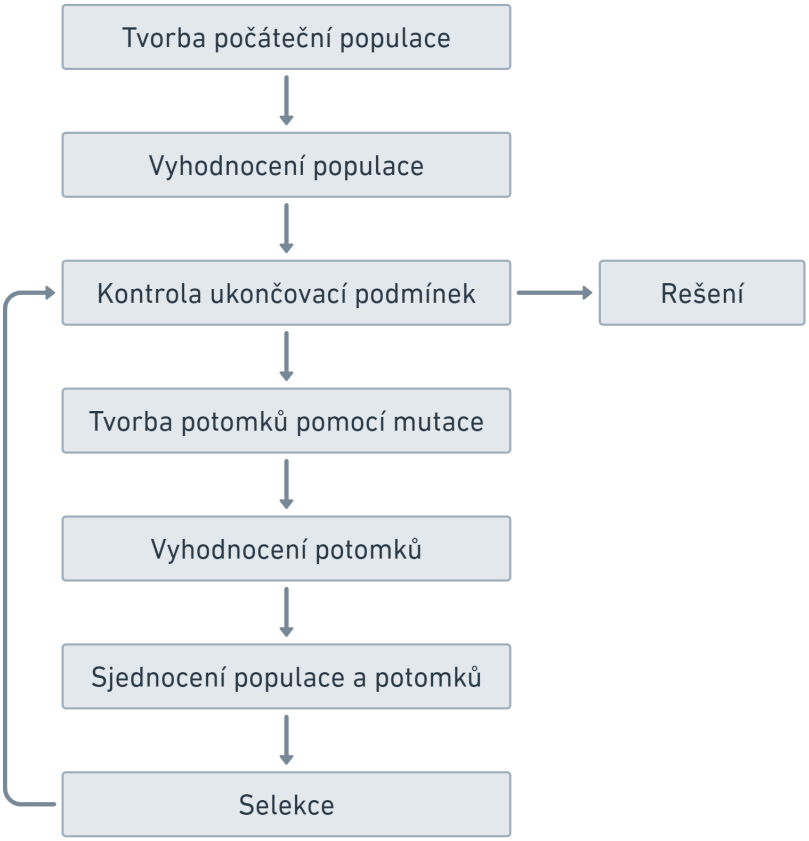
Dalšími genetickými operátory jsou křížení a mutace. Mutace má explorativní charakter, zatímco křížení exploitativní. Míra využití mutace a křížení se může v jednotlivých EA lišit. Tím vzniká celé spektrum algoritmů s různými vlastnostmi. Poměr mutace a křížení může být také měněn dynamicky za běhu evoluce. V některých případech se dá použít pouze jeden z těchto operátorů [22].

## 3.2 Evoluční programování

Jedná se o nejstarší formu EA. Používá fixní strukturu programu, přičemž předmětem evoluce jsou jednotlivé parametry. Jedním ze základních charakteristik evolučního programování (EP) je, že jako genetický operátor nepoužívá křížení, ale využívá pouze mutaci. EP pracuje přímo s fenotypem, a proto nevyžaduje žádnou speciální reprezentaci řešení ani

mapovací funkci pro převod mezi fenotypem a genotypem. Selekce v EP je často založena na náhodnosti [22].

Na obrázku 3.2 je zobrazeno schéma algoritmu EP. Nejprve je vygenerována náhodná populace - náhodná inicializace populace je základní a nejednodušší způsob inicializace, nicméně existují i jiné, složitější způsoby, jako například využití předem známých jedinců. Následuje ohodnocení populace pomocí fitness funkce. Pokud je řešená úloha optimalizačního charakteru, často se místo fitness funkce používá pojem účelová funkce. Poté je z populace vytvořena množina potomků pomocí genetického operátoru mutace. Tato množina je sjednocena s původní populací a nakonec je pomocí selekce na základě fitness funkce zredukována na požadovanou velikost. Takto vznikne nová populace. Proces se opakuje, dokud není naplněn některý z cílů algoritmu, jako je dosažení požadované hodnoty objektivní funkce nebo určitého počtu evolucí [22].

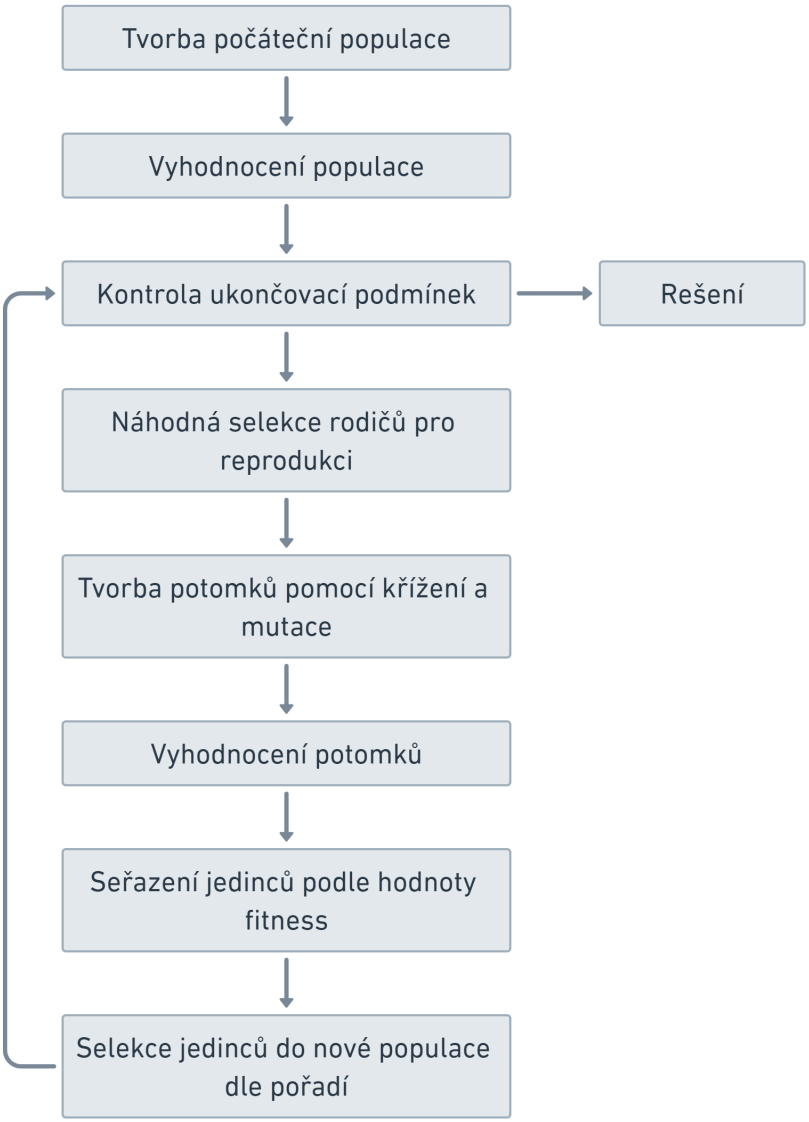



Obrázek 3.2: Schéma základního algoritmu evolučního programování.

## 3.3 Evoluční strategie

První evoluční strategií (ES) je tzv. (1+1)-ES. Ta používá pro tvorbu populace pouze jednoho rodiče a jednoho potomka. Nad rodičem jsou provedeny malé náhodné změny všech proměnných zároveň. Pokud má vzniklý potom lepší hodnotu fitness funkce než rodič,

stane se z něj nový rodič, jinak je jako rodič zachován původní jedinec. Tento algoritmus byl popsán jako minimální koncept pro imitaci organické evoluce. Základní forma ES využívá pouze mutaci, později bylo zapojeno i křížení [2].




Obrázek 3.3: Schéma základního algoritmu evoluční strategie.

ES se od ostatních EA vyznačuje tím, že selekce jedinců pro reprodukci je uniformní (není ovlivněna hodnotou fitness funkce). Selekce jedinců pro vytvoření nové populace je uspořádaná a deterministická. Variační operátory (mutace a křížení) lze kontrolovat pomocí parametrů. Jedinci se skládají z kandidátního řešení a řídících parametrů [19].

Pokud stanovíme, že populace v čase $t \geq 0$ se značí $P^t$. Jedinec $p \in P^t$ je tvořen dvojicí $p = (x, \Psi)$, kde $x \in X$ je prvkem z prohledávaného prostoru $X$ a $\Psi$ je konečná množina řídících parametrů. Fitness funkce $f : X \to \mathbb{R}$ mapuje prvek z množiny X na hodnotu z $\mathbb{R}$. Pak lze různé varianty ES formálně zapsat jako:

$$(\mu/\rho, \lambda, k) - ES,$$

kde $\mu$ značí počet rodičů. $\lambda$ počet potomků, $\rho$ počet rodičů, kteří se podílí na jedné tvorbě potomka a $k$ maximální počet generací, které jedinec může přežít. Zároveň platí, že $\mu, \rho, \lambda \in \mathbb{N}$, $k \in \mathbb{N} \cup \{\infty\}$ a zároveň $\rho \leq \mu$. Pokud $k < \infty$, poté $\lambda > \mu$ [19].

Do roku 1995 nebyl parametr $k$ součástí jedince a používalo se pouze $k = 1$ (strategie "čárka") a $k = \infty$ (strategie "plus"). Tento zápis $(\mu + \lambda)$-ES tedy odpovídá $(\mu/2, \infty, \lambda)$-ES [19].

| **Algoritmus** | **Rodiče** | **Reprodukce** | **Potomci** | **Životnost** |
|---|---|---|---|---|
| $(1+1)$ | $\mu = 1$ | $\rho = 1$ | $\lambda = 1$ | $k = \infty$ |
| $(1+\lambda)$ | $\mu = 1$ | $\rho = 1$ | $\lambda > 1$ | $k = \infty$ |
| $(1, \lambda)$ | $\mu = 1$ | $\rho = 1$ | $\lambda > 2$ | $k = 1$ |
| $(\mu+1)$ | $\mu \geq 2$ | $\rho = 2$ | $\lambda = 1$ | $k = \infty$ |
| $(\mu+\lambda)$ | $\mu \geq 2$ | $\rho = 2$ | $\lambda > 2$ | $k = \infty$ |
| $(\mu, \lambda)$ | $\mu \geq 2$ | $\rho = 2$ | $\lambda \geq \mu$ | $k = 1$ |

Tabulka 3.1: Tabulka zkrácených notací parametrů strategií typu "čárka"a "plus". Převzato z [19].

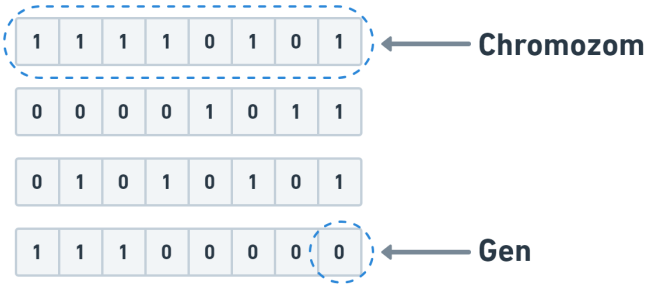
ES se využívá zejména k řešení úloh pro optimalizaci spojitých parametrů. Každý parametr v takovéto úloze je definován typem a intervalem hodnot, kterých může nabývat [22].

## 3.4 Genetické algoritmy

Genetický algoritmus (GA) je nejvíce rozšířenou formou EA. Využívá se pro řešení optimalizačních a konfiguračních problémů obsahujících velké množství parametrů. Tyto parametry se mohou navzájem ovlivňovat. Tato skutečnost společně s jejich velkým množstvím činí tento typ úloh nevhodný pro řešení pomocí tradičních metod[22].

Genotyp v GA má formu lineárně uspořádaného řetězce fixní délky. Do jednotlivých pozic v tomto řetězci (nebo-li genů) jsou kódovány vlastnosti fenotypu. To znamená. že pokud máme dva chromozomy stejného typu, potom i-tý gen bude reprezentovat stejnou vlastnost v obou chromozomech. Stavy, kterých může gen nabývat, se označují jako alely [10].

Způsob, jakým jsou jedinci kódováni, má velký vliv na použití GA pro řešení konkrétní úlohy. Binární kódování patří mezi jedny z nejstarších a nejpoužívanějších. V tomto kódování jsou individua reprezentována chromozomem určeným řetězcem fixní délky, jehož prvky mohou nabývat buď hodnoty 0 nebo 1 [10].




Obrázek 3.4: Příklad reprezentace 4 chromozomů genetického algoritmu v binárním kódování délky 8.

Selekce v genetických algoritmech se snaží imitovat přirozený výběr, přičemž zdatnější jedinci mají větší šanci přežít a zplodit potomky. Nejrozšířenější implementací přirozeného výběru je tzv. ruletový výběr. Na rozdíl od klasické rulety, kde má každé číslo stejnou pravděpodobnost být vybráno, v tomto případě jsou upřednostňováni jedinci s lepší hodnotou fitness funkce, kterým je na pomyslném ruletovém kole přiřazena větší výseč. Slabší jedinci mají menší výseč, což je nevyřazuje přímo, ale snižuje pravděpodobnost jejich výběru[10].

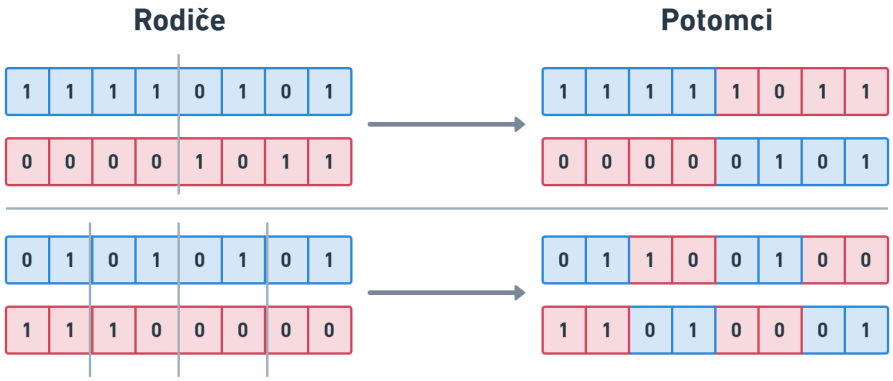
Existuje několik způsobů, jak takovéto ruletové kolo zkonstruovat. Jedním z nich je tzv. fitness-proportionate selection, kde má každé individuum přiřazenou pravděpodobnost, jejíž velikost je přímo úměrná ohodnocení tohoto individua [10].

| | Chromozom | Hodnota fitness | % z celkové hodnocení | Kumulovaná fitness |
|---|---|---|---|---|
| 1. | 1 1 1 1 0 1 0 1 | 6 | 0.3750 | 0.3750 |
| 2. | 0 0 0 0 1 0 1 1 | 3 | 0.1875 | 0.5625 |
| 3. | 0 1 0 1 0 1 0 1 | 4 | 0.2500 | 0.8125 |
| 4. | 1 1 1 0 0 0 0 0 | 3 | 0.1875 | 1.0000 |

Obrázek 3.5: Příklad ohodnocení jedinců z populace pro vytvoření ruletového kola.

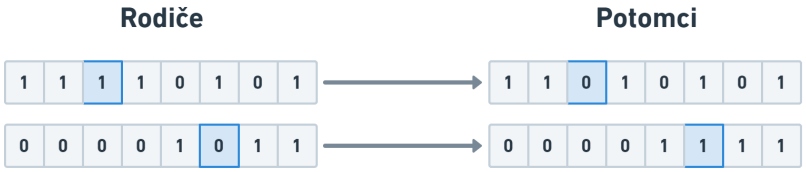
K vytvoření rulety stačí sečíst ohodnocení všech členů populace a poté každému přiřadit odpovídající kruhovou výseč. K výběru jedince pomocí této metody je výhodné použít kumulovanou fitness [10].

K vytvoření potomků jsou používány genetické operátory křížení a mutace. Existuje celá škála různý technik křížení, avšak všechny mají společnou tu vlastnost, že jde vždy o vzájemnou výměnu částí chromozomů [10]. Na obrázku 3.6 lze vidět jednobodové křížení, což je nejjednodušší varianta křížení používaném v GA.




Obrázek 3.6: Demonstrace tvorby potomků pomocí jednobodového křížení (nahoře) a více bodového křížení (dole).

Operátor mutace většinou velmi jednoduchým způsobem a s relativně malou pravděpodobností mění náhodně hodnotu jednotlivých genů. Konkrétně v případě binárního kódování to znamená, že vybraný gen v daném chromozomu změní svoji hodnotu z jedničky na nulu a naopak [10].



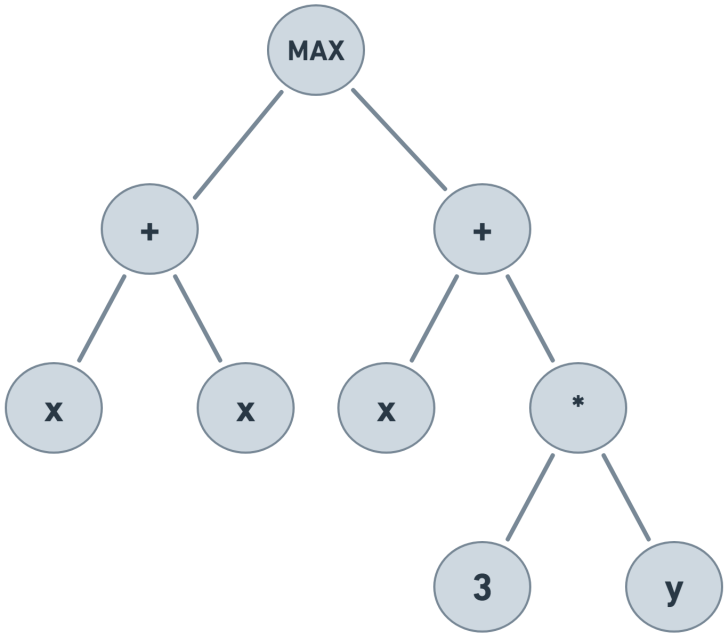

Obrázek 3.7: Vytvoření potomků pomocí genetického operátoru mutace.

## 3.5 Genetické programování

Genetické programování (GP) je jednou z metod evolučního výpočtu. Podstatou GP je automatická evoluce programu. Jednou z hlavních obtíží při evoluci počítačových programů je skutečnost, že programy jsou silně omezeny a musí dodržovat konkrétní gramatiku, aby mohly být zkompilovány. To znamená, že každý program musí mít správnou syntaktickou strukturu a musí se řídit pravidly programovacího jazyka, ve kterém je napsán. Při genetickém programování, kde dochází k náhodným mutacím a rekombinacím kódu, je výzvou udržet validitu a funkčnost programů po těchto změnách. Nevalidní nebo nesprávně strukturovaný kód nemůže být zkompilován nebo spuštěn, což ztěžuje nalezení efektivních a pracovních řešení prostřednictvím evolučních procesů [12].

### 3.5.1 Reprezentace

Programy v GP se obvykle reprezentují pomocí syntaktického stromu namísto řádků kódu. Ve složitějších případech je dokonce možné program reprezentovat jako několik stromů. Stromovou reprezentaci programu $\max(x + x, x + 3y)$ lze vidět na obrázku 3.8. Proměnné a konstanty programu $(x, y, 3)$ v GP nazýváme terminály, a ty jsou ve stromu v listových uzlech. Aritmetické operátory $(+, *, max)$ se nazývají funkce (neterminály) a jsou reprezentovány vnitřními uzly. Množina terminálů a funkcí společně tvoří primitivní množinu systému GP. V literatuře je běžné reprezentovat výrazy v prefixové podobě, která lépe zachycuje vztahy mezi části výrazy a odpovídajícími podstromy. Dříve zmíněný výraz lze zapsat v prefixové podobě jako $\max(+\ x\ x)(+\ x\ (*\ 3\ y))$ [17].



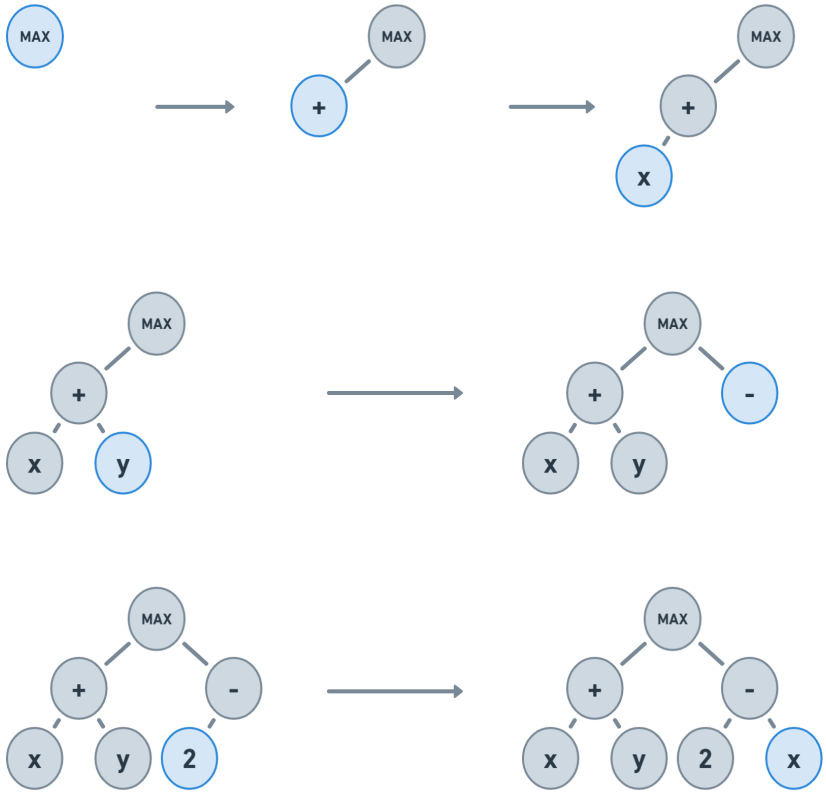

Obrázek 3.8: GP syntaktický strom reprezentující program $\max(x + x, x + 3y)$, převzato z [17].

### 3.5.2 Inicializace populace

Stejně jako v ostatních evolučních algoritmech, nejjednodušším způsobem, jak inicializovat populaci, je náhodně. Mezi další často používané inicializační metody patří tzv. full, growth, a jejich kombinace, ramped half-and-half [17].

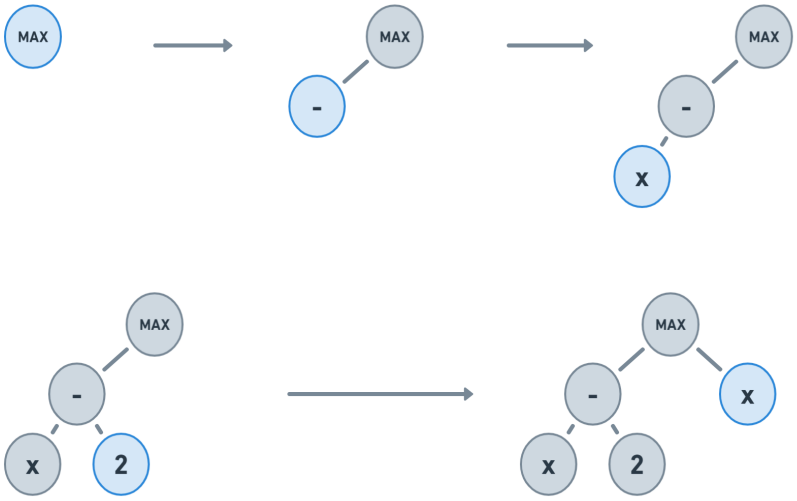
V případě metod full a growth jsou jedinci generováni tak, aby nepřekročili určitou maximální hloubku. Hloubka uzlu je počet hran, které je třeba projít po cestě do kořenového uzlu (ten má hloubku 0). Hloubka stromu je rovna maximální hloubce jeho uzlů. Metoda full generuje plně saturované stromy - vybírají se pouze uzly z množiny funkcí, dokud není dosaženo maximální velikosti stromu, poté je strom dokončen pomocí uzlů z množiny terminálů tak, aby byl syntakticky korektní. Obrázek 3.9 zobrazuje proces tvorby syntaktického stromu pomocí metody full. Je nutné podotknout, že stromy vytvořené této pomocí me-

tody mají stejnou velikost (počet uzlů) a tvar, pouze pokud množina funkcí obsahuje pouze funkce se stejným počtem operandů [17].




Obrázek 3.9: Tvorba syntaktického stromu pomocí metody full s maximální hloubkou 2, převzato z [17].

Metoda growth umožňuje vytvářet stromy s větší diverzitou velikostí a tvarů. uzlu jsou vybírány z celé primitivní množiny dokud se nedosáhne maximální hloubky, poté se vybírá pouze z množiny terminálů[17]. Proces tvorby syntaktického stromu pomocí metody growth je zobrazen na obrázku 3.9.




Obrázek 3.10: Tvorba syntaktického stromu pomocí metody growth s maximální hloubkou 2, převzato z [17].

Obě výše zmíněné metody samy o sobě dokáží generovat pouze úzkou škálu stromů. Proto Koza v roce 1992 navrhl kombinaci těchto metod, zvanou ramped half-and-half, při níž je polovina populace vytvořena pomocí metody full a druhá polovina pomocí metody growth. U metody growth je přitom hloubka postupně nastavena na počáteční úroveň 0 a následně zvětšována o jedničku až do požadované maximální hloubky [17].

### 3.5.3 Selekce

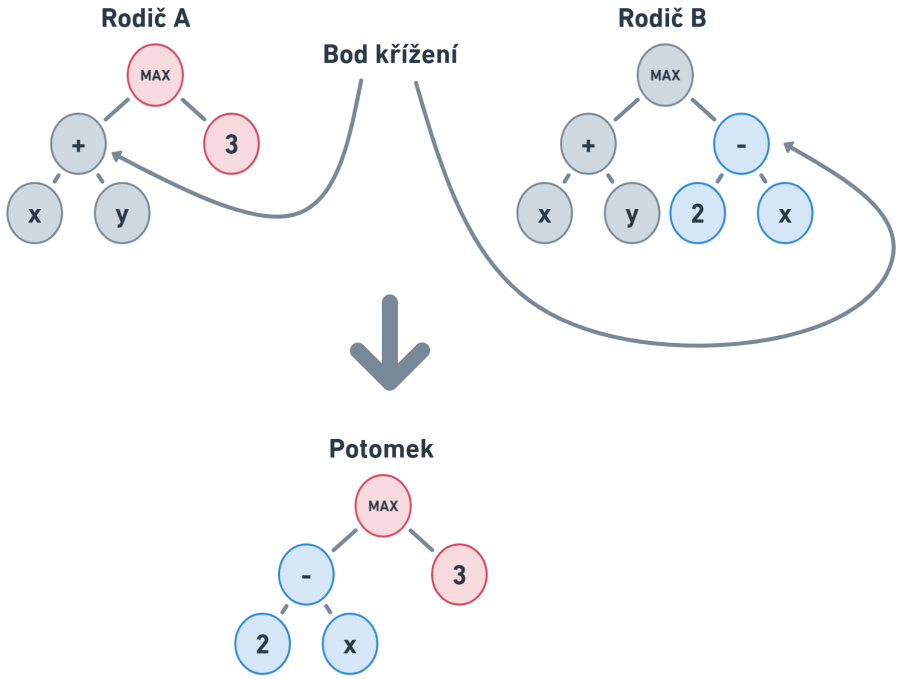
Stejně jako ve většině evolučních algoritmů, i v genetickém programování jsou genetické operátory aplikovány na jedince, kteří jsou pravděpodobnostně vybráni z populace na základě jejich fitness ohodnocení. To znamená, že jedinci s lepším hodnocením mají větší pravděpodobnost, že zplodí potomky. Nejběžnějším způsobem, jak vybrat jedince pro reprodukci, je metoda tournament selection [17].

Při tournament selection je nejprve náhodně vybrán určitý počet jedinců z populace. Poté se porovná jejich fitness a nejlepší z nich je vybrán pro reprodukci. Při výběru rodičů pro křížení je třeba celý proces zopakovat dvakrát, aby byli získáni dva rodiče. Díky tomu, že se porovnává pouze pořadí fitness hodnot a ne velikost jejich rozdílu, zůstává selekční tlak na populaci konstantní. To zabraňuje, aby jeden výjimečný jedinec zahltil populaci svými potomky a tím předešel ztrátě diverzity populace. Další výhodou této selekční metody je částečná náhodnost, způsobená náhodným výběrem jedinců do turnaje, což dává možnost i průměrně hodnoceným řešením zplodit potomky [17].

### 3.5.4 Křížení a mutace

V této části se GP výrazně liší od ostatních EA. Nejběžnější metodou křížení používanou v GP je křížení podstromů (zobrazeno na obrázku 3.11). V tomto případě máme dva rodiče,

u nichž jsou náhodně vybrány dva uzly jako body křížení. Potomek vznikne záměnou podstromu s kořenem v bodě křížení prvního rodiče s podstromem s kořenem v bodě křížení druhého rodiče. Typicky jsou před křížením vytvořeny kopie rodičů, se kterými se poté pracuje, aby nedošlo k jejich znehodnocení, pokud by byli vybráni pro křížení vícekrát. Z jednoho takového křížení je možné získat i dva potomky, ale běžně se vrací pouze jeden [17].




Obrázek 3.11: Příklad křížení podstromů v GP, převzato z [17].

Běžně používanou formou mutace v GP je mutace náhodného podstromu, kdy se náhodně vybere uzel jako výchozí bod mutace a podstrom s kořenem v tomto uzlu je nahrazen náhodně vygenerovaným stromem. Tento typ mutace lze implementovat jako křížení podstromu mezi jedním rodičem z populace a náhodně vygenerovaným stromem. Další formou mutace je tzv. jednobodová mutace, při které je náhodně změněna hodnota vybraného uzlu. Při této mutaci je třeba pohlídat aby nová hodnota v uzlu byla stejného typu - terminál/funkce a měla stejný počet operandů. Jednobodovou mutaci lze při průchodu stromem aplikovat na s určitou pravděpodobností na každý uzel [17].

Při tvorbě nové generace jsou operátory mutace a křížení v GP typicky exkluzivní, a každá z těchto operací je na jedince aplikována s určitou pravděpodobností. Pravděpodobnost křížení se typicky pohybuje kolem 90%, zatímco pravděpodobnost mutace se pohybuje v hodnotách kolem 1%. Pokud je součet pravděpodobností mutace a křížení menší než 1, použije se operátor reprodukce s pravděpodobností $1-p$. Reprodukce jednoduše zkopíruje jedince vybraného pomocí selekce do nové populace [17].

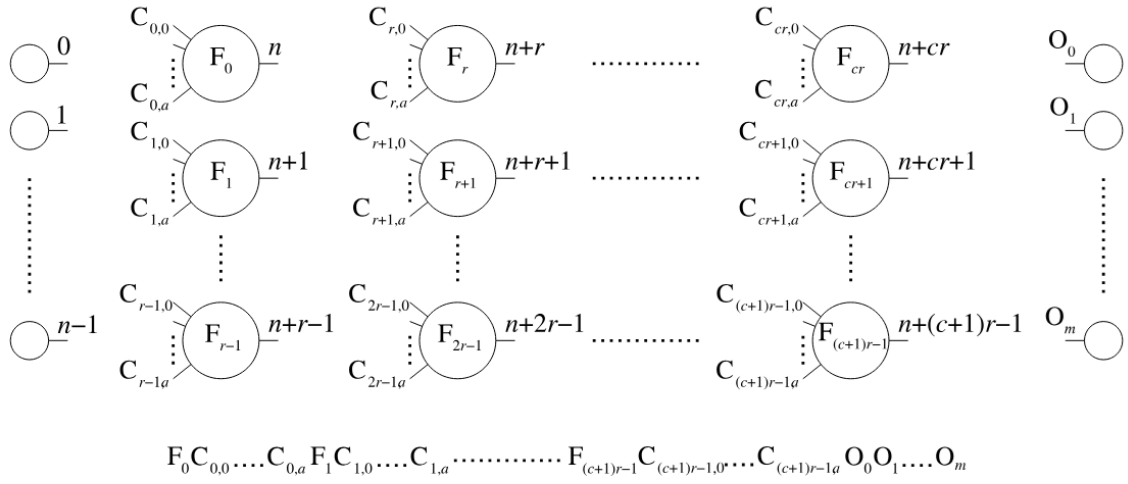
Podrobnější výčet různých implementací mutace a křížení lze nalézt v [17].

## 3.6 Kartézské genetické programování

V kartézském genetickém programování (CGP) je program reprezentován formou orientovaného acyklického grafu. Tyto grafy jsou tvořeny dvourozměrným polem výpočetních uzlů. Toto pole je zobrazeno na obrázku 3.12. Genotyp se skládá z genů, reprezentovaných jako celá čísla, které specifikují, odkud výpočetní uzel bere vstupní data, jakou operaci provádí a kde je možné získat výstupní data. Při dekódování genotypu na fenotyp mohou být některé uzly ignorovány, což se stane, pokud jejich výstup není použit v celkovém výpočtu řešení. Takto vynechané uzly nazýváme "nekódující". Genotyp v CGP má fixní délku; nicméně, velikost fenotypu (počet výpočetních uzlů) je proměnlivá a může kolísat od nuly do počtu uzlů definovaných v genotypu [11].

Funkce, které mohou výpočetní uzly provádět, jsou definovány uživatelem a jsou zapsány do look-up tabulky. Každý uzel reprezentuje jednu z těchto funkcí a je zakódován pomocí několika genů. Funkční gen určuje adresu výpočetní funkce v look-up tabulce. Další geny, nazývané "propojovací", kódují adresu vstupních dat pro uzel. Vstupy uzlů jsou dopředné, což znamená, že pocházejí z výstupů uzlů v předchozích sloupcích nebo přímo ze vstupu programu, který se někdy nazývá terminál. Počet vstupů uzlu závisí na maximální aritě funkce v look-up tabulce [11].

Parametry, jako jsou počet sloupců a řádků grafu v CGP, jsou definovány uživatelem společně s parametrem "Levels-back". Tento parametr určuje, ze kterých sloupců mohou uzly brát své vstupy. Nejmenší možná hodnota tohoto parametru je 1, kdy mohou uzly brát vstupy pouze z prvního sloupce vlevo. Pokud je hodnota tohoto parametru rovna počtu sloupců, uzly mohou brát vstupy ze všech sloupců na levo od nich. Různé kombinace těchto parametrů umožňují uživateli ovlivňovat celkový charakter grafů, se kterými CGP pracuje během evoluce [11].




Obrázek 3.12: Na obrázku je zobrazeno pole uzlů reprezentujících výpočetní funkce z look-up tabulky. Pole má $n_c$ sloupců a $n_r$ řádků. Počet vstupů programu je roven $n_i$, počet výstupů programu je roven $n_o$. Každý uzel má počet vstupů roven maximální aritě funkce $a$. Každá vstup má unikátní označení. Převzato z [11].

# Kapitola 4

# Filtrace obrazu

Zpracování obrazu je klíčovou součástí současných digitálních technologií, která se dotýká mnoha aspektů našeho každodenního života a vědecké práce. Zpracování obrazu se stalo nezbytným nástrojem pro rozšiřování našich možností poznání a interakce se světem. Na rozvoji této disciplíny se podílí nejen akademická komunita, ale i veřejnost, a to zejména díky cenově dostupným počítačům a rozšíření internetu, což zvyšuje přístup k informacím. Většina těchto informací je určena pro vizuální spotřebu v podobě textu, grafiky a obrázků [5].

## 4.1 Problematika šumu obrazu

Šum je nežádoucí složka vyskytující se v obrazech z mnoha důvodů. Gaussovský šum je součástí téměř každého signálu. Například dobře známý bílý šum na slabé televizní stanici je dobře modelován jako gaussovský. Zrnitý šum ve fotografických filmech je někdy modelován jako gaussovský a někdy jako Poissonův. Mnoho obrazů je poškozeno šumem "sůl a pepř", jako by někdo posypal obraz černými a bílými tečkami. Další šumy zahrnují kvantizační šum a skvrny na obrázku v koherentních světelných situacích [5].

Nechť $f(x)$ označuje obraz. Rozložíme obraz na požadovanou složku, $g(x)$, a šumovou složku, $q(x)$. Nejběžnější rozklad je aditivní:

$$f(x) = g(x) + q(x)$$

Například gaussovský šum je obvykle považován za aditivní složku. Druhý nejčastější rozklad je multiplikativní:

$$f(x) = g(x)q(x)$$

Příkladem šumu, který je často modelován jako multiplikativní, jsou skvrny na obrázcích [5].

Multiplikativní model lze převést na aditivní model pomocí logaritmů a aditivní model na multiplikativní pomocí umocňování:

$$log(f) = log(gq) = log(g) + log(q)$$

$$e^f = e^{g+q} = e^g e^q$$

Obě tyto formy se používají i přesto, že lze jeden převést na druhý a to proto, že hledáme jednoduché modely, které správně popisují chování systému. Existují i případy, kdy se žádná z těchto forem nehodí pro popsání daného šumu, například Poissonův šum a šum "sůl a pepř"[5].

### 4.1.1 Gaussovský šum

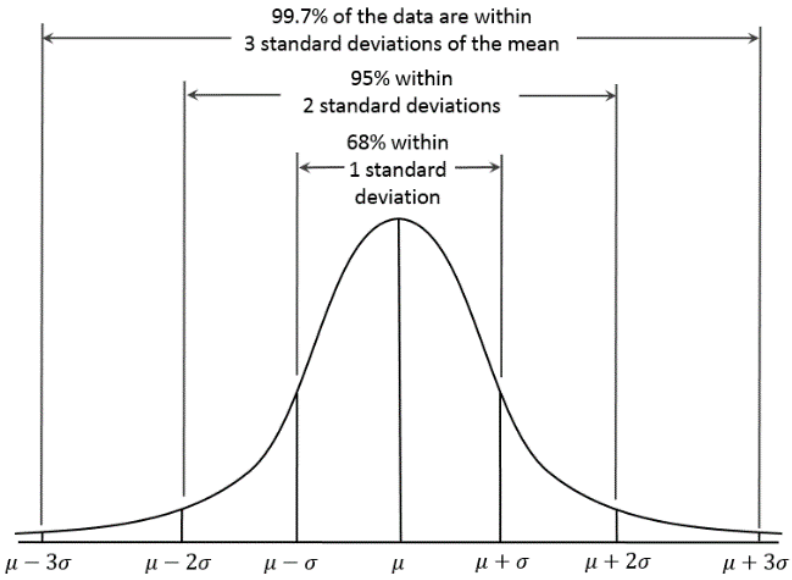
Nejčastěji se vyskytujícím šumem je pravděpodobně aditivní gaussovský šum. Je široce používán k modelování tepelného šumu a za často rozumných podmínek je omezujícím chováním jiných druhů šumů, například šumu při počítání fotonů a šumu zrnitosti filmu [5].

Obrázek 4.1: Porovnání originálního obrázku (vlevo) a obrázku s Gaussovským šumem se střední hodnotou 0 a směrodatnou odchylkou 30 (vpravo)[5].

Hustota rozložení jednorozměrného gaussovského šumu, $q$, s průměrem $\mu$ a rozptylem $\sigma^2$ je:

$$p_q(x) = \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

pro $-\infty < x < +\infty$. Značíme $x \sim \mathcal{N}(\mu, \sigma)$, což znamená, že náhodná proměnná $x$ je generována z gaussovského rozložení se střední hodnotou $\mu$ a rozptylem $\sigma^2$ [5].



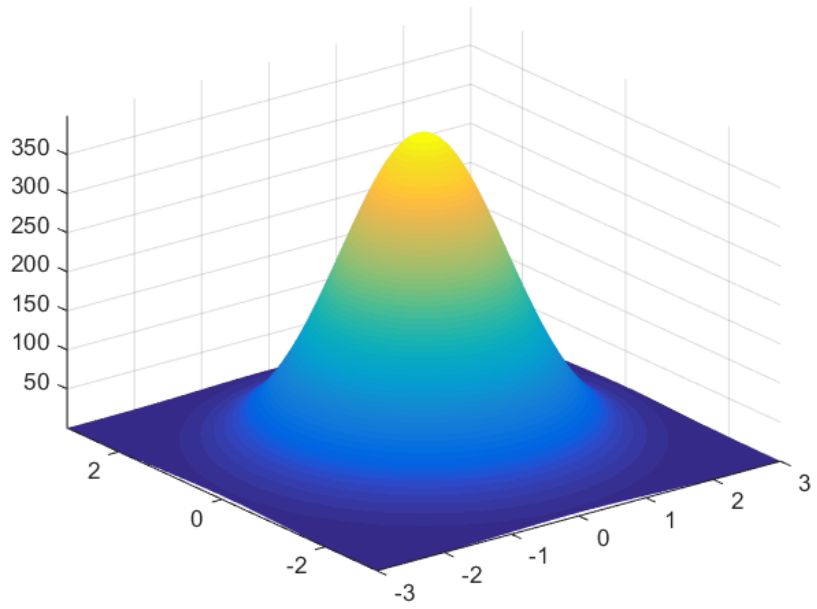

Obrázek 4.2: Příklad grafu jednorozměrného Gaussova rozložení. Převzato z [14].

V praxi je však rozsah hodnot gaussovského šumu omezen na přibližně $3\sigma$. Gaussovo rozložení je užitečným a přesným modelem pro mnoho procesů. Pokud je to nutné, hodnoty šumu mohou být ořezány tak, aby bylo $f > 0$ [5].

V případě kdy $a$ je náhodný vektor, hustota rozložení více-rozměrného gaussovského šumu je rovna:

$$p_a(a) = (2\pi)^{-\frac{n}{2}} |\Sigma|^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(a-\mu)^T \Sigma^{-1}(a-\mu)\right),$$

kde $\mu = E[a]$ je vektor středních hodnot a $\Sigma = E[(a-\mu)(a-\mu)^T]$ je kovarianční matice. Značíme $a \sim \mathcal{N}(\mu, \Sigma)$ [5].




Obrázek 4.3: Příklad grafu vícerozměrného Gaussova rozložení. Převzato z [6].

Gaussovský šum je možné odstranit pomocí lineárních filtrů. Jedním z takových filtrů je například gaussovský filtr. Tento filtr je založen na váženém průměru zpracovávané části obrázků, kdy je nejvíce zvýhodněn středových pixel.[5]

### 4.1.2 Impulsní šum

Tento termín popisuje celou škálu procesů, které vedou ke stejnému poškození původního obrazu: malé množství velmi zašuměných pixelů. Efekt je podobný, jako kdybychom posypali obraz černými a bílými tečkami - solí a pepřem [5].

Jedním z příkladů, kde vzniká šum soli a pepře, je přenos obrázků přes zašuměné digitální spoje. Nechť každý pixel je kvantizován na $B$ bitů obvyklým způsobem. Hodnota pixelu může být zapsána jako $X = \sum_{i=0}^{B-1} b_i 2^i$. Předpokládejme, že kanál je binárně symetrický s pravděpodobností překřížení $\epsilon$. Pak je každý bit překlopen s pravděpodobností $\epsilon$. Nazvěme přijatou hodnotu Y. Pak, předpokládáme-li, že překlopení bitů jsou nezávislá,

$$\Pr[|X - Y| = 2^i] = \epsilon(1-\epsilon)^{B-1}$$

pro $i = 0, 1, ..., B-1$. Dle [5] uvedeme základní charakteristiky šumu: Hodnota střední kvadratické chyby (MSE) nejvýznamnějšího bitu je $\epsilon 4^{B-1}$. Hodnota MSE všech ostatních bitů dohromady je $\epsilon \left(\frac{4^{B-1}-1}{3}\right)$. MSE nejvýznamnějšího bitu je přibližně třikrát větší než MSE všech ostatních bitů. Tedy pixely, kterým se při přenosu změní hodnota nejvýznamnějšího bitu, budou pravděpodobně černé nebo bílé.

Jednoduchý model zašumění obrázku lze zkonstruovat následovně: $f(x, y)$ je originální obrázek a $q(x, y)$ obrázek po zašumění.

$$\Pr[q = f] = 1 - \alpha,$$

$$\Pr[q = \text{MAX}] = \frac{\alpha}{2}a$$
$$\Pr[q = \text{MIN}] = \frac{\alpha}{2},$$

kde MAX a MIN jsou maximální a minimální hodnoty pixelu. Myšlenka tohoto modelu je taková, že hodnota pixelu v obrázku při aplikování šumu se nezmění s pravděpodobností $1 - \alpha$, a s pravděpodobností $\frac{\alpha}{2}$ se změní na hodnotu MAX nebo MIN [5].

Obrázek 4.4: Porovnání originálního obrázku (vpravo) a obrázku s 15% šumem sůl a pepř (vlevo) [5].

Tradičně se impulsní šum odstraňuje pomocí mediánového filtru (jedná se o nejvíce populární nelineární filtr), který na výstupu vrací medián zpracovávané části obrázku. Standardní mediánový filtr nedosahuje dobrých výsledků na obrázcích, které obsahují impulsní šum o vysoké intenzitě. Jednoduchý mediánový filtr který pracuje nad částí obrázku o velikosti $3 \times 3$ nebo $5 \times 5$ je vhodný pouze pro obrázky s intenzitou šumu menší než $10 - 20\%$ [21].

## 4.2 Genetické programování pro evoluční návrh obrazových filtrů

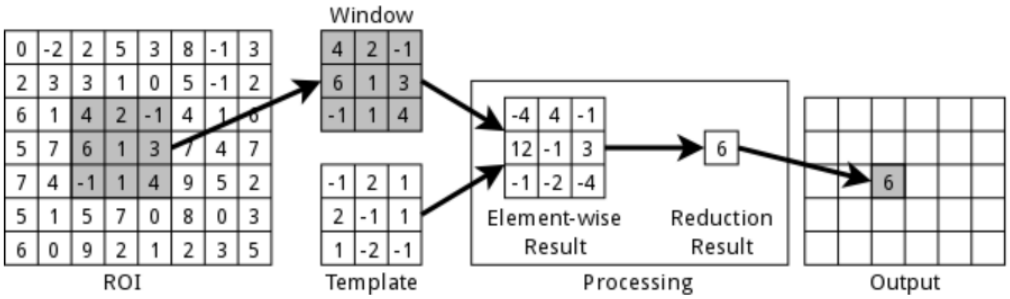
V oblasti zpracování obrazu je předzpracování, zahrnující filtraci, detekci hran, úpravu histogramu, jasu a kontrastu, zásadním prvním krokem. Toto předzpracování výrazně ovlivňuje následující fáze jako segmentaci a rozpoznávání obrazů. Většinou se využívají standardní filtry, které jsou manuálně upravovány pro konkrétní aplikaci, přičemž se optimalizují jejich koeficienty, struktura a další parametry.

Tradičně byly pro zpracování obrazu preferovány lineární filtry kvůli jejich matematickému základu. Avšak v některých případech poskytují nelineární filtry lepší výsledky, zejména v zachování hran a potlačení šumu. Pro návrh nelineárních filtrů se v poslední

době využívají evoluční techniky, jako je kartézské genetické programování (CGP). CGP dokázalo navrhovat filtry obrazu srovnatelné s tradičními metodami, jak z hlediska kvality, tak implementačních nákladů [12].

### 4.2.1 Funkce posuvného okna

Softwarové a hardwarové implementace filtrů obrazu většinou pracují v prostorové doméně. Tyto filtry analyzují hodnoty pixelů kolem určitého centrálního pixelu, v oblasti známé jako jádro (kernel) filtru. Pro realizaci tohoto konceptu je důležitá implementace funkce lokálního sousedství, často popisované jako funkce posuvného okna. Ta se aplikuje individuálně na každý pixel a je konzistentní pro různé části obrazu [21].

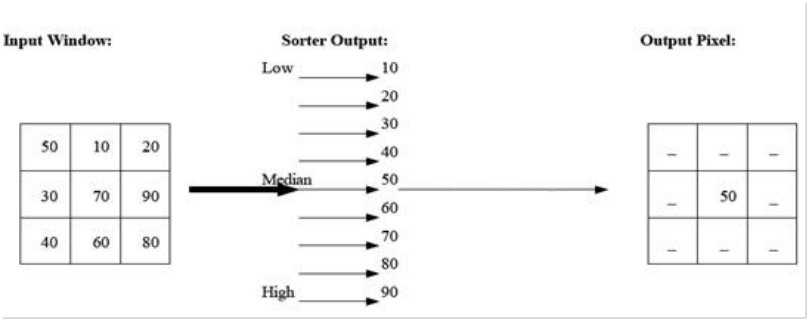



Obrázek 4.5: Demonstrace využití funkce posuvného okna, převzato z [13].

### 4.2.2 Konvenční filtry

Nejčastěji používaným nelineárním filtrem pro odstranění impulsního šumu je mediánový filtr. Výstupní hodnota pixelu po aplikaci tohoto filtru je rovna mediánu hodnot jádra. Nevýhodou je, že tento filtr není efektivní s vysokým šumem. Pro nízký šum do 20% stačí jednoduchý mediánový filtr s velikostí jádra 3x3 nebo 5x5 [21].

Lepších výsledků při práci s obrazy s vysokým šumem dosahují adaptivní mediánové filtry, ty operují s jádrem o velikosti $S_{max} \times S_{max}$ pixelů. Jádro je postupně zpracováno řadícími sítěmi se vstupy o velikosti 3x3, 5x5, ..., $S_{max} \times S_{max}$. Výstupem každé z těchto sítí je minimální hodnota, maximální hodnota a medián daného jádra. Celková výstup filtruje spočten na základě výstupů jednotlivých řadících sítí [21].




Obrázek 4.6: Mediánový filtr, převzato z [4].

### 4.2.3 Evoluční návrh filtrů

Každý obrazový filtr s jádrem o velikosti 3 x 3 lze brát jako funkci s devíti osmi-bitovými vstupy a jedním osmi-bitovým výstupem. Hodnota každého pixelu obrázku, jenž je zpracován tímto filtrem, je spočtena z jeho vlastní hodnoty a hodnot jeho okolí definovaného velikostí jádra.

Tedy například v CGP lze kandidátní řešení reprezentovat jako $N_c \times N_r$ uzlů, kdy každý z nich reprezentuje funkci se dvěma vstupy a jedním výstupem [21].

| Code | Function | Description | Code | Function | Description |
|---|---|---|---|---|---|
| 0 | 255 | Constant | 8 | $x \gg 1$ | Right shift by 1 |
| 1 | $x$ | Identity | 9 | $x \gg 2$ | Right shift by 2 |
| 2 | $255-x$ | Inversion | 10 | $swap(x,y)$ | Swap nibbles |
| 3 | $x \vee y$ | Bitwise OR | 11 | $x+y$ | + (addition) |
| 4 | $\bar{x} \vee y$ | Bitwise $\bar{x}$ OR y | 12 | $x +^S y$ | + with saturation |
| 5 | $x \wedge y$ | Bitwise AND | 13 | $(x+y) \gg 1$ | Average |
| 6 | $\overline{x \wedge y}$ | Bitwise NAND | 14 | $max(x,y)$ | Maximum |
| 7 | $x \oplus y$ | Bitwise XOR | 15 | $min(x,y)$ | Minimum |

Obrázek 4.7: Seznam funkcí které lze použít v uzlech, převzato z [21].

Při návrhu filtrů CGP obvykle používá pouze jeden genetický operátor - mutaci modifikující 5% chromozomu [21]. Inicializace populace na začátku algoritmu probíhá náhodně. Evoluce je obvykle ukončena po vyčerpání předem daného počtu generací [21].

Jako fitness funkci pro ohodnocení kvality řešení lze použít například peak signal-to-noise ratio (PSNR). Tato fitness funkce porovnává rozdíly mezi originálním obrázkem bez šumu a obrázkem po aplikace filtru. V algoritmu CGP se ji snažíme maximalizovat.

$$PSNR = 10 \log_{10} \left( \frac{255^2}{\frac{1}{MN} \sum_{i,j} (v(i,j) - w(i,j))^2} \right),$$

kde $M \times N$ je velikost obrazu, $v$ značí obraz po aplikaci filtru a $w$ značí původní originální obraz bez šumu. Nicméně výpočet této objektivní je drahý, proto je výhodnější použít funkci mean difference per pixel (MDPP), tj. střední absolutní chyba:

$$MDPP = \frac{1}{MN} \sum_{i=1}^{M} \sum_{j=1}^{N} |v(i,j) - w(i,j)|$$

Avšak MDPP jako účelovou funkci se snažíme v průběhu algoritmu minimalizovat [21].

# Kapitola 5

# Návrh řešení

Cílem teto práce je vytvořit model, který bude schopen obousměrně transformovat kandidátní řešení algoritmu genetického programování (GP) z běžně používané struktury syntaktického stromu na novou reprezentaci, která bude schopna lépe zachytit sémantické i syntaktické vlastnosti daného jedince. Tento model by měl umožnit převod mezi tradiční stromovou strukturou a novou reprezentací tak, aby bylo možné efektivně analyzovat a upravovat kandidátní řešení.

V článku, ze kterého tato práce vychází [7], je cílem GP symbolické regrese. Ta se snaží najít matematickou funkci, která nejlépe popisuje vztahy mezi proměnnými na základě vstupních dat a cílových výsledků, aniž by bylo nutné předem specifikovat formu této funkce. V našem případě se pokusíme uplatnit podobný princip na GP pro návrh obrazových operátorů.

## 5.1 Navržená reprezentace - Distributed embedding

Podobně jako v úlohách natural language processing (NLP), kde se větné struktury transformují na distribuované reprezentace (embeddings) pomocí transformačních modelů, lze i syntaktický strom v GP chápat jako strukturu jazyka jehož, slovník je tvořen sjednocením množiny neterminálů – funkce se dvěma osmi-bitovými vstupy a jedním osmi-bitovým výstupem a množiny terminálů – hodnoty pixelů oblasti jádra filtru v rozmezí 0 - 255. Tento přístup umožňuje modelům strojového učení zachytit komplexní vztahy a vzory v rámci stromu, což potenciálně zlepšuje návrh a funkčnost vytvořeného filtru. Tyto modely mohou efektivně zpracovávat strukturu stromu, kde kontext každé funkce či hodnoty pixelu mají vliv na výslednou podobu reprezentace [7].

Přínosem této reprezentace je potenciál zachytit sémantické vlastnosti v podobě vektorů reálných čísel. Pokud by se tedy dala tato reprezentace využít přímo v algoritmu GP genetické operátory by mohly přímo pracovat s vlastmi kandidátních řešení na rozdíl od klasické reprezentace syntaktického stromu, kde ovlivňují pouze jeho strukturu. Další potenciální využití jako nahrazení editační vzdálenosti stromu jsou diskutovány v [7].

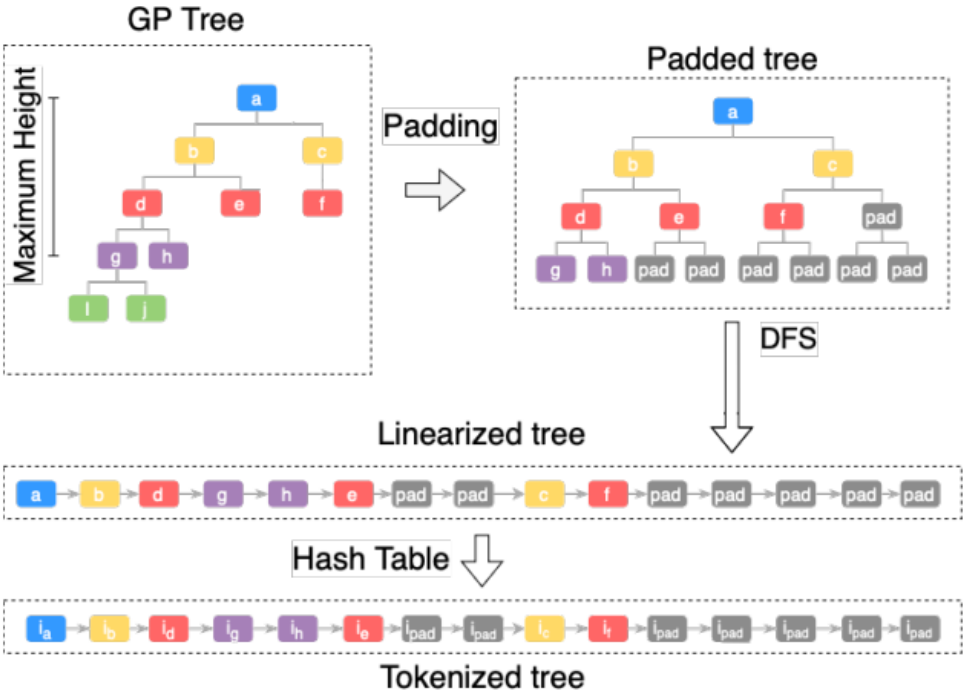
Transformátory jsou neuronové sítě, které byly vyvinuty k vyřešení tohoto problému. Hlavní rozdíl mezi transformátory a jinými rekurentními sítěmi je mechanismus vlastní pozornosti (self-attention). Například v NLP spočívá vlastní pozornost v násobení hodnoty, která reprezentuje slovo w, faktorem, který reprezentuje, zda ostatní slova ovlivňují význam slova w.

Z podstaty problému je vhodná architektura pro neuronovou síť pro převod reprezentace mezi stromovou strukturou a distribuovanou reprezentací typu enkodér-dekodér. Tyto dvě části transformeru se trénují současně, a po natrénování lze enkodér použít pro získání embeddingu ze stromové strukturu a dekodér pro opačný proces.

Výše popsaná neuronová síť očekává na vstupu lineární strukturu fixní velikosti, což syntaktický strom nesplňuje. Proto je třeba jej nejprve předzpracovat [7].

## 5.2 Předzpracování syntaktického stromu

Nejprve definujeme maximální hloubku stromu. Uzly stromu přesahující tuto hloubku jsou odstraněny. Větvě stromu kratší něž maximální hloubka jsou doplněny o speciální "padding"uzly. Tyto praktiky jsou podobné jako při práci s textem v NLP. Následuje linearizace stromové strukturu pomocí algoritmu depth first search (DFS). Takto zpracované stromy tokenizujeme pomocí hashovací tabulky, která převede jednotlivé uzly na numerické hodnoty - tokeny. Výstupem tohoto procesu je tokenizovaný strom, který může být použit jako vstup pro neuronovou síť [7].



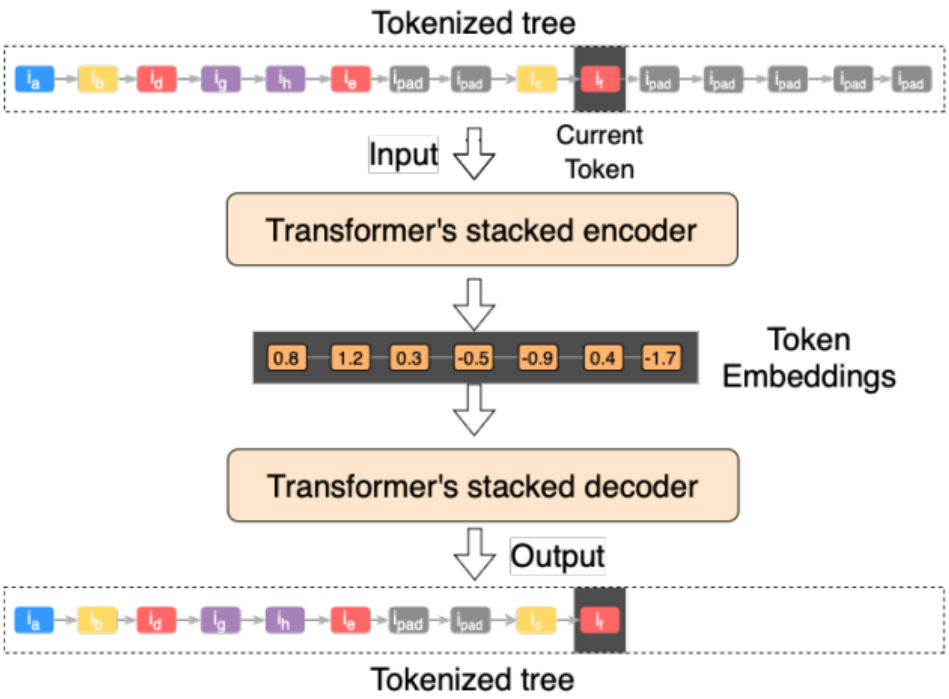

Obrázek 5.1: Proces převodu syntaktického stromu na tokenizovaný strom, převzato z [7].

## 5.3 Transformer

Jak bylo již zmíněno, transformer je rozdělen na dvě části - enkodér a dekodér. Trénování těchto částí probíhá současně. Enkodér mapuje tokenizovaný strom na abstraktní strukturu (embedding). Tato struktura obsahuje zakódované informace a vlastnosti vstupního stromu, které se enkodér naučil extrahovat. Enkodér zpracovává vstupní strom token po

tokenu, nicméně každý token má informace o svém okolí a učí se nastavovat větší váhy relevantním tokenům z tohoto okolí. V podstatě se jedná s schopnost učit se kontext. Výstupem zpracování každého tokenu je embedding o délce vstupního stromu

Následuje dekoder, do kterého vstupují jednotlivé embeddingy, a ten z nich generuje jednotlivé tokeny. Tento proces je autoregresivní, tedy každý výstup je součástí následujícího vstupu [7].

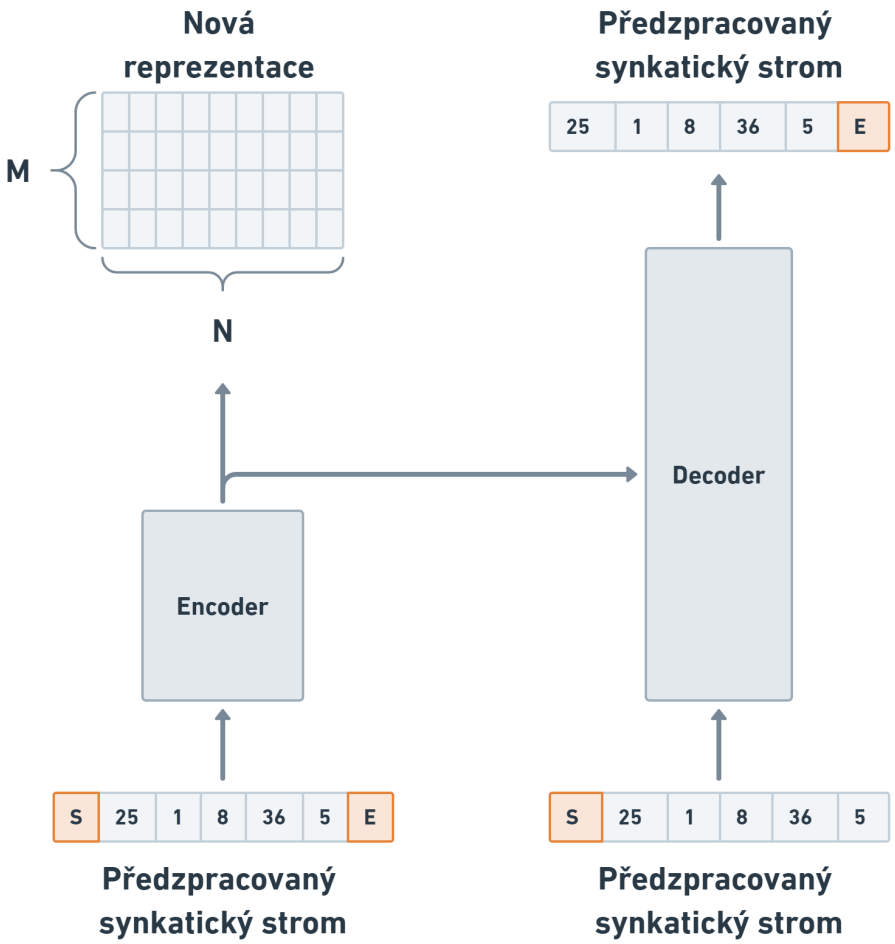



Obrázek 5.2: Proces trénování transformeru, převzato z [7].

Poté, co je transformer natrénován, jeho jednotlivé části lze odděleně používat pro převod mezi reprezentacemi syntaktického stromu a distribuované reprezentace. Avšak jak již bylo uvedeno, enkodér generuje pro každý token ze vstupního stromu jeden embedding - tyto výstupy je nutné ještě agregovat například pomocí konkatenace, sumy, nebo průměru [7].

V článku [7] je brán model transformeru jako black box – tedy pouze v kontextu jeho vstupů a výstupů (pro více informací je čtenář odkázán na [25]). Autoři nezmiňují použití části dekodéru pro zpětný převod získané reprezentace do původní podoby syntaktického stromu.

Konkrétní model, který jsem se rozhodl použít je sequnce-to-sequnce transformer pro překlad z portugalštiny do angličtiny. Vhodným upravením vstupů a výstupů tohoto modelu jej můžeme trénovat pro "překlad"z reprezentace syntaktického stromu zpět do stejné reprezentace. Díky tomu můžeme navíc od enkodéru pro zakódování stromu do embeddingu využít i dekodér pro zpětné dekódování do původní reprezentace, což nám potenciálně umožní využívat novou reprezentaci přímo v algoritmu GP.




Obrázek 5.3: Využití seq-to-seq modelu Transformeru pro trénování převodu syntaktického stromu do nové reprezentace a zpět do původní podoby. Jednotlivé vstupy a výstupy jsou rozšířeny o speciální znaky $S$ - začátek sekvence, $E$ - konec sekvence.

V kontextu modelu transformeru je nově vzniklá reprezentace pouze mezivýsledek při řešení nějakého jiného problému, to lze vidět na obrázku 5.3. Získaná reprezentace má tvar matice jejíž rozměry jsou rovny $M$ - délka sekvence, $N$ - délka embeddingu pro jeden znak sekvence (parametr modelu transformeru).

Abychom mohli efektivně využít i dekodér pro zpětné získání původního stromu, nechceme matici nové reprezentace agregovat, abychom zabránili ztrátě nebo poškození zakódované informace.

## 5.4 Získání trénovacích dat

Trénovací data pro neuronovou síť lze generovat přímo z algoritmu GP, ve kterém bude následně stromová struktura nahrazena novou reprezentací. Při běhu algoritmu si ukládáme každé kandidátní řešení společně s jeho ohodnocením a jeho velikostí. Tímto způsobem

dokážeme z jednoho běhu algoritmu vygenerovat velké množství trénovacích dat. Důležité je, aby dataset vyváženě reprezentoval zástupce s nízkou i vysokou hodnotou fitness funkce a různou velikostí.

Dalším způsobem pro získání dat je využít pouze inicializační metodu GP (growth, full, ramped-half-and-half) pro tvorbu dat. Zatímco jedinci extrahovaní z GP v průběhu evoluce budou postupně konvergovat k menším hodnotám fitness funkce. Pomocí inicializační metody budeme schopni získat rozmanitá data s velkou variabilitou fitness funkce a velikostí stromu. Navíc jsou vynechány režie spojené s během GP, což umožní generaci většího počtu jedinců v kratším čase.

Kombinací výše zmíněných metod lze vytvořit banku validních syntaktický stromů. Z této banky lze poté vhodně navzorkovat jedince na základě jejich vlastností (fitness hodnota a velikost) a vytvořit dataset pro trénování neuronové sítě.

## 5.5 Algoritmus genetického programování

Konkrétní úlohou, kterou GP řeší, je odstranění impulsního šumu z obrázku. Fenotypem v tomto případě je filtr, jenž lze chápat jako matematickou funkci popisující vztah mezi vstupy a výstupy posuvného okna, které postupně zpracovává daný obrázek. Vstupními daty jsou pixely obrázku se šumem a výstupními pixely obrázku bez šumu.

# Kapitola 6

# Implementace

Všechny implementační části jsou napsány v jazyce Python. Jedná se o kombinaci skriptů a jupiter notebooků určených pro běh v cloudové platformě Kaggle.

Kaggle je cloudová platforma, která byla založena v roce 2010 a v roce 2017 ji převzala společnost Google. Tato platforma poskytuje rozmanité zdroje pro učení, soutěžení a spolupráci na projektech založených na analýze dat. Kaggle nabízí širokou škálu datových sad, softwarových knihoven a soutěží v různých oblastech, jako jsou ekonomie, počítačové vidění a zpracování přirozeného jazyka. Uživatelé mohou přispívat svými daty, stahovat datasety poskytované jinými, účastnit se soutěží s finančními odměnami nebo spolupracovat na projektech a zdokonalovat své dovednosti v datové analýze a strojovém učení. Platforma rovněž poskytuje prostředí zvané „Kernels“, kde uživatelé mohou psát a spouštět skripty, sdílet své analýzy a modely přímo na platformě [1].

Tato platforma poskytuje možnost tvorby vlastních datasetů a modelů strojového učení. Navíc poskytuje zdarma přístup k GPU pro trénování těchto modelů. Vzhledem pozitivním zkušenostem s touto platformou a možností trénování implementovaných modelů souběžně s dalším vývojem jsou veškeré skripty vytvořené v této DP primárně určeny pro běh na této platformě.

## 6.1 Algoritmus genetického programování

Implementace GP nevyužívá žádné existující knihovny pro práci s GP. Zahrnuje několik zdrojových souborů v jazyce Python poskytujících rozhraní pro práci se syntaktickými stromy a běh evoluce. Tyto soubory tvoří Python balíček, který je využíván jako základní nástroj v ostatních částech DP – tvorba datasetu, experimenty atd.

Výsledný balíček s názvem "GP"je strukturován následovně:

- **filter_functions.py** - Obsahuje definice funkcí, které tvoří množinu neterminálních symbolů jazyka syntaktického stromu. Jedná se o jednoduché matematické a logické operace, včetně bitových posunů, jako jsou sčítání, odčítání, AND, OR atd. Tyto funkce mají aritu buď jedna nebo dvě.
- **fitness_functions.py** - Obsahuje definici funkcí určených pro použití v evoluci jako fitness funkce.
- **image_loader.py** - Obsahuje kód pro načtení a předzpracováni obrázků, které obsahují impulsní šum a odpovídajících původních obrázků. Zašuměné obrázky jsou předzpracovány na pole vstupů do funkce posuvného okna. Původní obrázky jsou

zpracovány na pole očekávaných výstupů z funkce posuvného okna. Dvojice takto předzpracovaných obrázků je poté použita během evoluce pro výpočet hodnoty fitness funkce kandidátních řešení.

- **tree.py** - Zde je implementace tříd Tree a TreeConfig. Třída Tree implementuje rozhraní filtru ve struktuře syntaktického stromu. Tato implementace zahrnuje nejen funkce nezbytné pro evoluci jako inicializaci jedince nebo implementaci genetických operátorů, ale i celou řadu pomocných funkcí, které byly využiti během vývoje například pro testování gramatické korektnosti stromů. Vzhledem k tomu že třída Tree je základním stavebním kamenem celé práce je jako jediná pokryta unit testy, které lze nalézt v souboru test_tree.py.

  Třída TreeConfig veškeré nastavitelné parametry třídy Tree a umožňuje ukládání a načítaní konfigurací parametrů ve formátu JSON.

- **evolution.py** - Samotná implementace GP, která umožňuje konfiguraci algoritmu pomocí třídy GPConfig. Tyto konfigurace je možné ukládat a načítat ze souborů ve formátu JSON podobně jako v případě třídy Tree. Implementace GP je obohacena o třídu GP_Stats, která vykresluje grafy spojené s během evoluce. Popis implementace důležitých částí GP je popsán v následujících podkapitolách.

### 6.1.1 Reprezentace syntaktického stromu

Instinktivní způsob, jak zachytit data reprezentující strukturu syntaktického stromu, spočívá v použití objektů představujících uzly, které obsahují znak z primitivní množiny jazyka a odkazy na potomky. Nicméně, vzhledem k tomu, že tyto stromy by následně v kontextu této práce bylo třeba transformovat pomocí procesu popsaného v sekci 5.2, je výhodné reprezentovat stromy jako pole hodnot typu uint16. Délka těchto polí je fixní a v rámci jedné instance algoritmu GP je vypočítána jako $2^{(\text{max\_depth}+1)} - 1$, kde max_depth je parametr, který vyjadřuje maximální hloubku stromu povolenou v dané instanci algoritmu GP.


Obrázek 6.1: Na obrázku lze vidět syntaktické strom reprezentující filtr pomocí jednotlivých uzlů (vpravo) a odpovídající pole hodnot z look-up tabulky, které je využito v algoritmu GP (vlevo). Reprezentace je v obou případech doplněna do maximální velikosti v dané hloubce o speciální uzly s hodnotou padding. V případě syntaktického stromu se jedná o šedé uzly s textem "pad", zatímco v případě pole se jedné o prvky s hodnotou 0.

Prvky pole jsou při inicializaci nastaveny na hodnotu 0. Při tvorbě stromu jsou hodnoty uzlů do pole zapisovány v preorder pořadí. Díky tomu lze vztahy mezi prvky (uzly) dopočítat

pomocí vzorců:

$$\text{levý potomek}(i) = 2i + 1$$

$$\text{pravý potomek}(i) = 2i + 2$$

$$\text{rodič}(i) = \left\lfloor \frac{i-1}{2} \right\rfloor,$$

kde symbol $i$ odpovídá indexu prvku v poli reprezentace a výraz $\lfloor x \rfloor$ reprezentuje dolní celou část čísla $x$, což je v jazyce python nahrazeno celočíselným dělením.

Hodnoty v poli odpovídají hodnotám z look-up tabulky, která mapuje primitivní množinu jazyka filtru na celá čísla. Tabulka 6.1 zobrazuje primární look-up tabulku, která byla použita v teto práci pro generování jedinců do datasetu. Všechny ostatní tabulky jsou odvozeny z teto pomocí transformací a číselných posunů.

Zapsání syntaktického stromu do pole v sobě zahrnuje celý proces předzpracování stromu pro následné použití v modelu strojového učení. Zároveň umožňuje jednoduché ukládání jedinců ve formátu csv a snížení režie na převod mezi stromovou reprezentací a novou reprezentací vytvořenou z modelu strojového učení při potenciálním použití v algoritmu GP.

| Klíč: int | Hodnota: Prvek primitivní množiny |
|---|---|
| 0 | padd |
| 1 | 0 |
| 2 | 1 |
| ⋮ | ⋮ |
| 256 | 255 |
| 257 | b_or |
| 258 | b_nor |
| 259 | b_and |
| 260 | b_nand |
| 261 | b_xor |
| 262 | add |
| 263 | sat_add |
| 264 | avg |
| 265 | f_max |
| 266 | f_min |
| 267 | sub |
| 268 | sat_sub |
| 269 | mul |
| 270 | pdiv |
| 271 | idx |
| 272 | inv |
| 273 | b_r1 |
| 274 | b_r2 |
| 275 | [0, 0] |
| 276 | [0, 1] |
| ⋮ | ⋮ |
| 279 | [0, 4] |
| 280 | [1, 0] |
| ⋮ | ⋮ |
| 299 | [4, 4] |
| 300 | SOS |
| 301 | EOS |

Tabulka 6.1: Look-up tabulka mapující prvky primitivní množiny na celé čísla. Klíč 0 je vyhrazen pro speciální symbol padding, hodnoty 1-256 odpovídají konstantám, 257-270 funkce s aritou dva, 271-274 funkce s aritou jedna, 275-299 vstupy z funkce posuvného okna. Klíč 300 reprezentuje speciální symbol, který značí začátek sekvence (SOS), obdobně klíč 301 značí konec sekvence (EOS).

### 6.1.2 Inicializace populace

Inicializace instance třídy Tree je implementováno ve funkci random_growth. Tato funkce rekurentně sestavuje strom dle syntaktických pravidel z primitivní množiny jazyka a postupně jej zapisuje do reprezentace v podobě pole hodnot z look-up tabulky. Pomocí parametrů function_prob je určena pravděpodobnost, zda je v daném uzlu stromu vygenerován neterminální symbol (funkce filtru). V opačném případě je v uzlu vygenerován terminální

symbol, a to buď vstup z posuvného okna s pravděpodobností dle parametru input_prob nebo v opačném případě konstanta. Pokud je hodnota function_prob rovna jedné, odpovídá inicializace jedince metodě full, jinak odpovídá metodě growth. Tyto metody jsou popsány v sekci 3.5.2.

### 6.1.3 Evoluce

Evoluce instance GP je spočtena pomocí funkce evolve. Před spuštěním evoluce je třeba nastavit její parametry pomocí třídy GPConfig. Parametry evoluce jsou následující:

- **population_size** - Velikost populace GP.
- **number_of_generations** - Počet iterací, po kterých je evoluce zastavena.
- **tournament_size** - Velikost turnaje při selekci.
- **mutation_prob** - Pravděpodobnost tvorby potomka pomocí mutace.
- **crossover_prob** - Pravděpodobnost tvorby potomka pomocí křížení.
- **seed** - Seed pseudo náhodného generátoru čísel, který umožňuje replikovat konkrétní běh algoritmu.
- **max_depth** - Maximální hloubka stromů v GP.
- **function_prob** - Pravděpodobnost vytvoření uzlu s funkcí během inicializace jedince.
- **input_prob** - Pravděpodobnost vytvoření uzlu se vstupem klouzavého okna během inicializace jedince.

Během evoluce je nejprve inicializována populace, poté je proveden určitý počet iterací definovaných parametrem number_of_generations. V každé iteraci je nejprve populace vyhodnocena pomocí fitness funkce. Prakticky jediné fitness funkce používané během celého vývoje a všech experimentů je MDPP. Tato fitness funkce je podrobněji popsána v sekci 4.2.3. Poté je v GP uložena kopie nejlepšího jedince z populace. Následuje tvorba nové populace, která se skládá ze dvou kroků. prvním krokem je selekce pomocí turnaje (viz. sekce 3.5.3). Velikost turnaje je určena parametrem GP tournament_size. Poté je z vybraného jedince vytvořen potomek buď pomocí křížení nebo mutace. V GP jsou implementovány dvě varianty mutace a to mutace jednoho uzlu a mutace podstromu. Šance vybrání varianty mutace je v obou případech 50%. Implementovaná varianta křížení je křížení podstromu. Využité varianty genetických operátorů jsou detailněji popsány v sekci 3.5.4.

## 6.2 Transformer

Implementance modelu transformeru včetně jednotlivých vrstev je z velké části převzata z [24] a upravena tak, aby ji bylo možné využít v rámci této práce. Prvním úkolem bylo zprovoznění původního modelu pro překlad z portugalštiny do angličtiny. Na základě správné funkcionality tohoto původního modelu byly poté implementovány potřebné úpravy a rozšíření a vznikl finální model, který lze použít pro trénovaní na převod mezi reprezentacemi. Původní funkční model je možné najít ve jupiter notebooku original_pt_eng_transformer.ipynb.

Finální implementace transformeru je v podobě python balíčku s názvem Transformer. Tento balíček je určen pro použití na platformě Kaggle. Balíček jo možné nahrát na Kaggle jako součást datasetu a poté je importovat viz obrázek 6.2.

```
import sys
sys.path.append("/kaggle/input/filters/Transformer")
from transformer import Transformer, masked_loss, masked_accuracy
```

Obrázek 6.2: Importování modelu Transformeru do v prostředí Kaggle jupiter notebooku.

Před importování datasetu je však třeba upravit verze předinstalovaných python balíčků, viz obrázek 6.3.

```
# Install CUDA Deep Neural Network library (cuDNN)
#!apt-get install --allow-change-held-packages libcudnn8=8.1.0.77-1+cuda11.2
# Uninstall current TensorFlow and related packages
!pip uninstall -y -q tensorflow keras tensorflow-estimator tensorflow-text
# Install a specific version of protobuf
!pip install -q protobuf==3.20.3
# Install specific versions of TensorFlow and TensorFlow Text
!pip install -q tensorflow==2.15.0 tensorflow-text==2.15.0
# Install TensorFlow datasets
!pip install -q tensorflow_datasets==4.5.2
```

Obrázek 6.3: Upravení prostředí Kaggle pro zajištění kompatibility s imporotvaným modelem transformeru.

Před samotným trénováním transformeru je třeba nahrát dataset ve formě csv souboru, kde na každém řádku je jedno kandidátní řešení syntaktického stromu v reprezentaci používané v GP (viz 6.1.1 ). Reprezentace nahrané z csv souboru jsou rozšířeny rozšířeny o SOS a EOS symboly a pomocí funkce make_dataset_from_numpy zpracovány na trénovací a validační batche.

Jako loss funkce při trénování modelu se používá masked_loss. Jedná se SparseCategoricalCrossentropy z frameworku Keras rozšířenou, tak aby zohledňovala pouze části stromů, které neobsahují symboly padding. Další metrikou je masked_accuracy, která počítá přesnost predikce modelu avšak obdobně jako masked_loss vynechává části se symboly padding.

Transformer vychází ze stejného modelu, jaký byl použit v článku, ze kterého tato DP vychází [7]. Detailnější technické informace o vnitřním fungování toho modelu je možné najít zde [24].

Parametry modelu:

- **BATCH_SIZE** - Velikost batchů použitých při trénování.
- **MAX_TOKENS** - Délka vstupního tensoru modelu. Odpovídá velikosti pole reprezentujícího filtr rozšířeného o symboly SOS a EOS.
- **SOS** - Hodnota klíče SOS symbolu v použité look-up tabulce.
- **EOS** - Hodnota klíče EOS symbolu v použité look-up tabulce.

- **VOCAB_SIZE** - Velikost slovníku modelu. Odpovídá velikosti look-up tabulky.
- **num_layers** - Počet bloků v enkodéru a dekodéru.
- **dff** - Velikost plně propojených vrstev modelu.
- **d_model** - Velikost embeddingu příslušícímu jednomu symbolu.
- **num_heads** - Počet hlav v Multi-Head Attention vrstvách.
- **dropout_rate** - Míra dropoutu.
- **epochs** - Počet epoch trénování modelu.
- **dataset_path** - Relativní cesta k datasetu.

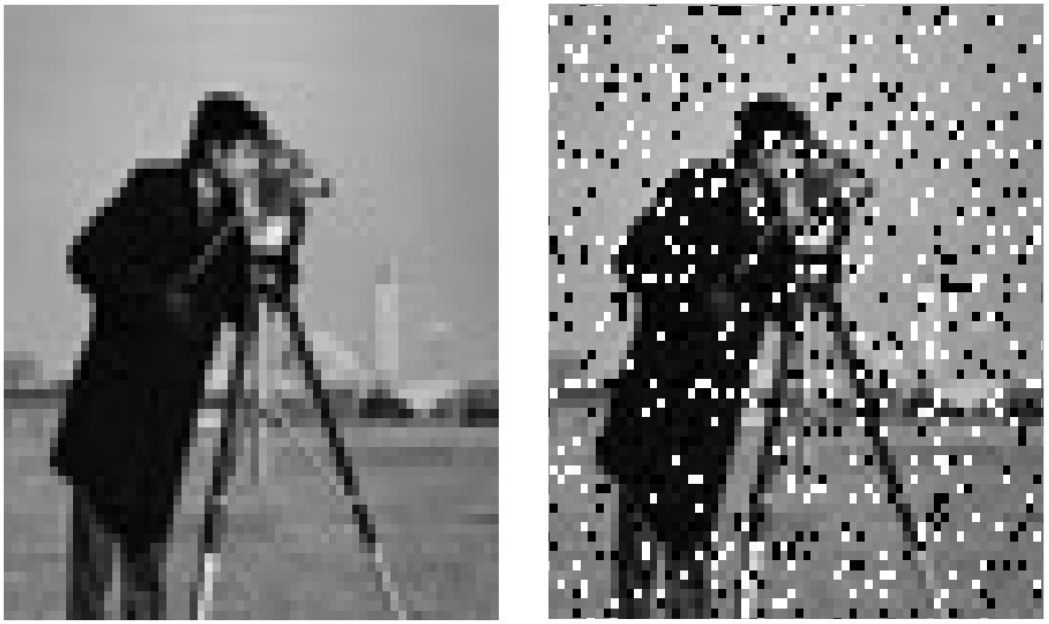
Po natrénování je možné model uložit ve formátu ".keras". Použití modelu spočívá v implementovaných rozšiřujících metodách encode, decode a decode_batch. Metoda encode využije enkodér transformeru pro vytvoření nové reprezentace z reprezentace syntaktického stromu ve formě pole. Metody decode a decode_batch slouží ke zpětnému dekódování na původní reprezentaci.

# Kapitola 7

# Experimenty

V této kapitole jsou popsány jednotlivé experimenty s nástroji, který byly vytvořeny během implementační fáze. Jednotlivé podkapitoly jsou v posloupnosti odpovídající pořadí, ve kterém byly provedeny. Experimenty na sebe logicky navazují.

Veškeré běhy GP v této práci jejichž účelem je odstranění impulsního šumu využívají trénovací obrázek fotografa z datasetu Set12[1] o velikosti $64 \times 64$, který obsahuje 15% impulsní šum viz obrázek 7.1.



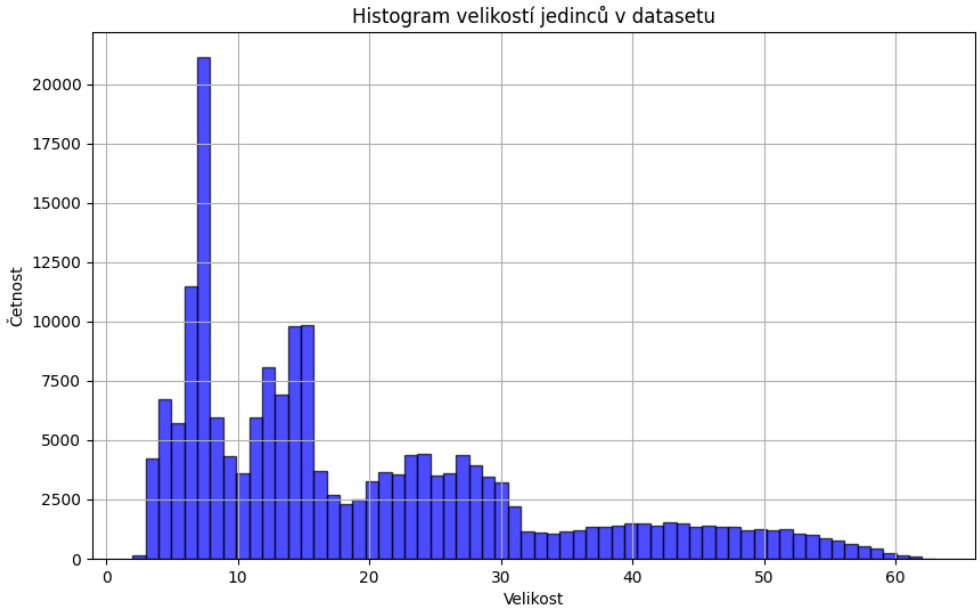
Obrázek 7.1: Dvojice obrázků o velikosti $64 \times 64$ používaná při bězích GP v rámci experimentů. Referenční obrázek je vlevo a obrázek s 15% impulsním šumem vpravo.

[1] https://paperswithcode.com/dataset/set12

## 7.1 Tvorba datasetu

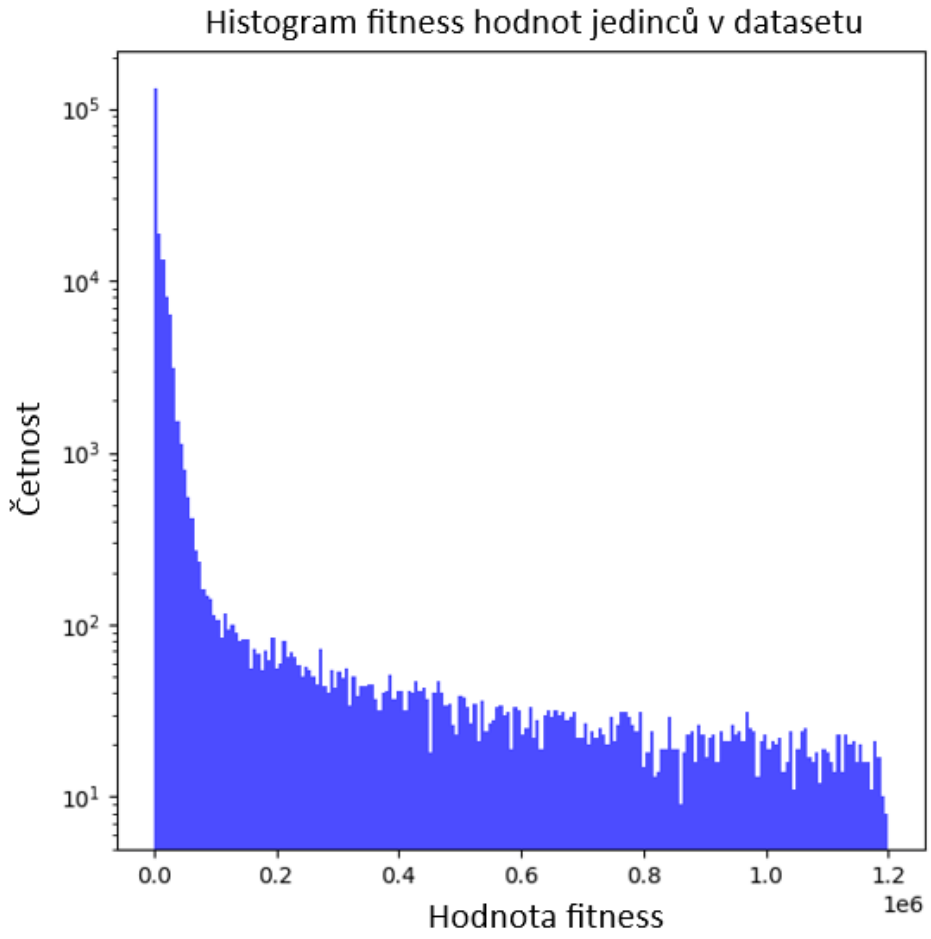
Prvním krokem vytvoření datasetu pro trénování modelu transformeru je vygenerování banky kandidátních řešení pomocí evoluce GP a inicializační metody ramped-half-and-half. K tomuto účelu byly vytvořeny dva skripty dp-filter-growth-v2.ipynb a dp-filter-evolution-v2.ipynb. Oba tyto skripty jsou určeny pro spuštění v prostředí Kaggle. Pomocí těchto skriptů bylo vygenerováno 818 533 unikátních syntaktických stromů o maximální velikosti 63 uzlů.

Vzhledem k nevyváženému poměru rozdělení hodnot fitness funkce těchto jedinců byla vygenerovaná data rozdělena do tří kategorií podle jejich fitness hodnot. První skupina tzv. outliers pro fitness hodnoty větší než 10000. Druhá skupina data_high pro fitness hodnoty od 500 do 10 000 a poslední skupiny data_low pro fitness hodnoty 500 a menší. Z těchto skupin byly poté navzorkovány jedinci a z nich vytvořeny výsledné datasety. Tento proces je implementován ve zdrojovém souboru dataset_processing.py




Obrázek 7.2: Histogram zobrazující rozdělení velikostí jedinců v datasetu 192_96-48-48.

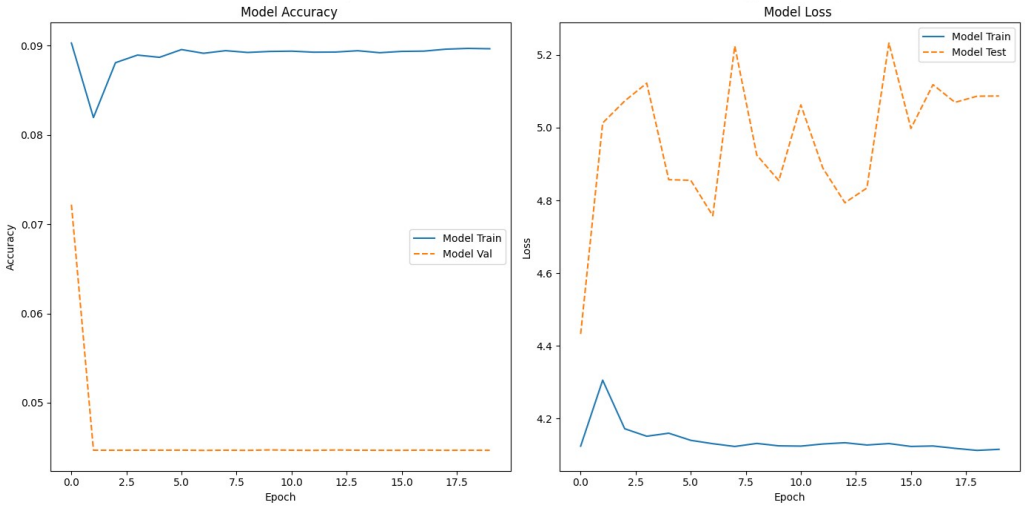
Takto byly vytvořeny dva datasety. Dataset "192_96-48-48", který je tvořen z 192 000 jedinců z nichž 96 000 je ze skupiny data_low, 48 000 ze skupiny data_high a 48 000 ze skupiny outliers. Druhý dataset "64_32-24-8", který je tvořen z 32 000 jedinců ze skupiny data_low, 24 000 jedinců ze skupiny data_high a 8 000 z outliers.




Obrázek 7.3: Histogram zobrazující rozdělení velikostí jedinců v datasetu 192_96-48-48.

## 7.2 Trénování modelu transformeru

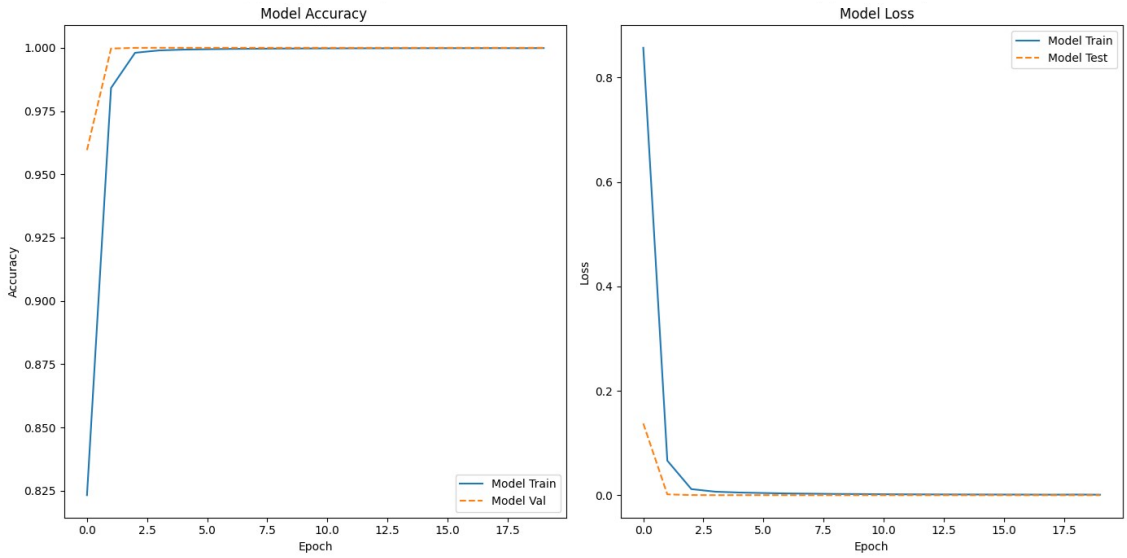
První trénování modelu transformeru proběhlo s nad datasetem 192_96-48-48. Dataset byl náhodně rozdělen na trénovací část o velikosti 153 600 (80%) a validační část o velikosti 38 400 (20%). Trénovací data byla rozdělena na batche o velikosti 64. Slovník modelu vycházel z look-up tabulky na obrázku 6.1. Model by trénován na GPU typu P-100. Trénování modelu ve 20 epochách trvalo celkem 105 minut. Struktura modelu odpovídala původnímu modelu pro překlad z portugalštiny do angličtiny.




Obrázek 7.4: Průběh trénování prvního modelu.

Z grafů na obrázku 7.4 je zjevné, že se model neučí. To by mohlo by způsobeno rozdílným významem speciálního symbolu padding v rámci jedince. Zatímco ve vektoru, který představuje zakódovaný strom v jazce jako portugalština nebo angličtina nemá padding v rámci vektoru žádný význam.. V reprezentaci pole představuje hodnota na daném indexu uzel, který je v rámci stromu vždy na stejné pozici. Symbol padding by tedy mohl představovat "prázdný"uzel, který by potenciálně mohl pomoci transformeru k lepšímu pochopení struktury stromu.

Na základě této myšlenky byly před učením transformovány všechny symboly padding v datasetu z hodnoty 0 na hodnotu 301, která není brána jako speciální symbol, ale jako běžné slovo ze slovníku modelu. Trénování stejného modelu na upraveném slovníku je možné vidět na obrázku 7.5.




Obrázek 7.5: Průběh trénování prvního modelu s upraveným slovníkem, kde speciální symbol padd je brán jako běžné slovo.

Model, který pracoval s upraveným slovníkem, vykazoval stabilnější průběh trénování a také dosáhl relativně vysoké přesnosti již během několika málo epoch. Existuje několik důvodů, proč se model dokáže natrénovat v tak krátké době. Jedním z nich může být například velký rozdíl ve velikosti slovníků: zatímco slovník původního modelu obsahuje přes 7000 tokenů, slovník syntaktického stromu, který reprezentuje filtr, má pouze 300 tokenů. Dalším faktorem může být velikost datasetu, kde velikost původního datasetu je mnohem menší než u datasetu 192_96-48-48. Trénování veškerých modelů v této práci proběhlo pomocí skriptu tree-transformer.ipynb

## 7.3 Přehled modelů

Celkem bylo vytvořeno a natrénováno a vyhodnoceno 20 modelů. V rámci různých konfigurací modelu je kladen důraz na celkovou velikost modelu, velikost reprezentace a variantu s paddingem jako speciálním symbolem a bez. Některé modely byly trénovány na speciálně upraveném slovníku, kdy 256 symbolů slovníku reprezentujících konstanty byly nahrazeny pouze 4 symboly, která sdružovaly konstanty v intervalech o velikosti 64. Tato úprava však neměla žádný zásadní vliv na chování modelu. Přehled natrénovaných modelů lze vidět v tabulce 7.1.

| Název modelu | loss | acc | val_loss | val_acc |
|---|---|---|---|---|
| 2_4_128_model_1_padd_16 | 0.0196841 | 0.9963772 | 0.0002740 | 1.0 |
| 2_4_128_model_1_padd_32 | 0.0025020 | 0.9995628 | 0.0000021 | 1.0 |
| 2_4_128_model_1_padd_64 | 0.0012503 | 0.9997965 | 0.0000014 | 1.0 |
| 2_4_128_model_64_padd_16 | 0.0062543 | 0.9986272 | 0.0000159 | 1.0 |
| 2_4_128_model_64_padd_128 | 0.0004559 | 0.9999155 | 0.0000003 | 1.0 |
| 2_4_256_model_64_padd_16 | 0.0044968 | 0.9990441 | 0.0000247 | 1.0 |
| 2_4_256_model_64_padd_128 | 0.0004367 | 0.9999241 | 0.0000007 | 1.0 |
| 2_4_512_model_64_padd_16 | 0.0049197 | 0.9989948 | 0.0000207 | 1.0 |
| 2_4_512_model_64_padd_128 | 0.0006800 | 0.9998932 | 0.0000026 | 1.0 |
| 2_8_256_model_64_padd_16 | 0.0032378 | 0.9992896 | 0.0000016 | 1.0 |
| 2_8_256_model_64_padd_128 | 0.0003323 | 0.9999354 | 0.0000005 | 1.0 |
| 2_8_512_model_64_padd_16 | 0.0038771 | 0.9991767 | 0.0000053 | 1.0 |
| 2_8_512_model_64_padd_128 | 0.0003919 | 0.9999347 | 0.0000015 | 1.0 |
| basic_model_1_128 | 4.1153584 | 0.0896461 | 5.0871162 | 0.0446704 |
| basic_model_1_padd_128 | 0.0008874 | 0.9998798 | 0.0000009 | 1.0 |
| basic_model_64_128 | 3.3818576 | 0.1100343 | 4.2804828 | 0.0444540 |
| basic_model_64_padd_8 | 0.2108711 | 0.9407723 | 1.4399203 | 0.7497746 |
| basic_model_64_padd_16 | 0.0040904 | 0.9991941 | 0.0000474 | 1.0 |
| basic_model_64_padd_32 | 0.0010678 | 0.9998078 | 0.0000022 | 1.0 |
| basic_model_64_padd_64 | 0.0007155 | 0.9998763 | 0.0000025 | 1.0 |
| basic_model_64_padd_128 | 0.0006601 | 0.9999018 | 0.0000046 | 1.0 |

Tabulka 7.1: Přehled modelů natrénovaných na datasetu 192_96-48-48. Parametry modelu v názvech modelu jsou zapsány v pořadí num_layers, num_heads, dff nebo "basic_model"pokud hodnoty odpovídají původnímu transformeru z [24]. Číslo za slovem model označuje velikost intervalu konstant pro konstanty. Pokud název obsahuje slovo "padd"není padding brán jako speciální symbol, ale jako součást slovníku. Poslední číslo v názvu odpovídá parametru d_model (velikost reprezentace).

## 7.4 Vyhodnocení enkodéru a dekodéru modelu

Vyhodnocení enkodéru modelu je pojato podobně jako ve studii [7]. Pro tento účel byla k datasetu 192_96-48-48 dodatečně vygenerována testovací sada obsahující 1280 nových jedinců. Mezi páry těchto jedinců byla změřena tree edit distance. Následně byli jedinci transformováni pomocí enkodéru do nové reprezentace, mezi kterou byla změřena cosinová vzdálenost. Mezi těmito metrikami byla následně vypočítána Spearmanova korelace. Pokud by byla korelace mezi těmito metrikami velmi vysoká, naznačovalo by to, že nová reprezentace zachycuje informace ekvivalentní původní reprezentaci. Naopak, malý rozdíl v korelaci by mohl naznačovat, že nová reprezentace je schopna zachytit více informací než původní. Vzhledem k tomu, že původní reprezentace je syntaktický strom, by to znamenalo, že nová reprezentace dokáže odhalit část sémantické informace, která je relevantní pro danou problematiku. Jelikož reprezentace jsou ve tvaru matice a cosinová vzdálenost se počítá mezi vektory, musí být matice linearizovány do jednorozměrného vektoru (jednotlivé vektory jsou poskládány za sebe) mezi těmito vektory je poté spočítána cosinová vzdálenost. Takto spočtená cosinová vzdálenost bude dále označována jako overall_cosine.

Druhým způsobem výpočtu cosinovi vzdálenosti mezi maticemi v rámci vyhodnocení reprezentace je avarage_cosine - průměrná kosínova vzdálenost mezi odpovídajícími sloupci matic. Těmto dvěma způsobům výpočtu odpovídají koeficienty Spearmanovy korelace - overall_Spearman a avarage_Spearman.

K vyhodnocení dekodéru je použita stejná testovací sada o 1280 jedincích. Z těchto jedinců je vytvořeno 20 batchů o velikosti 64. Tyto batche jsou postupně převedeny pomocí enkodéru do nové reprezentace a poté pomocí dekodéru zpět. Nakonec je změřeno do jaké míry odpovídají symboly převedeného stromu symbolům stromu původního (v tabulce 7.2 je tato metrika nazvána jako Přesnost D ).

Tato vyhodnocení jsou implementována ve skriptu tree-transformer-eval.ipynb.

| Název modelu | Overall_S | Avarage_S | Přesnost D | Trvání |
|---|---|---|---|---|
| 2_4_128_model_1_padd_16 | 0.98612 | 0.98893 | 1.0 | 9.5 |
| 2_4_128_model_1_padd_32 | 0.98968 | 0.99045 | 1.0 | 11.7 |
| 2_4_128_model_1_padd_64 | 0.95953 | 0.95655 | 1.0 | 14.1 |
| 2_4_128_model_64_padd_16 | 0.99250 | 0.99275 | 1.0 | 9.9 |
| 2_4_128_model_64_padd_128 | 0.99538 | 0.99633 | 1.0 | 15.6 |
| 2_4_256_model_64_padd_16 | 0.99176 | 0.99259 | 1.0 | 8.2 |
| 2_4_256_model_64_padd_128 | 0.99364 | 0.99529 | 1.0 | 16.5 |
| 2_4_512_model_64_padd_16 | 0.98821 | 0.98941 | 1.0 | 8.4 |
| 2_4_512_model_64_padd_128 | 0.98746 | 0.99112 | 1.0 | 15.9 |
| 2_8_256_model_64_padd_16 | 0.99118 | 0.99357 | 1.0 | 10.9 |
| 2_8_256_model_64_padd_128 | 0.99044 | 0.99183 | 1.0 | 23.0 |
| 2_8_512_model_64_padd_16 | 0.99129 | 0.99266 | 1.0 | 10.3 |
| 2_8_512_model_64_padd_128 | 0.99213 | 0.99456 | 1.0 | 25.5 |
| basic_model_1_128 | 0.09141 | 0.10957 | 0.00547 | 55.2 |
| basic_model_1_padd_128 | 0.99231 | 0.99183 | 1.0 | 53.8 |
| basic_model_64_128 | 0.08300 | 0.16269 | 0.00547 | 51.7 |
| basic_model_64_padd_8 | 0.96299 | 0.96326 | 0.77965 | 19.6 |
| basic_model_64_padd_16 | 0.98833 | 0.98968 | 1.0 | 21.3 |
| basic_model_64_padd_32 | 0.98937 | 0.98987 | 1.0 | 24.9 |
| basic_model_64_padd_64 | 0.99541 | 0.99594 | 1.0 | 32.9 |
| basic_model_64_padd_128 | 0.98476 | 0.98183 | 1.0 | 53.5 |

Tabulka 7.2: Vyhodnocení enkodéru a dekodéru modelů. Overall_S je Overall_Spearmanův koeficient. Avarage_S je Avarage_Spearmanův koeficient, Přesnost D je přesnost dekodéru, Trvání je průměrná doba dekódování jednoho batche v sekundách.

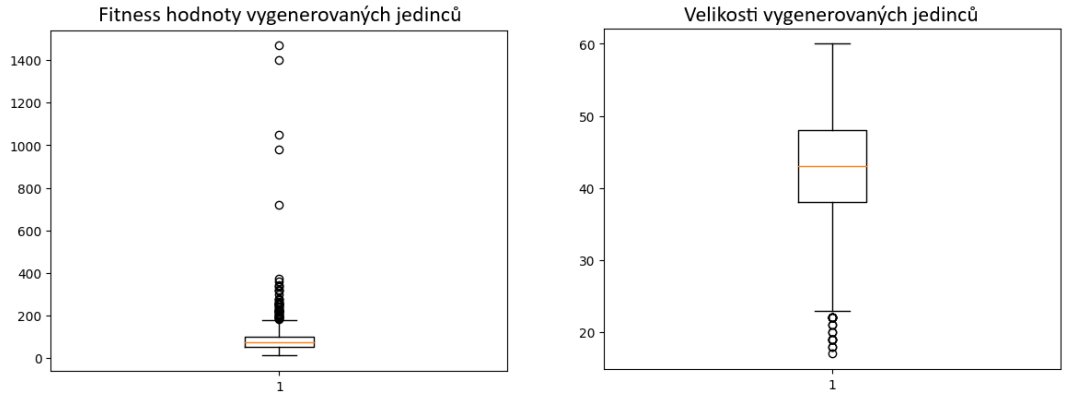
Modely jejichž slovník obsahuje padding jako speciální znak (0) jsou vyhodnoceny nejhůře. Velikost reprezentace v rámci modelů se zdá mít v rozsahu 16-128 pouze malý vliv na korelaci vzdáleností avšak zásadně ovlivňuje dobu trvání dekódování. Reprezentace o velikost 8 dosahuje poměrně vysoké hodnoty korelace vzdáleností, ale dekodér již nedokáže přesně transformovat novou reprezentaci do reprezentace původní. Nejlepších výsledků dosahují nejmenší modely s velikostí reprezentace 16, 32 a 64. Korelace těchto modelů dosahuje vysokých hodnot, ale zároveň zanechává dost prostoru pro potenciální zachycení sémantiky. Přesnost a trvání dekódování těchto modelů umožňuje jejich využití v algoritmu GP.

## 7.5 Inicializace, mutace a křížení v nové reprezentaci

V rámci těchto experimentů se pracuje s modelem 2_4_128_model_1_padd_16. Velikost vektoru reprezentujícího jeden symbol (uzel) stromu v tomto modelu je 16. Při práci se stromy o velikosti 65 (včetně SOS a EOS) je tedy reprezentace ve tvaru matice o rozměrech $65 \times 16$.

Inicializace jedince v nové reprezentaci je jednoduše implementována jako vytvoření matice náhodných hodnot dané velikosti. U každé takto vygenerované reprezentace je třeba zjistit, zda se jedné o validní strom, tedy jestli po dekódování reprezentace vznikne syntakticky korektní jedinec. Toho lze docílit dekódováním reprezentace na stromovou reprezentaci a kontrolním výpočtem. Pokud je tento výpočet platný, strom je validní. V opačném případě se jedné o syntakticky špatně sestavený strom a reprezentace není validní. Bylo provedeno několik experimentů, které s lišily intervalem, ze kterých jsou hodnoty matice generovány. V každém z těchto experimentů bylo vygenerováno 6400 jedinců. Validita vygenerovaných jedinců se pohybovala v rozmezí $30-50\%$. Výsledky experimentu s nejvyšší mírou validních jedinců 52.62% jsou zobrazeny na obrázku 7.6.

Implementace inicializace jedince včetně provedených experimentů je ve skriptu dp-filter-inicialization.ipynb.




Obrázek 7.6: Rozdělení velikostí a fitness funkcí korektních jedinců vygenerovaných inicializací matice náhodnými hodnotami z intervalu (-3 , 3).

Mutace je implementována pomocí zavedení šumu vygenerovaného z Laplaceova rozdělení do určité části matice. Parametry této mutace jsou parametry Laplaceova rozdělení a kolik % hodnot z matice je změněno. Počet vygenerovaných validních jedinců s různým nastavením parametrů se v tomto případě pohyboval od v rozmezí $28-99\%$. U maticí, které zaváděli do reprezentace menší změny byly míra úspěšnosti vyšší viz. tabulka 7.3. Implementace mutace včetně provedených experimentů je ve skriptu dp-filter-mutation.ipynb.

Křížení je implementováno velmi jednoduchou formou. Potomek je vytvořen tak že každá hodnota matice je vybrána buď z jednoho nebo druhého rodiče se stejnou pravděpodobností. Byl proveden experiment, ve kterým bylo vytvořeno 500 nových jedinců pomocí tohoto způsobu křížení. Z těchto jedinců bylo 13.2% validních. Implementace křížení včetně provedených experimentů je ve skriptu dp-filter-crossover.ipynb.

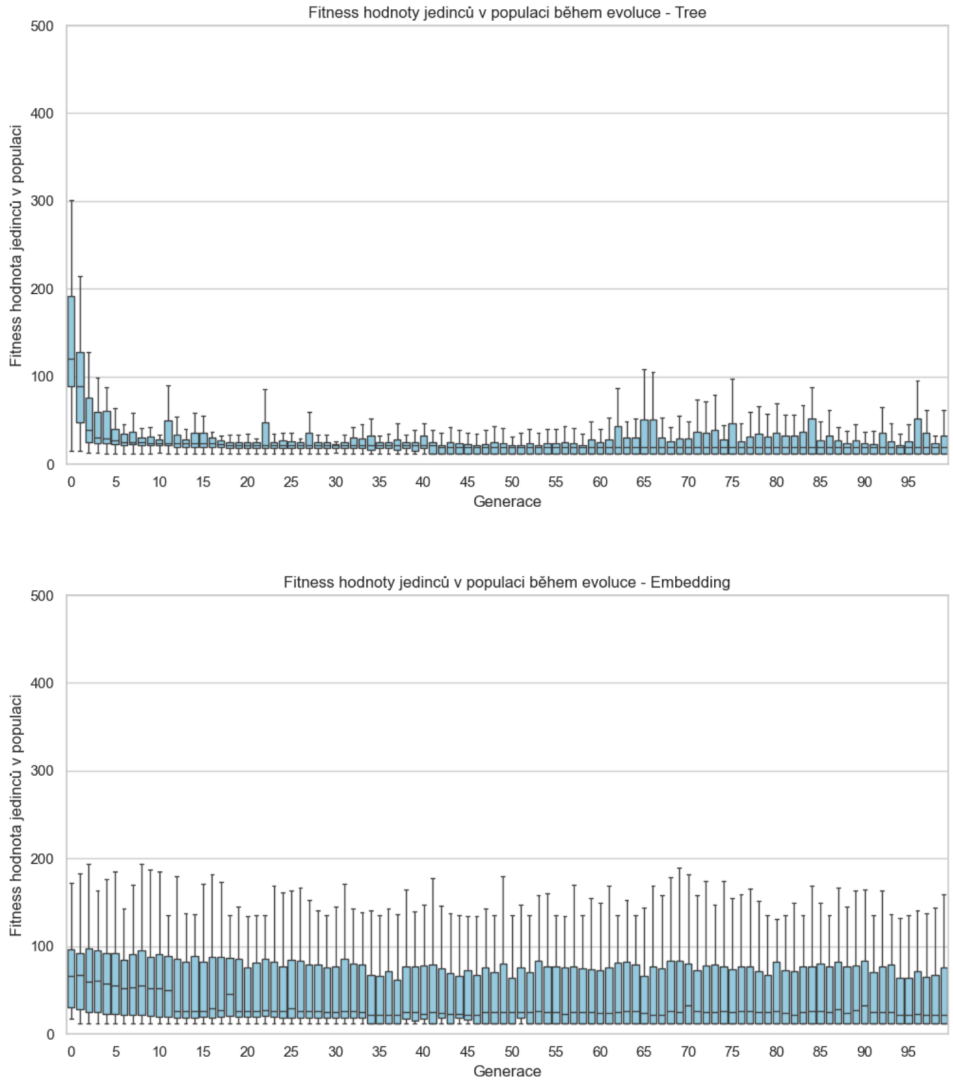
| Číslo pokusu | $\lambda$ | % změněných hodnot | % validních jedinců |
|---|---|---|---|
| 1 | 0.1 | 5 | 99 |
| 2 | 0.2 | 5 | 94 |
| 3 | 0.5 | 5 | 52 |
| 4 | 1 | 5 | 27 |
| 5 | 2 | 5 | 32 |
| 6 | 0.1 | 20 | 96 |
| 7 | 0.2 | 20 | 79 |
| 8 | 0.5 | 20 | 28 |
| 9 | 1 | 20 | 36 |
| 10 | 2 | 20 | 37 |

Tabulka 7.3: Míra úspěšnosti tvorby validního potomka při mutaci s různými parametry. V každém experimentu bylo pomocí mutace vytvořeno 100 jedinců

## 7.6 Evoluce s využitím nové reprezentace

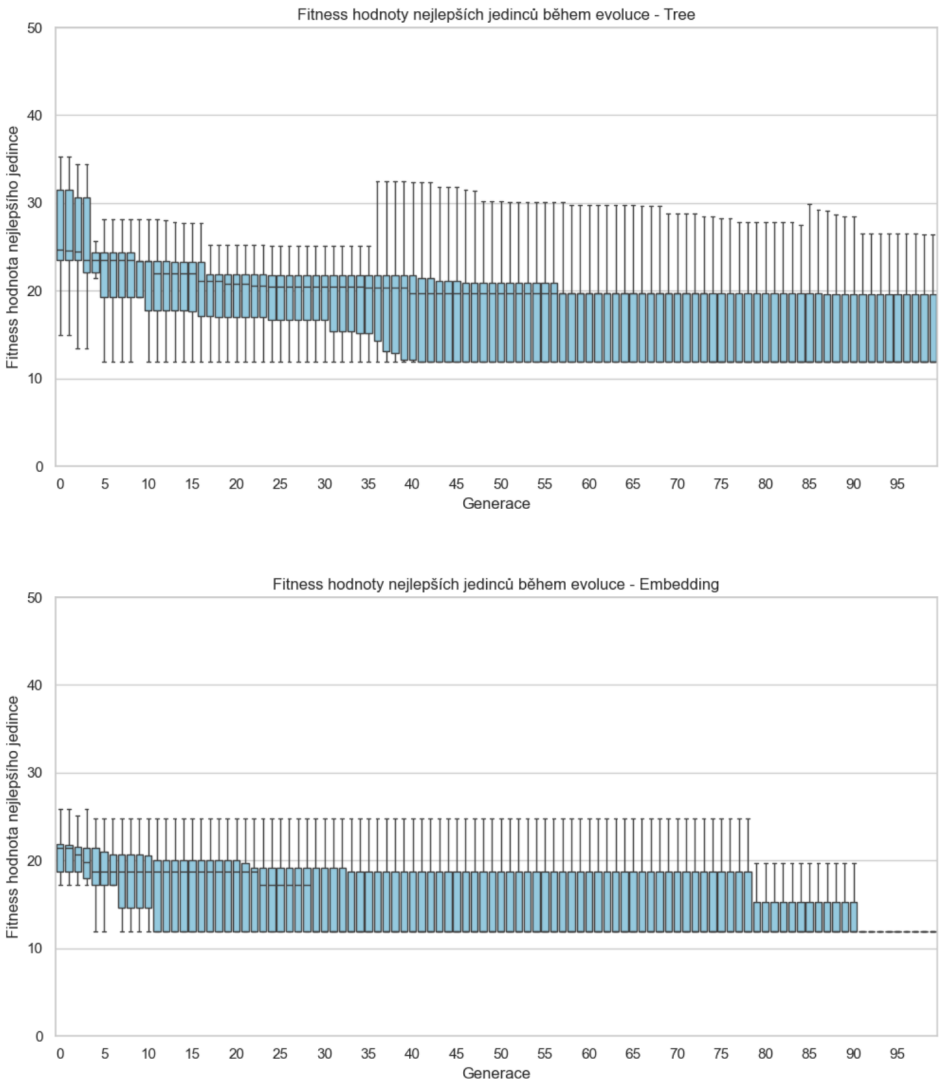
V tomto experimentu je otestováno použití nové reprezentace v algoritmu GP. V rámci experimentu je spuštěno 22 běhů GP s původní reprezentací (GP-Tree) a 22 běhů GP upraveného tak, aby pracoval s novou reprezentací (GP-Emb). Porovnání variant GP je na obrázcích 7.7, 7.8 a 7.9[2]. Parametry běhů GP v obou variantách byly nastaveny na: population_size = 10, number_of_generations = 100, tournament_size = 3, mutation_prob = 0,2, crossover_prob = 0,8, max_depth = 5, function_prob = 0.9, input_prob = 0.7. Algoritmus GP pracující nad nově vytvořenou reprezentací využíval model 2_4_128_model_1_padd_16. Parametry pro mutaci byly nastaveny na $\lambda = 1$, část změněných hodnot v matici = 0,10 (v matici je tedy změněno 10% hodnot). Hodnoty v prvcích matice jedinců byly při inicializace jedince náhodně vybírány z intervalu (-1,1).

[2]V grafech na těchto obrázcích zachycují krabicové grafy data ze všech běhů dané varianty GP v jednotlivých generacích. Odlehlé hodnoty jsou pro lepší viditelnost skryty.



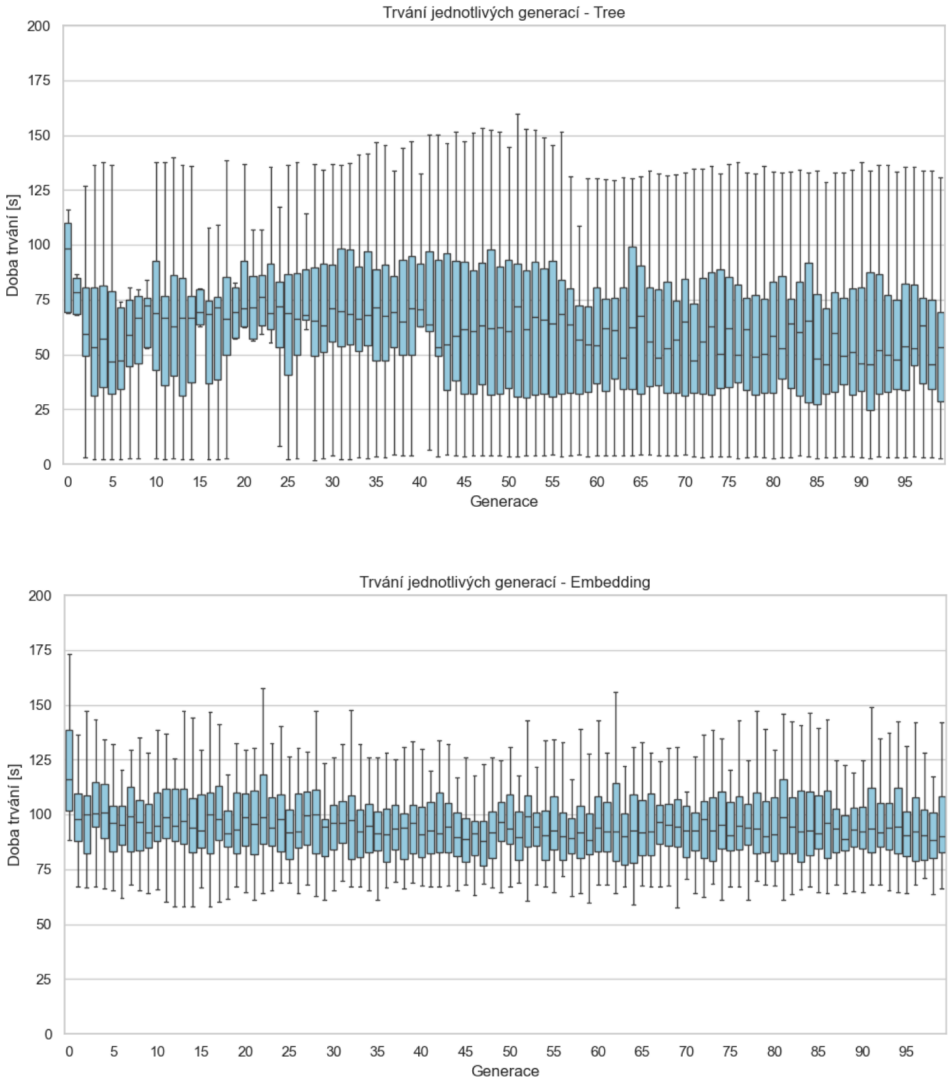

Obrázek 7.7: Porovnání hodnot fitness funkcí v jednotlivých generacích 40 běhů GP využívajícího stromovou reprezentaci a 40 běhů GP upraveného pro běh s novou reprezentací.

V případě GP-Tree je míra explorace v průběhu evoluce menší než v případě GP-Emb. To je pravděpodobně způsobeno velmi rozdílnými operátory mutace a křížení v obou variantách. Inicializace populace v algoritmu GP-Emb vytváří lepší jedince s menší hodnotou fitness.par




Obrázek 7.8: Porovnání hodnot fitness funkcí nejlepších jedinců v jednotlivých generacích 40 běhů GP využívajícího stromovou reprezentaci a 40 běhů GP upraveného pro běh s novou reprezentací.

Z grafů na obrázku 7.8 je zřejmé, že algoritmu GP-Emb konverguje k lepším řešením dříve než algoritmus GP-Tree. Finální kvalita řešení je v obou případech srovnatelná.




Obrázek 7.9: Porovnání doby trvání jednotlivých generacích 40 běhů GP využívajícího stromovou reprezentaci a 40 běhů GP upraveného pro běh s novou reprezentací.

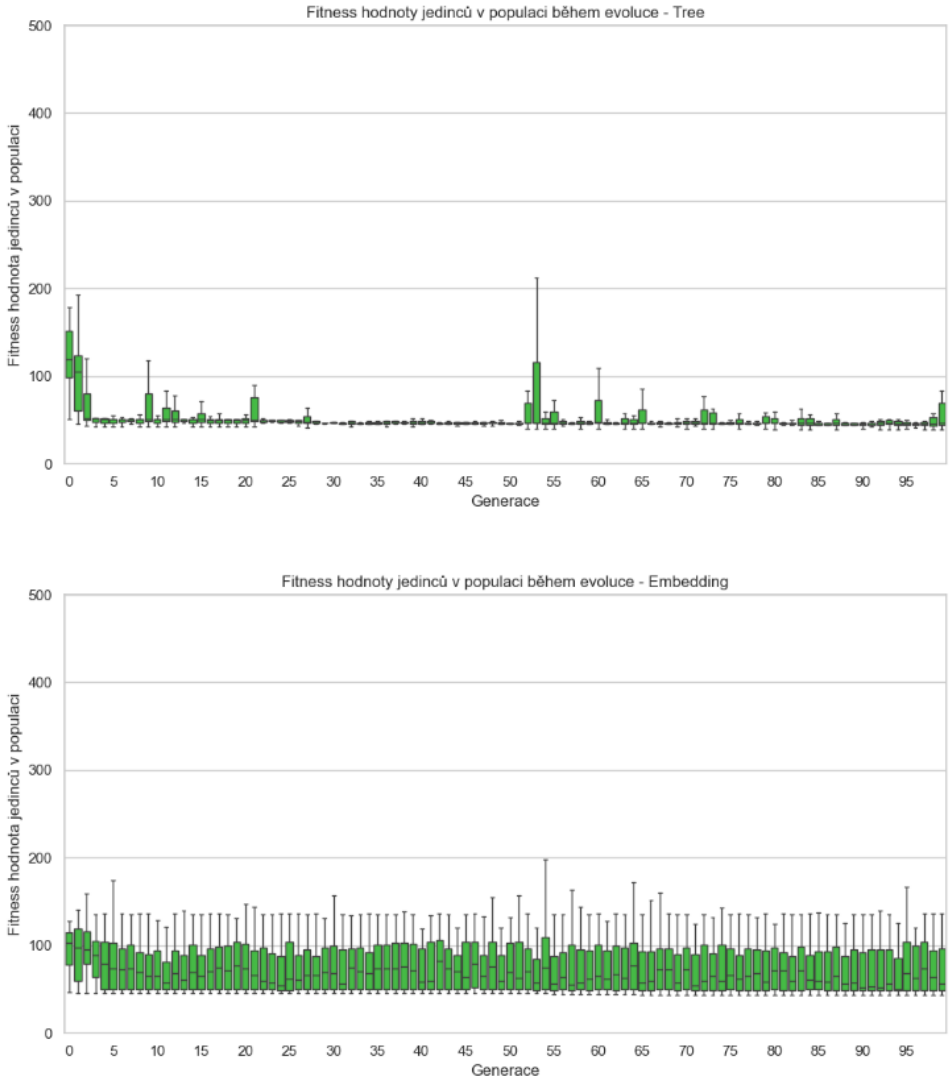
GP, který pracuje s novou reprezentací má o něco delší dobu trvání generace. To je zřejmě způsobeno zejména ze dvou důvodů. Prvním důvodem je potřeba dekódování jedince a test validity při výpočtu jeho fitness hodnoty a vytvoření nového jedince pomocí inicializace, mutace a křížení. Druhým důvodem je, že nová GP pracující s novou reprezentací byl implementován pouze jako rozšíření původního algoritmu GP a tedy není optimalizován pro použití nové reprezentace.

## 7.7 Využití nové reprezentace v GP pro detekci hran

Při učení reprezentace se v modelu nevyužívá fitness hodnota získaná z GP pro odstranění šumu. Využívá se pouze samotná struktura syntaktického stromu. Tato skutečnost by mohla

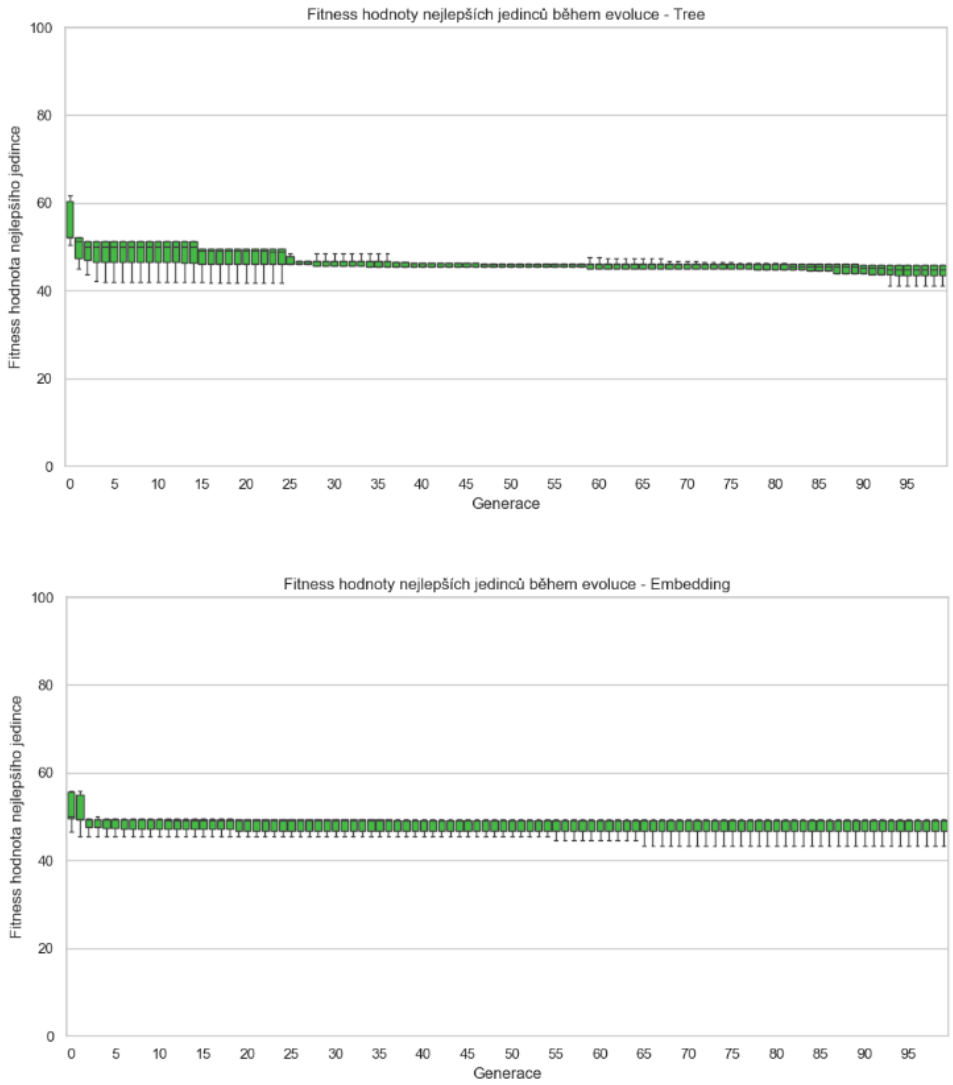
umožnit využívat model opakovaně v GP, které používají stromovou reprezentaci se stejnou primitivní množinou jazyka.

V tomto experimentu je provedeno několik běhů evoluce GP-Tree a GP-Emb upravených tak, aby řešili úlohu detekce hran v obrázku. Algoritmy jsou spuštěny se stejnými parametry jako v experimentu, který je popsán v kapitole 7.6. Cílový obrázek použití při evoluci byl vytvořen pomocí metody Sobel-Fieldman v online nástroji pinetools[3]. Výsledky experimentu jsou zobrazeny na obrázcích 7.10, 7.11, 7.12 a 7.13.

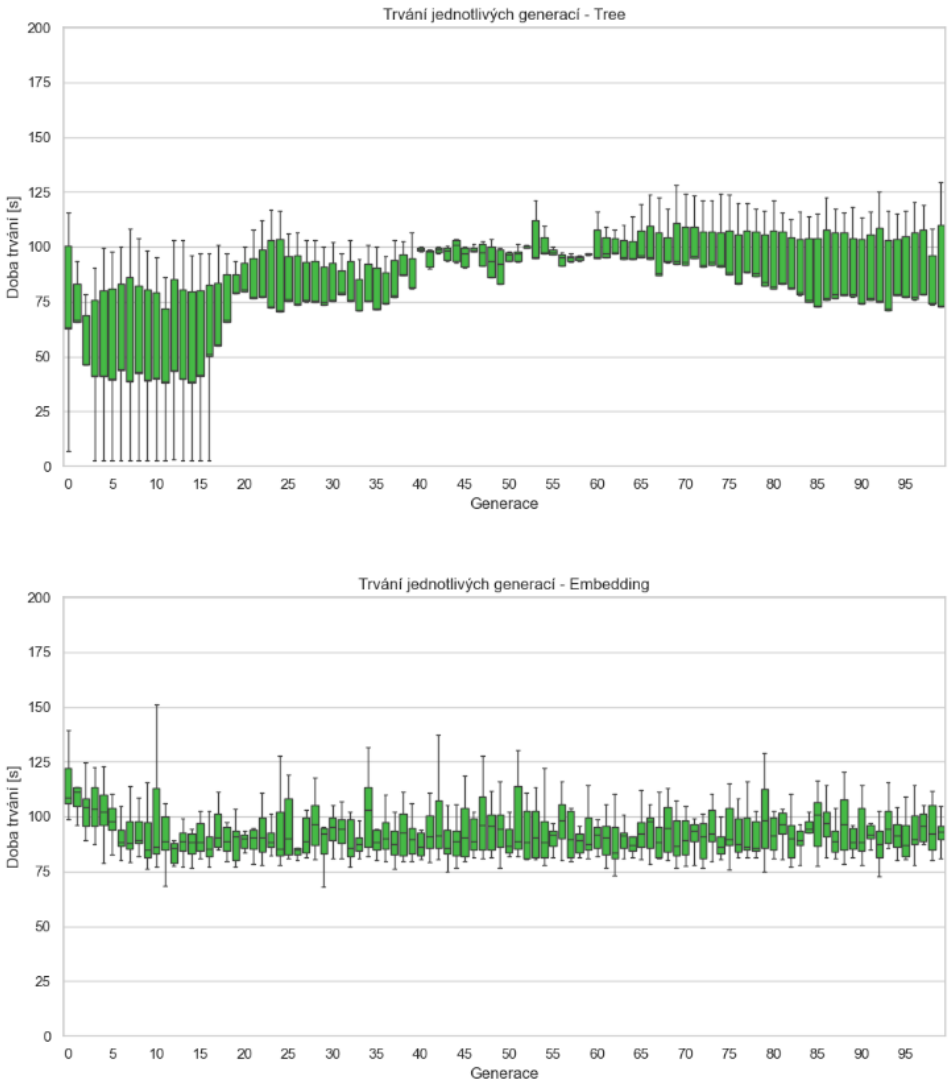



Obrázek 7.10: Porovnání hodnot fitness funkcí v jednotlivých generacích 10 běhů GP využívajícího stromovou reprezentaci a 10 běhů GP upraveného pro běh s novou reprezentací v úloze detekce hran.

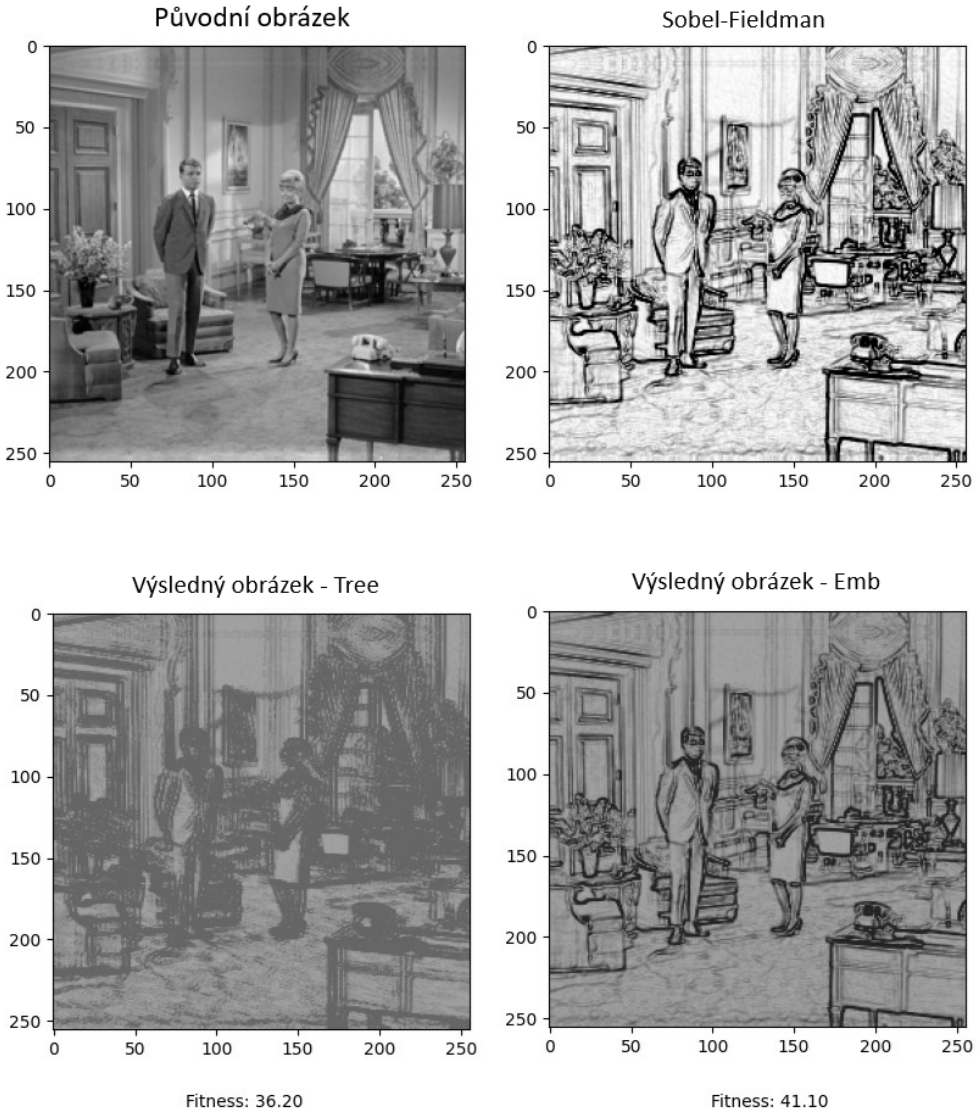
[3]https://pinetools.com/image-edge-detection




Obrázek 7.11: Porovnání hodnot fitness funkcí nejlepších jedinců v jednotlivých generacích 10 běhů GP využívajícího stromovou reprezentaci a 10 běhů GP upraveného pro běh s novou reprezentací v úloze detekce hran.




Obrázek 7.12: Porovnání doby trvání jednotlivých generacích 10 běhů GP využívajícího stromovou reprezentaci a 10 běhů GP upraveného pro běh s novou reprezentací v úloze detekce hran.




Obrázek 7.13: Jeden z testovací obrázků o velikosti $256 \times 256$ zpracovaný pomocí filtru z nejlepších běhů GP-Emb a GP-Tree.

Na obrázku 7.13 jsou zobrazeny výsledné obrázky, na které byly aplikovány získané filtry. Je nutné poznamenat, že z důvodu výpočetní náročnosti, která je způsobena maximální velikostí filtru jsou jednotlivé běhy provedeny pouze na 100 generací a tudíž mají získané finální výsledky stále relativně vysokou fitness hodnotu fitness funkce. Z výsledků získaných během experimentu je zjevné, že novou reprezentaci je možné použít i v GP s jiným typem úlohy, než na které byla reprezentace trénována. Tato skutečnost částečně nahrazuje nevýhodu spojenou s přidanou režií, která je nutná pro trénování Transformeru.

# Kapitola 8

# Závěr

Cílem této práce bylo seznámit se s metodami strojového učení používanými v oblasti automatického návrhu reprezentace, se zaměřením na hluboké neuronové sítě, a navrhnout automatizovanou metodu tvorby této reprezentace. Nová reprezentace je určena pro algoritmus genetického programování (GP), který řeší úlohy v oblasti zpracování obrazu. Konkrétně byla zvolena úloha odstranění impulsního šumu pomocí GP. Dalším cílem práce byla implementace navržené metody pro automatické získání reprezentace a ověření její funkčnosti a chování.

Zvolená reprezentace je založena na neuronové síti typu transformer. Tato reprezentace vychází z článku [7], ale byla upravena tak, aby umožňovala zpětnou transformaci nově vzniklé reprezentace do původní reprezentace. Návrh řešení pro tvorbu této nové reprezentace z původní reprezentace syntaktického stromu je popsán v kapitole 5. Tato kapitola rovněž popisuje architekturu modelu pro tvorbu reprezentace, získání datové sady pro trénování modelu a algoritmus GP pro odstranění impulsního šumu.

Implementace nástrojů, algoritmů a pomocných funkcí, které jsou následně využity v jednotlivých experimentech s novou reprezentací, jsou popsány v kapitole 6. Veškerý zdrojový kód, který vznikl v rámci této práce, byl spouštěn na platformě Kaggle, která umožňuje spouštění skriptů s využitím GPU akcelerátorů.

V poslední části práce, popsané v kapitole 7, jsou uvedeny provedené experimenty. Byl vytvořen model pro automatickou tvorbu reprezentace, který původně sloužil pro překlad z portugalštiny do angličtiny. Z tohoto základního upraveného modelu bylo poté vytvořeno přes 20 variant modelů, které byly natrénovány na několika jazykových verzích datasetu vzniklého v jednom z experimentů. Enkodérové části modelů byly nepřímo vyhodnoceny pomocí Spearmanovy korelace mezi editační vzdáleností dvou stromů v původní reprezentaci a cosinovou vzdáleností mezi stejnými stromy převedenými do nové reprezentace. Dekodérové části byly vyhodnoceny na základě přesnosti, s jakou dokážou převést novou reprezentaci zpět do původní. V dalších experimentech byl v nové reprezentaci navržen a vyhodnocen způsob inicializace nového jedince a genetické operátory křížení a mutace, které byly poté použity pro upravený algoritmus GP pracující s novou reprezentací.

Přínos této práce vidím ve funkční části dekodéru, která v původním článku [7] není vůbec využita. Ta umožňuje využít novou reprezentaci přímo v algoritmu GP. Finální ověření funkčnosti nové reprezentace přímo v algoritmu GP otevírá celou řadu způsobů, jak na tuto práci navázat.

# Literatura


[1] *Kaggle: Your Home for Data Science*. Google, 2017 [cit. 07.5.2024]. Platforma pro datové vědce a strojové učení. Dostupné z: https://www.kaggle.com.

[2] Bartz Beielstein, T., Branke, J., Mehnen, J. a Mersmann, O. Evolutionary algorithms. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*. květen 2014, sv. 4, č. 3, s. 178–195, [cit. 29.3.2024]. DOI: 10.1002/widm.1124. ISSN 1942-4795.

[3] Bengio, Y., Courville, A. a Vincent, P. Representation Learning: A Review and New Perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.* USA: IEEE Computer Society. aug 2013, sv. 35, č. 8, s. 1798–1828, [cit. 19.1.2024]. DOI: 10.1109/TPAMI.2013.50. ISSN 0162-8828. Dostupné z: https://doi.org/10.1109/TPAMI.2013.50.

[4] Boateng, K., Weyori, B. a Laar, D. Improving the Effectiveness of the Median Filter. *International Journal of Electronics and Communication Engineering*. Leden 2012, sv. 5, s. 85–97, [cit. 23.1.2024]. ISSN 0974-2166.

[5] Boncelet, C. Chapter 7 - Image Noise Models. In: Bovik, A., ed. *The Essential Guide to Image Processing*. Boston: Academic Press, 2009, s. 143–167 [cit. 23.1.2024]. DOI: https://doi.org/10.1016/B978-0-12-374457-9.00007-X. ISBN 978-0-12-374457-9. Dostupné z: https://www.sciencedirect.com/science/article/pii/B978012374457900007X.

[6] Brilliant.org. *Multivariate Normal Distribution — Brilliant Wiki*. 2024 [cit. 23.1.2024]. Dostupné z: https://brilliant.org/wiki/multivariate-normal-distribution/.

[7] 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 [cit. 23.1.2024]. GECCO '23. DOI: 10.1145/3583131.3590423. ISBN 9798400701191. Dostupné z: https://doi.org/10.1145/3583131.3590423.

[8] Gurney, K. *An Introduction to Neural Networks*. USA: Taylor & Francis, Inc., 1997 [cit. 26.4.2024]. ISBN 1857286731. Dostupné z: https://doi.org/10.1201/9781315273570.

[9] Haykin, S. *Neural Networks: A Comprehensive Foundation (3rd Edition)*. USA: Prentice-Hall, Inc., 2007 [cit. 26.4.2024]. ISBN 0131471392.

[10] Hynek, J. *Genetické algoritmy a genetické programování*. Grada, 2008 [cit. 29.3.2024]. Prvodce (Grada). ISBN 9788024726953. Dostupné z: https://books.google.cz/books?id=RFlXh5Fd_7MC.

[11] Miller, J. Cartesian Genetic Programming. In: *Cartesian Genetic Programming*. červen 2011, sv. 43, s. 17–34 [cit. 23.1.2024]. Natural Computing Series. DOI: 10.1007/978-3-642-17310-3. ISBN 978-3-642-17309-7. Dostupné z: https://www.fit.vut.cz/research/publication/9771.

[12] Miller, J. Introduction to Evolutionary Computation and Genetic Programming. In: *Cartesian Genetic Programming*. červen 2011, sv. 43, s. 1–16 [cit. 23.1.2024]. Natural Computing Series. DOI: 10.1007/978-3-642-17310-3. ISBN 978-3-642-17309-7. Dostupné z: https://www.fit.vut.cz/research/publication/9771.

[13] Moore, N., Leeser, M. a King, L. S. *Adaptable Sliding Window Operations with CUDA*. 2024 [cit. 24.1.2024]. Dostupné z: https://coe.northeastern.edu/Research/rcl/projects/adaptableCUDA.php.

[14] Morales, F. *The Normal Distribution for Data Scientists* [Medium]. January 2023 [cit. 23.1.2024]. Available online. Dostupné z: https://medium.com/analytics-vidhya/the-normal-distribution-for-data-scientists-6de041a01cb9.

[15] Munro, P. Backpropagation. In: Sammut, C. a Webb, G. I., ed. *Encyclopedia of Machine Learning*. Boston, MA: Springer US, 2010, s. 73–73 [cit. 26.4.2024]. DOI: 10.1007/978-0-387-30164-8_51. ISBN 978-0-387-30164-8. Dostupné z: https://doi.org/10.1007/978-0-387-30164-8_51.

[16] O'Shea, K. a Nash, R. An Introduction to Convolutional Neural Networks. *ArXiv e-prints*. Listopad 2015, [cit. 10.5.2024]. DOI: https://doi.org/10.48550/arXiv.1511.08458.

[17] Poli, R., Langdon, W. B. a McPhee, N. F. *A Field Guide to Genetic Programming*. Lulu Enterprises, UK Ltd, 2008 [cit. 29.3.2024]. ISBN 978-1-4092-0073-4.

[18] Ramík, J. Soft Computing: Overview and Recent Developments in Fuzzy Optimization. In:. 2001 [cit. 29.3.2024]. Dostupné z: https://api.semanticscholar.org/CorpusID:123643581.

[19] Rudolph, G. Handbook of Natural Computing. In: Grzegorz Rozenberg, J. N. K., ed. Springer Berlin, Heidelberg, 2012, kap. Evolutionary Strategies [cit. 29.4.2024]. DOI: 10.1007/978-3-540-92910-9. ISBN 978-3-540-92909-3. Dostupné z: https://doi.org/10.1007/978-3-540-92910-9.

[20] Sarker, I. H. Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions. *SN Computer Science*. 2021, sv. 2, č. 6, s. 420, [cit. 19.1.2024]. DOI: 10.1007/s42979-021-00815-1. ISSN 2661-8907. Dostupné z: https://doi.org/10.1007/s42979-021-00815-1.

[21] Sekanina, L., Harding, L. S., Banzhaf, W. a Kowaliw, T. Image Processing and CGP. In: *Cartesian Genetic Programming*. Springer Verlag, 2011, s. 181–215 [cit. 23.1.2024]. Natural Computing Series. ISBN 978-3-642-17309-7. Dostupné z: https://www.fit.vut.cz/research/publication/9771.

[22] Sloss, A. N. a Gustafson, S. Genetic Programming Theory and Practice XVII. In: Banzhaf, W., Goodman, E., Sheneman, L., Leonardo, T. a Worzel, B., ed. Cham: Springer International Publishing, 2020, kap. 2019 Evolutionary Algorithms Review, s. 307–344 [cit. 29.3.2024]. DOI: 10.1007/978-3-030-39958-0_16. ISBN 978-3-030-39958-0.

[23] Svozil, D., Kvasnicka, V. a Pospichal, J. Introduction to multi-layer feed-forward neural networks. *Chemometrics and Intelligent Laboratory Systems*. 1997, sv. 39, č. 1, s. 43–62, [cit. 26.4.2024]. DOI: https://doi.org/10.1016/S0169-7439(97)00061-0. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0169743997000610.

[24] TensorFlow Team. *Transformer model for language understanding*. 2024 [cit. 07.5.2024]. Accessed: 2024-05-07. Dostupné z: https://www.tensorflow.org/text/tutorials/transformer.

[25] 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 [cit. 26.4.2024]. NIPS'17. DOI: 10.5555/3295222.3295349. ISBN 9781510860964. Dostupné z: https://dl.acm.org/doi/10.5555/3295222.3295349.

[26] Ústav zdravotnických informací a statistiky ČR. *Neurony: základní stavební jednotky nervového systému*. 2024 [cit. 1.5.2024]. Dostupné z: https://www.nzip.cz/clanek/1399-neurony-zakladni-stavebni-jednotky-nervoveho-systemu#:~:text=Neurony%20jsou%20vysoce%20specializovan%C3%A9%20bu%C5%88ky,a%20axon%20(nervov%C3%A9%20vl%C3%A1kno).

# Příloha A

# Obsah paměťového média

/ **src**
/ / **kaggle**
/ / **tests**
/ / **packages**
/ / / **GP**
/ / / **Transformer**
/ **data**
/ **technicka_zprava**
/ **README.md**