VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

EVOLUČNÍ NÁVRH NELINEÁRNÍCH FUNKCÍ PRO KONVOLUČNÍ NEURONOVÉ SÍTĚ

EVOLUTIONARY DESIGN OF NONLINEAR FUNCTIONS

FOR CONVOLUTIONAL NEURAL NETWORKS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. MARTIN HLADIŠ**
AUTHOR

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

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání diplomové práce

154436

Ústav:	Ústav počítačových systémů (UPSY)
Student:	**Hladiš Martin, Bc.**
Program:	Informační technologie a umělá inteligence
Specializace:	Strojové učení
Název:	**Evoluční návrh nelineárních funkcí pro konvoluční neuronové sítě**
Kategorie:	Umělá inteligence
Akademický rok:	2023/24

Zadání:

1. Seznamte se s konvolučními neuronovými sítěmi (CNN), evolučními algoritmy (EA) a možnostmi využití EA pro návrh a optimalizaci CNN. Zaměřte se na optimalizaci nelineárních aktivačních funkcí.
2. Navrhněte způsob využití EA pro automatizovaný návrh aktivačních funkcí pro neuronové sítě, např. s cílem zlepšit kvalitu klasifikace pro danou datovou sadu.
3. Implementujte EA pro automatizovaný návrh aktivačních funkcí.
4. Na zvolených modelech CNN a vhodných datových sadách ověřte funkčnost navržené metody.
5. Zhodnoťte dosažené výsledky a diskutujte možnosti pokračování projektu.

Literatura:

- Dle pokynů vedoucího.

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 diplomové práce je návrh a implementace programu pro automatizovaný návrh nelineárních aktivačních funkcí pro konvoluční neuronové sítě (CNN) s využitím evolučních algoritmů. Využití automatického návrhu poskytuje nezávislý pohled na systematické prozkoumání širokého spektra aktivačních funkcí a identifikaci těch nejlepších. Metoda zvolená v práci pro automatický návrh je formou evolučních algoritmů nazývanou jako kartézské genetické programování, které pro zakódování řešení využívá grafovou reprezentaci. Tato technika umožňuje definici sady matematických primitiv, která definuje prohledávací prostor, a tak jednoduše parametrizuje návrh. Implementovaný přístup byl otestován na několika různých architekturách a datasetech (LeNet-5 & MNIST, ResNet-10 & FashionMNIST, WRN-40-4 & CIFAR-10). Experimenty dokázaly, že přístup dokáže nalézt aktivační funkce, které statisticky zlepšují přesnost CNN oproti běžně využívané funkci ReLU.

Abstract

The aim of this thesis is to design and implement a program for automated design of nonlinear activation functions for convolutional neural networks (CNN) using evolutionary algorithms. The use of automated design provides an independent view to systematically explore a wide range of activation functions and identify the best ones. The method for automatic design chosen in this thesis is a form of evolutionary algorithms referred to as Cartesian genetic programming, which uses a graph representation to encode the solution. This technique allows for the definition of a set of mathematical primitives that define the search space, and thus simply parameterize the design. The implemented approach has been tested on several different architectures and datasets (LeNet-5 & MNIST, ResNet-10 & FashionMNIST, WRN-40-4 & CIFAR-10). Experiments have shown that the approach can find activation functions that statistically improve the accuracy of the architecture over the commonly used ReLU function.

Klíčová slova

Aktivační funkce, Evoluční algoritmy, Kartézské genetické programování, AutoML, Hluboké učení, Konvoluční neuronové sítě

Keywords

Activation Functions, Evolutionary algorithms, Cartesian Genetic Programming, AutoML, Deep Learning, Convolutional neural nets

Citace

HLADIŠ, Martin. *Evoluční návrh nelineárních funkcí pro konvoluční neuronové sítě*. Brno, 2024. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. LUKÁŠ SEKANINA, Ph.D.

Evoluční návrh nelineárních funkcí pro konvoluční neuronové sítě

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.

.....
Martin Hladiš
15. května 2024

Poděkování

Rád bych poděkoval svému vedoucímu práce panu profesoru Ing. Lukáši Sekaninovi, Ph.D. za odborné vedení, konzultace, čas a cenné rady při tvorbě této práce. Dále bych rád poděkoval docentu Ing. Jiřímu Jarošovi, Ph.D. za zprostředkování přístupu a výpočetních zdrojů pro práci. Dík patří také mé rodině a blízkým za podporu při studiu a při tvoření této práce. Tato práce byla podpořena Ministerstvem školství, mládeže a tělovýchovy České republiky prostřednictvím e-INFRA CZ (ID:90140).

Obsah

Kapitola 1

Úvod

Umělá inteligence je jedním z nejrychleji rostoucích odvětví v oblasti informatiky a stává se nedílnou součástí našeho každodenního života. Značný pokrok v této oblasti je z velké části dán rapidním nárůstem výpočetního výkonu, a díky tomuto vzestupu jsme schopni řešit problémy, které byly v době vzniku konceptu umělé inteligence v padesátých letech minulého století téměř nepředstavitelné.

Tento pokrok je také částečně dán i vývojem nových technologií, jako například konvolučních neuronových sítí (CNN), které v současné době dominují oblasti rozpoznávání obrazu. Jejich popularita vysoce vzrostla po úspěchu sítě AlexNet [18] v soutěži ImageNet 2012 Challenge, kde se tato síť s výrazným odstupem od konkurence umístila na prvním místě. Jedním z hlavních důvodů tohoto úspěchu byla speciální inovativní architektura, která dokáže z obrazových dat vyextrahovat potřebné informace k úspěšnému rozpoznávání obrazu.

Nemálo výzkumníků se od té doby zabývalo způsoby, jak tuto technologii dále vylepšit. Jednou z možností je vytvoření architektury CNN, která by měla lepší vlastnosti a tím i lepší schopnost pro řešení požadované úlohy. Tato činnost však vyžaduje velké množství oborových znalostí, zkušeností a také značný počet pokusů. Tyto pokusy jsou značně časově náročné, ne vždy efektivní a bez záruky výsledku. Z tohoto důvodu je nutné hledat další cesty, jak k tomuto problému přistupovat a zefektivnit průběh designu. Automatický návrh se zdá být jako správná cesta pro nezaujatý pohled na systematické prozkoumávání širokého spektra možností z prohledávacího prostoru, jež vyhovuje vlastnostem tohoto problému.

I tato diplomová práce se zabývá možnostmi zlepšení architektury CNN. Zaměřuje se na oblast automatického návrhu nelineárních aktivačních funkcí, které mají za úkol plně využít architekturu CNN a vylepšit tak přesnost při klasifikaci obrazových dat oproti standardizované aktivační funkci ReLU.

Tato práce se zaměřuje na metody, které se v určitých ohledech chovají jako černá skříňka. Umožňují převést zásadní část řešení problému do podoby algoritmu, který problém řeší prostřednictvím pokusů a omylů, přičemž čerpají inspiraci z biologické evoluce. Tyto algoritmy, které fungují tímto způsobem, se nazývají evoluční. Speciálně pro návrh aktivačních funkcí byla vybrána metoda kartézského genetického programování.

Diplomovová práce je strukturována následovně: v kapitole 2 jsou rozebrány základy a hlavní myšlenky evolučních algoritmů. Dále je zde zaměřeno na princip kartézského genetického programování. Kapitola 3 se zabývá popisem umělých neuronových sítí. Značná část této kapitoly je pak zaměřena na konvoluční neuronové sítě spolu se seznámením s využitými architekturami CNN a testovanými datasety provedených experimentů. Teoretickou část ukončuje kapitola 4, která spojuje principy neuronových sítí s evolučními algoritmy, ji-

nak také nazývanými jako neuroevoluce. V této kapitole se nachází popis evolučního návrhu různých částí umělých neuronových sítí, současně je zde však kladen důraz na popis návrhu aktivačních funkcí. V kapitole číslo 5 se nachází nastínění návrhu aktivačních funkcí prezentovaných v této práci včetně popisu využití kartézského genetického programování a hlavní evoluční smyčky celého programu. Kapitola 6 shrnuje implementaci a návod pro spuštění implementovaného programu. V kapitole 7 se nachází popis provedených experimentů spolu s jejich vyhodnocením a dále jsou zde nastíněny možnosti případného pokračování práce. Práce je zakončena kapitolou 8, kde jsou shrnuty dosažené výsledky.

Kapitola 2

Evoluční algoritmy

Každý den se inženýři v různých oblastech své práce setkávají se složitě řešitelnými a výpočetně náročnými problémy. Mnohé problémy jde definovat jako optimalizační úlohu, kde cílem je nalezení řešení x z množiny všech kandidátních řešení S, které by nejlépe splňovalo zadané požadavky a omezující podmínky. Splnění požadavků definuje tzv. účelová funkce $f(x) : S \to \mathbb{R}$, kterou se řešení snaží maximalizovat (nebo minimalizovat).

V reálných problémech jsou účelové funkce nespojité, nelineární, nekonvexní, multimodální a dimenzionálně závislé na složitosti problému. Díky těmto vlastnostem neumožňuje řešit problémy běžnými deterministickými algoritmy, které by za těchto podmínek běžely příliš dlouhou dobu a s neurčitými výsledky.

V takto složitých podmínkách reálného světa se výzkumníkům osvědčilo využití evolučních algoritmů (zkráceně EA). Jedná se stochastické meta-heuristické metody, které nezaručují optimální řešení, avšak dokáží překonat lokální minima, a tím často zajistit vhodný výsledek dostačující pro použití v praxi.

2.1 Inspirace v biologické evoluci

Evoluční algoritmy jsou inspirovány biologickým procesem evoluce, který popsal britský přírodovědec Charles Darwin ve své knize *O původu druhů* [8]. Hlavní myšlenkou je proces přirozené selekce, pomocí které se populace živých organismů adaptuje a mění. Jedinci v populaci jsou přirozeně variabilní, což znamená, že se všichni v něčem liší. Tato variabilita znamená, že někteří jedinci mají vlastnosti, které jsou pro dané prostředí vhodnější než jiné. Schopnější jedinci mají větší šanci pro nalezení vhodného partnera a zplození většího množství potomků, tzv. *přežití nejzdatnějších*. Adaptivní vlastnosti rodičů se přenáší na nové pokolení a postupem času se tyto výhodné znaky stávají v populaci běžnějšími. Tímto procesem přírodního výběru se výhodné vlastnosti přenášejí do další generace.

Biologická evoluce může být chápána jako optimalizační proces, nalezení nejlépe přizpůsobeného organismu v podmínkách daných selekčním tlakem. Není proto divu, že se tímto procesem inspirovalo celé odvětví počítačové vědy, když sílu tohoto procesu je možno spatřit ve všech místech světa.

První pokusy využít darwinovské principy k automatizovanému řešení problémů se datují do 50. let 20. století zejména pracemi H. J. Bremermanna [4], R. M. Friedberga [12]. V 60. letech nezávisle na sobě vznikly tři rozdílné směry evolučních algoritmů, které se nazývají **evoluční programování**, **genetické algoritmy**, **evoluční strategie**. Začátkem 90. let k nim přibyl čtvrtý směr, a to **genetické programovaní**, a později vznikly další varianty, např. **diferenciální evoluce** apod.

2.2 Základní principy a komponenty EA

Všechny zmíněné varianty mají společné rysy, které převzaly z biologické evoluce. Mezi hlavní principy patří práce s populací jedinců, kde jedinec představuje zakódované řešení problému. Populace je iterativně šlechtěna pomocí biologií inspirovaných operátorů k nalezení nejlepšího řešení. Mezi ně patří výběr rodičů (selekce), křížení rodičů a náhodná mutace jedinců. U jedinců je vyhodnocována jejich kvalita pomocí tzv. **fitness** funkce, lépe ohodnocené řešení potom zvýhodněno u výběru pro další generaci a tím roste kvalita celé populace řešení [5, 9].

Obecný evoluční algoritmus je popsán pseudokódem 1. Každý evoluční algoritmus je složen ze základních komponent a v následujících podkapitolách budou vysvětleny jednotlivé kroky a prvky algoritmu.

Algorithm 1: Obecný evoluční algoritmus

```
INICIALIZACE populace náhodnými kandidátními řešeními;
EVALUACE kandidátních řešení;
while !podmínka ukončení do
    SELEKCE rodičů;
    KŘÍŽENÍ párů rodičů;
    MUTACE vzniklých potomků;
    SELEKCE jedinců pro další iteraci;
    EVALUACE nových kandidátních řešení;
end
```

2.2.1 Reprezentace problému

Volba reprezentace kandidátního řešení je první a jeden z nejdůležitějších kroků při využití EA algoritmů. Jeho specifikace propojuje kontext problému s prohledávacím prostorem kandidátních řešení a definuje jeho variabilitu. Vytváří se mapování mezi objekty reálného světa nazývané jako **fenotyp** na jejich reprezentaci v EA algoritmu. Zakódování v prostředí EA se běžně nazývá **genotyp**, nebo **chromozom** a jeho jednotlivé části se nazývají **geny**. Výběr vhodné reprezentace ovlivňuje složitost a celkovou účinnost algoritmu, a proto je vhodné této činnosti věnovat dostatek času při analýze a řešení problému.

V EA algoritmech se využívá mnoho různých druhů reprezentací. Nejznámější a nejvíce využívané jsou:

- Binární reprezentace – chromozom nabývá reprezentace posloupnosti bitů a jako genetické operátory jsou využívány bitové operace.
- Reprezentace pomocí reálných čísel – chromozom se skládá z vektoru reálných čísel. Genetické operátory jsou prováděny pomocí matematických operací s reálnými čísly.
- Stromová reprezentace – chromozom je reprezentován hiearchickou stromovou strukturou a jako genetické operátory jsou využívány operace nad uzly stromu nebo nad celými podstromy.

2.2.2 Populace

Populaci v evolučním algoritmu tvoří všechny kandidátní řešení aktuální generace. Jedinci v populaci představují statické objekty, které se nemění, ani neadaptují. Schopnost změny připadá k populaci a k jejímu iteračnímu vývoji, neboli schopnosti aktuálního řešení vytvářet potomky.

Velikost populace se v EA algoritmu ve většině případech nemění a zůstává stejná během evolučního prohledávání. Toto vytváří limitované zdroje a selekční tlak na populaci. Velikost populace ovlivňuje rychlost konvergence algoritmu (vyhodnocení fitness funkce u každého jedince) a také kvalitu jeho řešení. Při zvolení příliš malé velikosti populace nemusí populace dosahovat dostatečné **diverzity**, která by vedla k dosažení globálního minima.

S populací v EA je spojena tzv. **inicializace** populace, což označuje vytvoření první iterace jedinců. Ve většině případů se první generace skládá z náhodně vytvořených genotypů, avšak může se zvolit metoda zakódování aktuálně užívaných řešení (větší fitness populace). Tento postup zavádí do algoritmu další informace o doméně problému, které mohou vést k rychlejší konvergenci.

2.2.3 Mechanismus selekce

Jedná se o proces výběru jedinců, který má za úkol rozdělit jedince na základě jejich kvality a umožnit kvalitnějším jedincům stát se rodiči další generace. Výběrem lepších jedinců se na potomky (angl. *offspring*) přenáší lepší vlastnosti a to vede k posouvání kvality celé populace.

Pro výběr rodičů se většinou používají pravděpodobnostní metody tak, že lépe ohodnocení jedinci mají větší pravděpodobnost výběru než ti hůře ohodnocení. Tímto se zajistí, aby se do výběru dostali i hůře ohodnocení jedinci, kteří můžou správnou kombinací svého genotypu pomoci populaci dosáhnout lepších výsledků. Některé z nejvíce používaných metod pro výběr rodičů jsou uvedené níže v této sekci.

Další druh selekce, který v EA algoritmech probíhá, je tzv. **výběr přeživšího**. Jedná se o výběr jedinců, kteří se dostanou do další generace. Tyto metody jsou většinou založeny na věku jedince (počet generací jedince účastnícího se populace), nebo na základě fitness hodnoty.

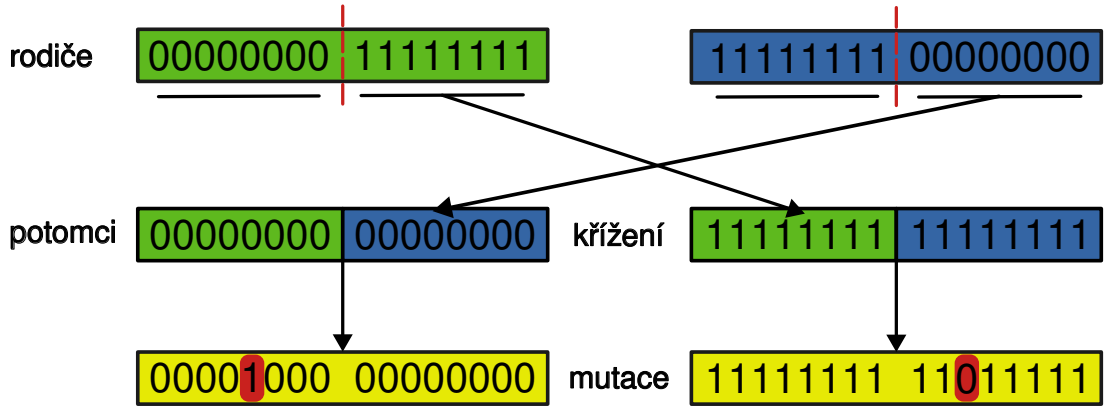
Při využití věku jedince je nejvíce využívaná tzv. **generační výměna**. Nová populace u této metody obsahuje pouze jedince z množiny potomků. Metoda, která umožňuje průchod K jedinců z minulé generace, se nazývá **postupná výměna** (angl. *steady-state selection*). Pro výběr jedinců se mohou používat metody selekce rodičů. V metodách založených na fitness hodnotě se většinou do další generace dostávají nejlépe ohodnocení jedinci nebo jejich skupina (**elitizmus**). Tím je zajištěno, že v další generaci se nachází alespoň některé kvalitní řešení. V praxi se využívá kombinace těchto způsobů nebo některé další alternativy jako například **turnaj**, **ruleta**, které umožňují i hůře ohodnoceným jedincům účasnit se vývoje populace.

2.2.4 Křížení a mutace

Křížení a mutace jsou dva genetické operátory, které slouží k vytváření nových jedinců z předchozí generace. Využití obou genetických operátorů v EA je závislé na typu řešené úlohy, jelikož těmito operátory upravujeme míru průzkumu algoritmu. Proto není vhodné v některých případech jeden z operátorů použít. Dále je nutné dodat, že oba genetické operátory úzce souvisí z druhem zvolené reprezentace problému v genotypu.

Využití operátoru **křížení** vychází z myšlenky, že kombinací dvou jedinců s požadovanými vlastnostmi vznikne potomek, který tyto vlastnosti pozitivně kombinuje. V evolučních algoritmech operátory křížení využívají stochastického charakteru, jelikož neznáme žádný správný postup k určení kombinace genotypu rodičů, které by měly vést řešení ke globálnímu minimu.

Operátor **mutace** s určitou malou pravděpodobností modifikuje genotyp potomka za účelem zvětšení diverzity populace a vytvoření vlastností, kterými rodiče nedisponovali. Mutace umožňuje dostat řešení z lokálních minim. Je však důležité, aby síla mutace na genotyp nebyla příliš velká, což by vedlo degradování EA k náhodnému prohledávání.



Obrázek 2.1: Ukázka reprezentace jedinců v binární podobě spolu s jednobodovým křížením a provedení mutace na nově vzniklých potomcích.

2.2.5 Ukončující podmínka

Posledním prvkem v EA je ukončující podmínka. Ve většině problému není jednoduché rozhodnout, kdy ukončit výpočet algoritmu. Při řešení obtížných problému neexistují žádné přesně specifikované hranice, kterých by mělo řešení problému dosáhnout a ani zda je možno najít lepší řešení. Proto se obvykle používají tyto možnosti:

1. celková doba výpočtu,
2. počet generací v rámci EA,
3. doba, kdy se fitness hodnota nezlepší přes určitou hranici nebo
4. diverzita populace klesne pod určitou hranici.

Pokud má problém známé globální minimum, může být tato hodnota využita jako ukončující podmínka spolu se zmíněnými možnostmi.

2.3 Kartézské genetické programování

Kartézské genetické programování (angl. *Cartesian Genetic Programming*, zkráceně CGP) je speciální druh genetického programování, kde jedinci jsou reprezentováni jako acyklické orientované grafy představující programy. Poprvé je J.F. Miller představil v roce 1999, ačkoliv již o dva roky dříve byl princip využit pro návrh elektronických obvodů. CGP našlo během let aplikaci v mnoha odvětvích, jako například od tvorby neuronových sítí, přes symbolickou regresi až po automatický design [22, 24].

2.3.1 Reprezentace jedince

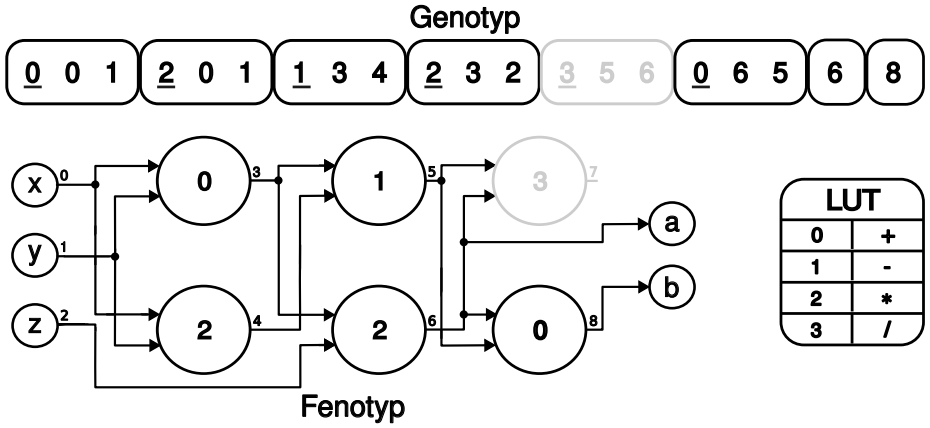
Ve standardním CGP, jak již bylo zmíněno, jsou jedinci reprezentováni acyklickým orientovaným grafem, kde uzly jsou poskládány do dvoudimenzionální mřížky (řádky, sloupce). Každý uzel v grafu představuje konkrétní výpočetní funkci a každý uzel je zakódován určitým počtem celočíselných genů v genotypu jedince. Jednotlivé funkce mají přiřazen svůj identifikátor, neboli svůj **funkční gen**, který je definován v uživatelské LUT (look-up tabulka). Následující geny uzlu určují, odkud do uzlu budou přicházet vstupy, jsou to tzv. **propojovací geny**. Jako vstupy uzlů jsou buď brány výstupy z předchozích sloupců nebo přímo vstup programu. Opět je to jednoznačně určená adresa, která definuje místo v celkové datové struktuře. Počet vstupů propojovacích genů je určeno maximální aritou výpočetní funkce v LUT. Pokud některá z funkcí vyžaduje menší počet vstupů, jsou přebytečné propojovací geny ignorovány.

Celkový genotyp jedince tvoří spolu se zakódovanými uzly mřížky také požadovaný počet výstupů programu. Výstupy jsou zakódovány posledními geny každého jedince a představují adresu uzlu, která poskytuje konečný výstup.

CGP má tři uživatelsky specifikovatelné parametry, které jsou počet řádků r, počet sloupců c a levels-back l. První dva z těchto parametrů určují maximální počet výpočetních uzlů, který je dán výrazem $r \times c$. Parametr l určuje konektivitu grafu vytvořením omezení na vstupy uzlů. Při $l = 1$, uzel smí brát své vstupy pouze z výstupů předchozího sloupce, nebo pokud se jedná o uzel z první vrstvy tak přímo z hlavního vstupu programu. Zvyšováním hodnoty l se dosah uzlů zvětšuje. Pokud bude nastaven parametr na $l = c$, uzly můžou být propojeny s jakýmkoliv výstupem uzlu předchozích sloupců, či vstupu programu. Nastavením těchto parametrů lze docílit vzniku různých topologií grafů.

Ukázku využití CGP pro navrhnutí matematických výrazu je zobrazena na obrázku 2.2. Úkolem CGP je navrhnout program se třemi vstupy a dvěma výstupy. Vstupy jsou označeny jako x, y, z a výstupy a, b. Výpočetní mřížku uživatel definoval jako 2 řádky a 3 sloupce. Parametr l je nastaven na 2. LUT tabulka obsahuje matematické operace sčítání (0), odečítání (1), násobení (2) a dělení (3). Výsledek hledání ukazuje nalezenou grafovou strukturu a demonstruje transformaci genotypu na fenotyp. Nalezené výrazy jsou definovány rovnicemi:

$$a = (x + y) \cdot z,$$
$$b = ((x + y) \cdot z) + ((x + y) - (x \cdot y)).$$



Obrázek 2.2: Schéma k příkladu návrhu matematického výrazu pomocí CGP. Šedou barvou je označen uzel, který se neúčastní výpočtu výstupu, tzv. **neaktivní uzel**. V genotypu je tato část označena šedě a nazývá se **nekódující gen**. Černou barvou jsou označeny aktivní části genotypu a fenotypu.

2.3.2 Evoluce jedinců

CGP pro vývoj své populace jedinců většinou využívá pouze genetický operátor mutace. Již v originální práci na CGP [23], Miller zjistil, že křížení nemělo velký vliv na účinnost CGP, a proto ve většině jeho následné práce křížení ignoroval. Důvodem, proč se křížení nepodařilo úspěšně aplikovat v mnoha praktických úlohách je, že dosud nebyl nalezen způsob jak, zachovat při křížení pozitivní vlastnosti obou rodičů a tak vytvářet potomky s vyšší fitness.

Hlavní dva druhy mutace využívající se v CGP jsou bodová mutace (angl. *point mutation*) a pravděpodobnostní mutace (angl. *probabilistic mutation*). V prvním případě uživatel volí procento z celkového počtu genů, na kterých proběhne mutace. Při pravděpodobnostní mutaci zvolené procento určuje, s jakou pravděpodobností má každý gen jedince šanci na mutaci. Mutace reprezentuje změnu genů na další validní hodnoty. Pokud zvolený mutující gen reprezentuje výpočetní uzel, změna probíhá náhodným vybráním validní adresy z LUT. Při mutaci propojovacích genů jsou validní hodnoty adresy výstupů předchozích sloupců nebo adresy vstupů programu omezeny parametrem levels-back. Validní mutace genu výstupu je změna hodnoty na jakoukoliv adresu uzlu nebo vstupu.

Praktickým použitím CGP se ukázalo, že velké procento uzlů je často kompletně neaktivní (obrázek 2.2) a tyto uzly nemají žádný vliv na podobu výsledného fenotypu, a tedy ani na fitness hodnotu (velká redundance aktivních genů). Z toho vyplývá, že velká část mutací proběhne v těchto oblastech a nebude mít žádný okamžitý vliv (tzv. **neutrální mutace**). Vliv těchto neutrálních změn se může projevit v pozdějších generacích vývoje populace, jelikož malá změna dokáže změnit kompletně výsledný program. Tomuto jevu se v CGP prostředí říká **neutrální drift** a ukázalo se zkoumáním, že tento jev je v mnoha případech velmi prospěšný pro efektivitu algoritmu [21].

2.3.3 Prohledávací algoritmus

Standartní CGP využívá prohledávací algoritmus inspirovaný evoluční strategii (μ+λ). Nejčastěji se algoritmus vyskytuje ve formě (1+4)[24]. Parametry μ, λ značí chování algoritmu, kde μ je počet kandidátních řešení, které jsou vybrány jako rodiče, zatímco λ označuje počet nově vygenerovaných potomků. Tvorba nové populace v CGP tedy funguje na principu mutace nejlépe vyhodnoceného jedince z předchozí generace a vytvoření několika potomků, kteří se spolu s rodičem dostanou do další generace. Funkce algoritmu je popsána pseudokódem 2.

Velmi podstatná vlastnost tohoto algoritmu se týká selekce nejlépe ohodnoceného jedince jako rodiče. Při jeho výběru může nastat situace, že jeden či více potomků dosahuje stejné hodnoty fitness jako rodič a žádný z potomků nemá lepší fitness. V tomto případě se rodičem stane náhodný potomek, který má stejnou hodnotu fitness s rodičem. Toto rozhodnutí souvisí s jevem neutrálního driftu a má podstatný vliv na efektivitu procesu evoluce. Jak již bylo vzpomenuto výše, tak i při stejné hodnotě fitness jedinců se jejich kódující i nekódující geny se mohou lišit. Výběrem nově vzniklého jedince tímto do procesu evoluce vnáší další užitečnou diverzitu.

Algorithm 2: Prohledávací algoritmus CGP $(1 + \lambda)$

```
INICIALIZACE populace o velikost 1 + λ ;
EVALUACE kandidátních řešení;
while !podmínka ukončení do
    SELEKCE nejlepšího jedince jako rodiče;
    for (i = 0, i < λ, i++) do
        MUTACE rodiče k vniku potomka i;
    end
    EVALUACE nových kandidátních řešení;
end
```

Kapitola 3

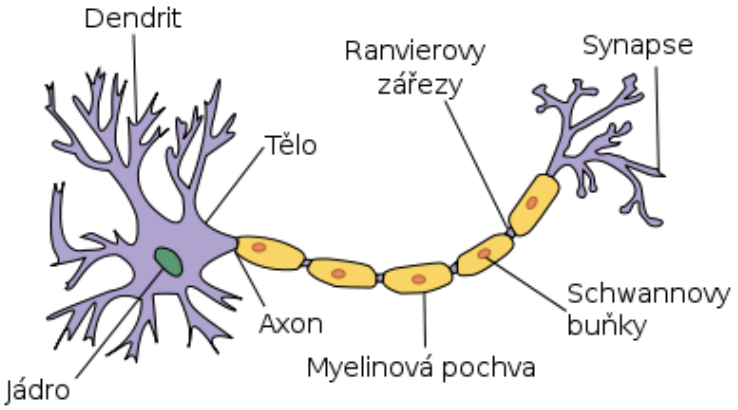
Umělé neuronové sítě

Umělé neuronové sítě (*Artificial Neural Networks*, nebo-li zkráceně ANN) odpovídají jednomu z mnoha výpočetních modelů umělé inteligence. Jejich počáteční vývoj byl inspirován biologickým nervovým systémem a jeho atraktivními vlastnosmti při zpracování informací.

Jeden z prvních objevů, který vzbudil zájem o toto odvětví, provedli neurovědec Warren McCulloch a matematik Walter Pitts, kteří v roce 1943 přišli s konceptuálním návrhem umělého neuronu. Demonstrovali jej pomocí elektrických obvodů. Jejich práce ukázala, jak jednoduchá výpočetní jednotka může replikovat logické nebo matematické funkce. Dalším výrazným krokem v této oblasti se stal model **perceptronu** a jeho základního algoritmu pro učení. Tyto objevy položily základy neuronovým sítím, jak jsou známy dnes. Tato kapitola byla zpracována s využitím zejména [30].

3.1 Umělý neuron

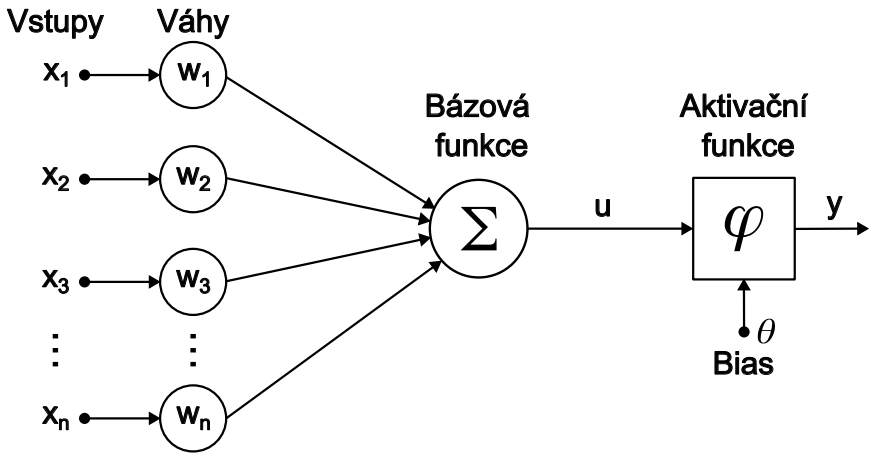
Umělý neuron je základní stavební jednotkou umělých neuronových sítí. Snaží se imitovat princip a anatomii biologického neuronu obrázek 3.1. Hlavní část biologického neuronu je tvořena tělem buňky (soma), které obsahuje jádro, obklopené velkým množstvím malých výběžků, nazývaných **dendrity**. Dendrity slouží jako vstup neuronu a jsou zodpovědné za příjímání impulzů z ostatních neuronů. Výstup neuronu se nazývá **axón**, který je stejně jako dendrity rozvětvený, aby mohl vytvářet více vazeb s ostatními neurony. Spojení dvou neuronů je provedeno pomocí synaptických vazeb. Jedná se o elektrochemické vazby, které dokáží přenášet impulzy mezi neurony.



Obrázek 3.1: Biologický neuron. Převzato z [6].

Impulzy jsou v těle neuronu akumulovány a při překročení vnitřního potenciálu je neuron excitován. V takovém případě dojde k přenosu impulzu z těla neuronu k synaptické vazbě, kde proběhne reakce, která tuto vazbu posílí nebo zeslabí. Tento proces je popsán jako základní operace potřebná pro učení živočišných druhů, kterou poprvé popsal Donald Hebb v *The Organization of Behavior* [14] v roce 1949.

Model umělého neuronu převzal princip upravitelných vah, a díky tomuto dokáže pomocí algoritmu učení vylepšit své chování pro danou úlohu. Základní schéma umělého neuronu je zobrazeno na obrázku 3.2.



Obrázek 3.2: Model umělého neuronu.

Neuron bere jako svůj vstup n rozměrný vektor $\vec{x} = (x_1, x_2, \ldots, x_n)$. S každým vstupem x_i je propojena váha w_i pro $i = 1, 2, \ldots, n$, neboli vektor $\vec{w} = (w_1, w_2, \ldots, w_n)$. Jednotlivé vstupy jsou následně váhově sečteny do vnitřního potenciálu neuronu a přičtena prahová hodnota bias θ (běžně $w_0 = \theta$). Tato operace se nazývá bázová funkce a je matematicky vyjádřena:

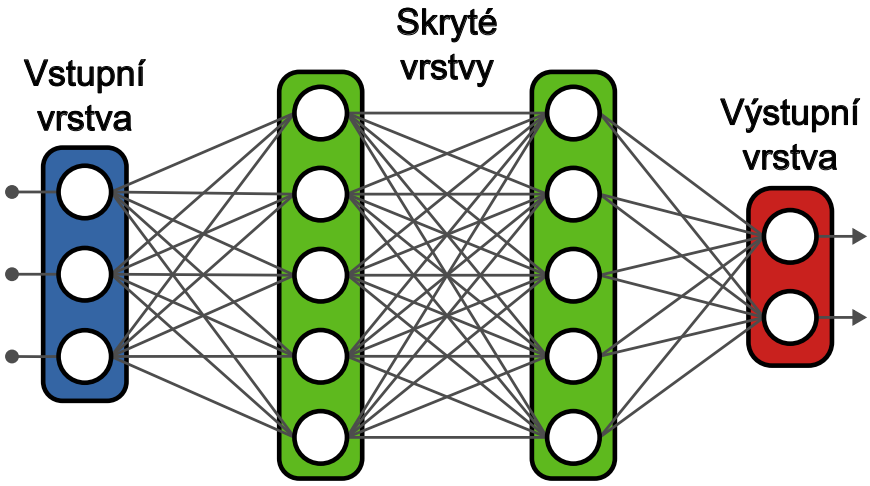
$$u = \vec{x}\vec{w} = \sum_{i=1}^{n} x_i w_i + \theta. \tag{3.1}$$

Vnitřní potenciál neuronu je stále jen lineární vážená kombinace vstupu, která nemá velkou popisující sílu vazeb v komplexním světě. Z tohoto důvodu je další prvkem umělého neuronu tzv. **aktivační funkce**, která zavádí do modelu nelinearitu a transformuje výstup do požadovaného intervalu (mění aktivaci). Více o používaných aktivačních funkcích je popsáno v sekci 3.3.

3.2 Dopředná neuronová síť

Dopředné sítě (angl. *feed-forward network*) jsou nejběžnější typ neuronových sítí. Sítě tvoří skupiny umělých neuronů poskládaných do vrstev (vstupní, skryté, výstupní) propojovaných jedním směrem tak, aby vznikl acyklický dopředný graf. Ukázka dopředné sítě je zobrazena na obrázku 3.3.

Informace při výpočtu odezvy sítě putuje od vstupní vrstvy k výstupní. Na vstup sítě je přikládán vektor $\vec{x}$ a každý neuron ve vstupní vrstvě vypočítá svůj vnitřní potenciál neuronu, na který poté aplikuje aktivační funkci a vytvoří tak výstup neuronu. Tyto výstupy jsou následně poskytnuty další vrstvě jako vstup. Tento princip se opakuje, až se informace dostane do výstupní vrstvy, která vytvoří celkový výstup sítě.



Obrázek 3.3: Model acyklické dopředné neuronové sítě.

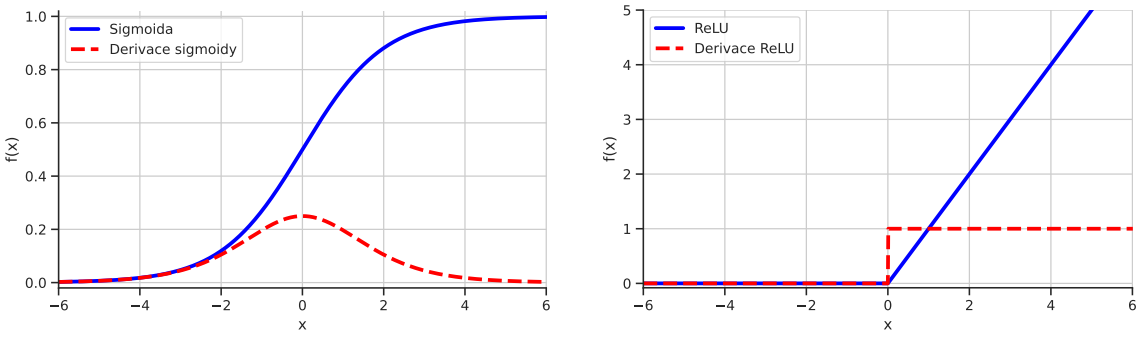
3.3 Aktivační funkce

Aktivační funkce jsou matematická funkce, které transformují aktivaci neuronu. Tedy upravují schopnost propagace svého výstupu dalším neuronům. Touto transformací v síti vzniká nelinearita, díky které je síť schopna popsat složitější vztahy mezi daty reálného světa.

Správný výběr aktivační funkce v neuronových sítích má velký vliv na celkové chování sítě a ovlivňuje proces konvergence při trénování. Je však důležité zdůraznit, že neexistuje jednotné pravidlo pro volbu aktivační funkce platící pro všechny případy užití. Nicméně jsou některé žádoucí vlastnosti, které by měla aktivační funkce splňovat [17]:

- Nelinearita – Je jednou z nejdůležitějších vlastností aktivační funkce. Oproti lineárním výrazně zlepšuje schopnost učení neuronových sítí. G. Cybenko [7] a K. Hornik [15] v roce 1989 tuto vlastnost zkoumali a dokázali, že spolu s dalšími omezujícími podmínkami na aktivační funkce lze využít síť s architekturou, alespoň jedné skryté vrstvy s dostatečným počtem neuronů jako univerzální funkční aproximátor (tzv. **univerzální aproximační teorém**).
- Výpočetně nenáročné – Aktivační funkce musí být snadno vyhodnotitelná z hlediska výpočtu. To má potenciál výrazně zlepšit efektivitu sítě.
- Omezení saturace – Aktivační funkce by měla minimalizovat riziko saturace, kdy je výstup funkce příliš blízký k jejím asymptotám, což by mohlo zpomalit nebo ztížit průběh učení při gradientních metodách (tzv. **problém mizejícího gradientu**).
- Konečný výstupní rozsah – Trénovací přístupy založené na gradientu jsou stabilnější, když obor hodnot aktivační funkce je omezený.
- Diferencovatelnost – Nejvíce žádoucí vlastností pro používání optimalizačních postupů založených na gradientu jsou spojitě diferencovatelné aktivační funkce. Na těchto metodách je založeno učení neuronových sítí sekce (3.4) a tato spojitá diference zajistí, že algoritmus zpětné propagace pracuje správně na všech hodnotách.

Níže v této sekci jsou uvedeny některé z nejvíce využívaných aktivačních funkcí.



(a) Sigmoida (b) ReLU

Obrázek 3.4: Průběhy aktivačních funkcí a jejich derivací.

3.3.1 Sigmoida

Sigmoida, neboli logistická funkce často značená jako $\sigma(x)$ transformuje rozsah vstupních hodnot do intervalu (0, 1). Při vysokých x-hodnotách se výstupní hodnota funkce limitně blíží k hodnotě 1, naopak při vysokých záporných k 0. Funkce je vhodná pro predikci pravděpodobností díky jejímu vhodnému oboru hodnot. Se sigmoidou souvisí problém mizejícího gradientu zmíněném v této sekci (obrázek 3.4a). Funkce Sigmoidy je definována vztahem:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{3.2}$$

3.3.2 ReLU

ReLU (angl. *Rectified Linear Unit*) je nejvíce využívaná aktivační funkce pro hluboké učení. Hlavní výhodou funkce je výpočetní nenáročnost a schopnost řešit problém mizejícího gradientu. ReLU při kladných hodnotách x propaguje stejnou hodnotu na výstup, avšak v záporných hodnotách vstupu je výstupem 0, viz obrázek 3.4b. Nevýhodou užití ReLU je tzv. umrtvení neuronů, což je stav, kdy neurony jsou vyřazeny z kladného aktivačního rozsahu. V tomto stavu gradient chybové funkce se rovná nule, což zapříčiňuje, že nedochází k úpravě vah a tak se neuron neúčastní řešení požadovaného problému. Funkce ReLU je definována vztahem:

$$f(x) = max(0, x) = \begin{cases} x & \text{pro } x \geq 0, \\ 0 & \text{pro } x < 0. \end{cases} \tag{3.3}$$

Existují modifikace funkce ReLU jako jsou například **Leaky ReLU** nebo **eLU**, které mění průběh funkce v záporné části vstupních hodnot a snaží se tak řešit zmíněný problém.

3.3.3 Softmax

Funkce Softmax se obvykle využívá v sítích řešící klasifikační problém. Jedná se o klasifikaci do předem definovaného počtu tříd K. Na vstupu funkce je tedy K dimenzionální vektor, který je funkcí normalizován do vektoru se součtem o velikosti 1. Každá z těchto hodnot lze interpretovat příslušnost k dané třídě, neboli pravděpodobnost. Funkce Softmax je obecně definována takto:

$$f(x)_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}} \tag{3.4}$$

3.4 Učení neuronové sítě

Existuje několik přístupů k trénování neuronové sítě. Tato práce zaměřuje na učení s učitelem (angl. *supervised learning*). Toto učení je založeno na znalosti **ground truth** dat, což je znalost správných výstupů k příslušné vstupní datové sadě. Z porovnání aktuálního výstupu sítě a ground truth dat lze získat **chybovou funkci** (angl. *loss function*), která měří úspěšnost predikce neuronové sítě. Běžná metrika chyby využívaná pří tréninku klasifikačních úloh je tzv. **kategorická křížová entropie** (angl. *categorical cross-entropy*), která je matematicky definována:

$$L(\hat{y}, \vec{y}) = -\sum_{i=1}^{n} y_i \cdot \log(\hat{y}_i), \tag{3.5}$$

kde n je počet klasifikačních tříd, y_i je ground truth one-hot zakodovaný vektor (one-hot encoding znamená, že právě jedna třída bude mít pravděpodobnost 1 a ostatní 0) a $\hat{y}_i$ je vektor predikce výsledků z funkce softmax.

Učení neuronové sítě spočívá v opakované úpravě vah tak, aby chybová funkce byla co nejmenší pro celou datovou sadu. Váhy jsou upravované pomocí optimalizační metody zvané **gradientní sestup** (angl. *gradient descent*), využívající záporného gradientu. Gradient je vektor parciálních derivací chybové funkce vzhledem k určité váze neuronové sítě. Úprava vah je definovaná následujícím vzorcem:

$$w_{i+1} = w_i - \alpha \nabla f(w_i), \tag{3.6}$$

kde w_i reprezentuje váhy před aktualizací, w_{i+1} váhy po aktualizaci. α je parametr sítě zvaný **krok učení** (angl. *learning rate*), $\nabla f(w_i)$ je gradient chybové funkce.

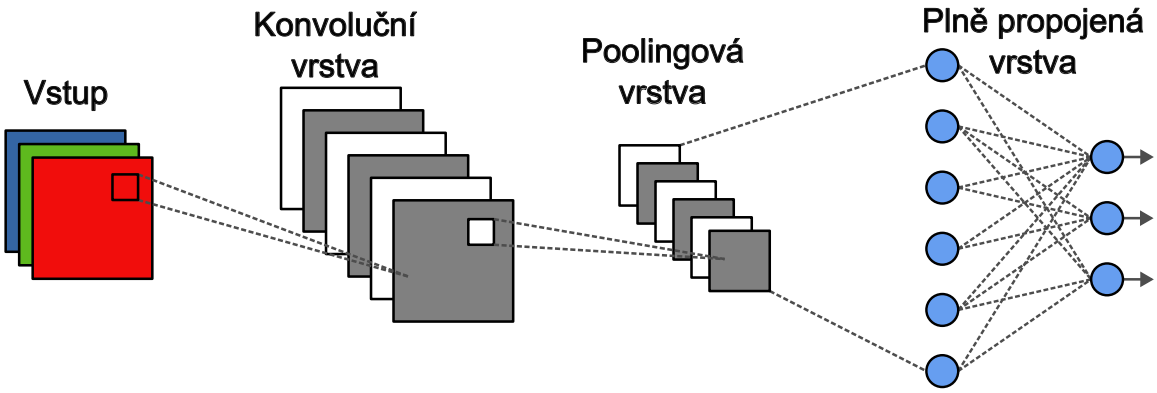
Pro efektivní výpočty gradientů se využívá algoritmus **zpětné šíření chyby** (angl. *back-propagation*). Tento algoritmus využívá tzv. řetězového pravidla, což je systematický způsob výpočtu gradientu směrem od výstupní vrstvy ke vstupní. Část výpočtu je znovu použita pro výpočet gradientu další vrstvy.

Existují varianty základního gradientního sestupu jako jsou **Stochastic** a **Mini-Batch**. Tyto varianty se liší počtem vzorků, které jsou využity pro výpočet chybové funkce k aktualizaci vah.

3.5 Konvoluční neuronové sítě

Konvoluční neuronové sítě (angl. *Convolutional neural nets*, zkráceně CNNs) jsou speciálním druhem dopředných neuronových sítí, které jsou v dnešní době nejvíce používané pro zpracování obrazu a řeči. Hlavní výhodou je, že síť vytváří lokální receptivní pole, pomocí kterého dokáže z dat vyextrahovat potřebné užitečné rysy za použití menšího počtu trénovacích parametrů (vah) než u normálních neuronových sítí. To umožňuje vytvářet hlubší neuronové sítě bez exponenciálního výpočetního nárůstu, a tak umožňuje řešit složitější problémy.

Základní architektura CNN se skládá ze čtyř typů vrstev – konvoluční, aktivační, pooling a plně propojené vrstvy. Typické umístění vrstev CNN je uvedena na obrázku 3.5. Vývojem v oblasti hlubokého učení se do této čtveřice přidala také normalizační vrstva, která vylepšuje vlastnosti architektur. Používá se většinou před či za aktivační vrstvou a normalizuje tok mapy příznaků v síti [1, 19].



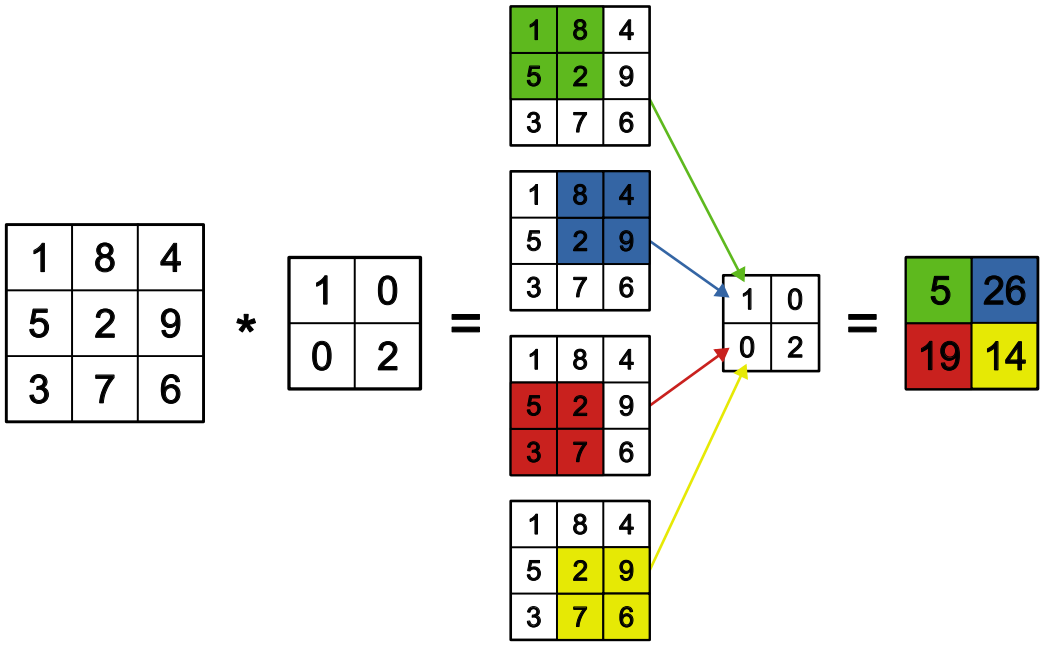
Obrázek 3.5: Klasická architektura konvoluční neuronové sítě.

3.5.1 Konvoluční vrstva

Tato vrstva je nejdůležitější částí CNN, slouží k extrakci užitečných rysů ze vstupních dat. Skládá se ze sady prostorově malých trénovatelných filtrů, které pro extrakci užívají operace konvoluce vyjádřenou vztahem:

$$O[x,y] = \sum_{i=0}^{k_1-1} \sum_{j=0}^{k_2-1} K[i,j] * I[x+i, y+j], \tag{3.7}$$

kde $O[x,y]$ představuje novou hodnotu pixelu na pozici $[x,y]$, I značí vstupní snímek a K označuje konvoluční filtr (angl. kernel) o rozměrech $k_1 \times k_2$. Graficky je tato operace vyobrazena na obrázku 3.6. Z tohoto obrázku si lze představit funkci vrstvy jako posouvání konvolučního filtru po vstupu a počítání vážené sumy překryvu oblastí. Výstupem operace je tzv. mapa příznaků, neboli mapy aktivace konvolučního filtru.

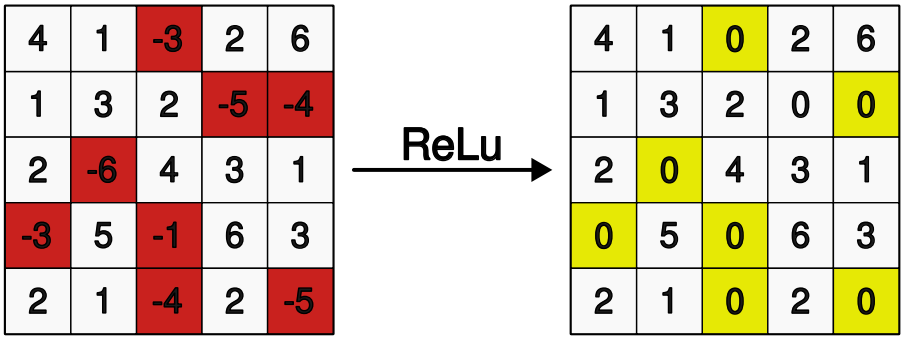


Obrázek 3.6: Grafická reprezentace konvoluce se vstupem.

Konvoluční filtry mají trénovatelné váhy uspořádány do 3D matice, kde dimenze H značí výšku, W šířku a C počet kanálů, který je převzat z dimenze vstupu. Konvoluční vrstva pak obsahuje sadu těchto filtrů, přičemž počet sad filtrů udává výstupní počet map příznaků, neboli počet výstupních kanálů.

3.5.2 Aktivační vrstva

Účelem této vrstvy je stejně jako u normálních neuronových sítí zavést do modelu nelinearitu a tím zlepšit vlastnosti modelu. K tomu se používají stejné aktivační funkce jako v sekci 3.3. Aktivační vrstva se nejčastěji používá na výstup z konvolučních vrstev. U konvolučních neuronových sítí převládá používání funkce ReLU. Ukázka použití této funkce na vstup je zobrazena na obrázku 3.7



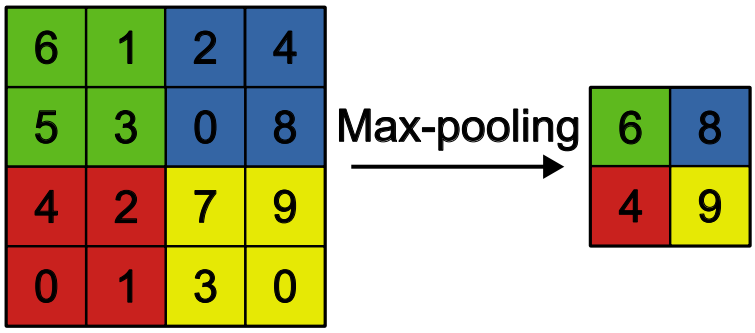
Obrázek 3.7: Ukázka aplikace aktivační funkce ReLU na vstup.

3.5.3 Poolingová vrstva

Poolingová vrstva slouží k redukci prostorových rozměrů vstupních dat a zachovává ty nejdůležitější informace. Obvykle se vrstva používá za konvoluční a aktivační vrstvou, kde dojde k nárůstu dat s počtem zvolených konvolučních filtrů. Díky jejímu využití se sníží počet parametrů, což ovlivní výpočetní náročnost modelu.

Hlavní myšlenkou, na které je vrstva založena, je, že není potřeba znát přesnou polohu vyextrahovaných rysů, avšak postačí pouze jejich přibližná vzájemná poloha. Model se tak stává více robustní vůči translaci (posuvu) ve vstupním obrázku.

Nejvíce používaným typem poolingové vrstvy je tzv. max-pooling. Funkce max-poolingu je zobrazena na obrázku 3.8.



Obrázek 3.8: Ukázka funkce max-pooling vrstvy. Vstupní obrázek je barevně rozdělen do dlaždic a do výstupu je posunuta maximální hodnota z každé dlaždice.

3.5.4 Dávková normalizační vrstva

Dávková normalizační vrstva (angl. *batch normalization layer*) je osvědčený způsob jak zrychlit učení neuronových sítí, a mnoho studií potvrdilo její klíčovou roli k dosažení state-of-the-art výsledků.

Hlavní problém, který vrstva řeší, je tzv. **internal covariate shift**. Tento problém se vyznačuje posuvem rozložení aktivace neuronů během učení neuronové sítě. Dokonce i malý posun průchodem sítě posiluje, což může způsobit zpomalení konvergence během tréninku.

Dávková normalizační vrstva se tomu snaží zabránit a provádí korekci aktivace neuronů mezi vrstvami a transformuje ji do normálního rozložení se střední hodnotou 0 a rozptylem 1. Chování vrstvy pro trénovací dávku $\mathcal{B} = \{x, ..., x_m\}$ o velikosti dávky m je vyjádřeno rovnicemi:

$$\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} x_i, \tag{3.8}$$

$$\sigma^2_{\mathcal{B}} = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu_{\mathcal{B}})^2, \tag{3.9}$$

$$\hat{x_i} = \frac{x_i - \sigma^2_{\mathcal{B}}}{\sqrt{\mu^2_{\mathcal{B}} + \epsilon}}, \tag{3.10}$$

$$y_i = \gamma\hat{x_i} + \beta \equiv BN_{\gamma,\beta}(x), \tag{3.11}$$

kde μ značí střední hodnotu v dávce $\mathcal{B}$ a σ^2 její rozptyl, x_i jsou jednotlivé trénovací vstupy a výstupem je normalizovaná dávka označena jako $y_i \equiv BN_{\gamma,\beta}(x)$, jelikož normalizace může změnit význam dat, jsou zde dva trénovací parametry, které tomu zabraňují γ, β.

Použitím této vrstvy se zvyšuje odolnost sítě vůči počátečním parametrům, což umožňuje akcelerovat trénink pomocí větších hodnot trénovacího kroku [16].

3.5.5 Plně propojená vrstva

Plně propojená vrstva je klasická dopředná neuronová síť. V CNN se využívá několik těchto vrstev k určení finální predikce sítě. Jako vstup je dodávána 2D příznaková mapa, ta nejprve musí být zploštěna na 1D vektor, aby mohla být použita jako validní vstup. Dále je tento vstup zpracován plně propojenými vrstvami a předán výstupní vrstvě. Ve většině případů počet neuronů výstupní vrstvy určuje počet tříd klasifikace.

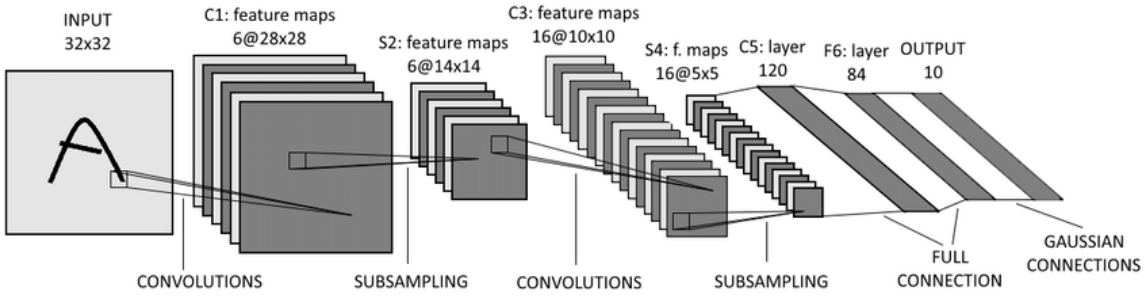
3.6 Využité architektury CNN

V průběhu let byly vyvinuty různé architektury konvolučních neuronových sítí (CNN), které se postupně stávaly state-of-the-art v oblasti hlubokého učení. Pro provedení experimentů s automatickým návrhem aktivačních funkcí byly vybrány tři různé CNN s rostoucí složitostí v počtu parametrů. Jednotlivé architektury jsou popsány v této sekci níže.

3.6.1 LeNet-5

LeNet-5 [20] je jednoduchá dopředná konvoluční neuronovou síť, která započala první kroky v oblasti hlubokého učení. Architektura byla navrhnuta v roce 1998 Yannem LeCunem a kolegy s cílem klasifikovat ručně psané číslice pro poštovní službu Spojených států amerických k strojovému třídění zásilek podle poštovního směrovacích čísel.

Skládá se ze sedmi vrstev, které jsou zobrazeny na obrázku 3.9, nebo také znázorněny v tabulce 3.1. Vstupem pro tuto síť, jak je možno vidět z tabulky, jsou černobílé obrázky o rozměru 32×32 pixelů.



Obrázek 3.9: Ukázka architektury LeNet-5[1].

Vrstva	Kanály	Výstup	Jádro	Stride	Aktivační funkce
Vstup	1	32×32	–	–	–
Konvoluční	6	28×28	5×5	1	ReLU/PReLU/∗
Avg pooling	6	14×14	2×2	2	–
Konvoluční	16	10×10	5×5	1	ReLU/PReLU/∗
Avg pooling	16	5×5	2×2	2	–
Konvoluční	120	1×1	5×5	1	ReLU/PReLU/∗
Plně propojená	84	–	–	–	ReLU/PReLU/∗
Plně propojená	10	–	–	–	–

Tabulka 3.1: Architektura sítě LeNet-5. Symbol ∗ značí aktivační funkce vniklé při experimentech.

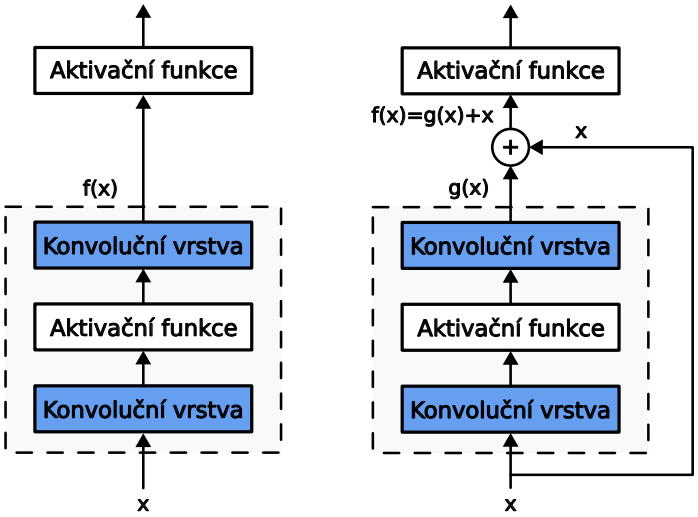
[1] Převzato z https://www.researchgate.net/figure/Architecture-of-LeNet-5_fig3_313808170.

3.6.2 ResNet-10

ResNet-10 [13] představuje konvoluční neuronovou síť z rodiny sítí ResNet, která se specializuje na vytváření velmi hlubokých neuronových sítí. Při vytváření hlubokých CNN se tradičně setkává s problémem degradace výkonu při použití metody zpětné šíření chyby. Tento jev způsobuje, že vrstvy stále vzdálenější od výstupu mají potíže s aktualizací svých vah během trénování.

ResNet se vypořádává s tímto problémem pomocí **reziduálního propojení** (angl. *skip connection*), jak je znázorněno na obrázku 3.10. Tato propojení umožňují efektivnější šíření chyby během procesu zpětné propagace. Díky reziduálním blokům může síť snadněji učit identitu a lépe optimalizovat váhy, což v konečném důsledku umožňuje konstrukci a trénování hlubších a výkonnějších neuronových sítí.

Architektura modelu ResNet-10 je popsána tabulkou 3.2. Jak je patrné, model se skládá z čtyř reziduálních bloků, které jsou zobrazeny na pravém obrázku 3.10. V experimentovaném modelu je každý reziduální blok obohacen o dávkové normalizační vrstvy umístěné před každou aktivační funkcí.



Obrázek 3.10: Ukázka reziduálního propojení v CNN architektuře. V běžném bloku (vlevo) je nutné, aby část uvnitř šrafovaného boxu se přímo naučila mapování $f(x)$. V reziduálním bloku (vpravo) musí část uvnitř šrafovaného boxu naučit reziduální mapování $f(x) = g(x) + x$, což usnadňuje učení mapování identity $f(x) = x$.

Vrstva	Kanály	Jádro	Stride	Aktivační funkce
Konvoluční	64	7×7	1	ReLU/PReLU/∗
Residuální	64/64	3×3	1	ReLU/PReLU/∗
Residuální	128/64	3×3	2	ReLU/PReLU/∗
Residuální	256/64	3×3	2	ReLU/PReLU/∗
Residuální	512/64	3×3	2	ReLU/PReLU/∗
Plně propojená	10	–	–	–

Tabulka 3.2: Architektura sítě ResNet-10. Symbol ∗ značí aktivační funkce vniklé při experimentech.

3.6.3 WRN-40-4

WRN-40-4 [32] název této architektury značí zkratku Wide Residual Network (WRN) s parametry 40-4. Tato architektura neuronové sítě je odvozena z ResNetu a rozšiřuje tento koncept tím, že se zaměřuje na zvětšení šířky (počet filtrů v konvolučních vrstvách) a snížení hloubky sítě (počet vrstev) v porovnání s klasickými modely ResNet. Těmito změnami sítě WRN dosahují větší přesnosti odhadu a zmenšení hloubky sítě pomáhá jejich rychlejší konvergenci.

Konkrétně parametry označují počet vrstev a nárůst počtu filtrů. V tomto případě 40 znamená, že síť má celkem 40 vrstev, a číslo 4 označuje nárůst počtu filtrů v jednotlivých sekcích modelu. Celková architektura je popsána tabulkou 3.3.

Vrstva	Kanály	Jádro	Stride	Aktivační funkce
Konvoluční	16	3×3	1	ReLU/PReLU/∗
6×Residuální	64	3×3	1	ReLU/PReLU/∗
6×Residuální	128	3×3	1	ReLU/PReLU/∗
6×Residuální	256	3×3	1	ReLU/PReLU/∗
Plně propojená	10	–	–	–

Tabulka 3.3: Architektura sítě WRN-40-4. Symbol ∗ značí aktivační funkce vniklé při experimentech.

V tabulce 3.4 jsou uvedeny vybrané architektury konvolučních sítí a porovnání jejich složitosti vzhledem k počtu parametrů.

Architektura	Počet parametrů (v milionech)
LeNet-5	0,06
ResNet-10	1,12
WRN-40-4	8,94

Tabulka 3.4: Porovnání počtu parametrů vybraných modelů použitých v experimentech s metodou automatického návrhu aktivačních funkcí.

3.7 Klasifikační úloha

Klasifikační úloha je jednou ze základních využití konvolučních neuronových sítí. Jedná se o úlohu přiřazení objektu ke správné třídě. Všechny experimenty s návrhem aktivačních funkcí budou testovány a vyvíjeny na klasifikační úloze. Proto byly zvoleny tři velmi známé a využívané klasifikační datové sady – MNIST, CIFAR-10, FashionMNIST. Jednotlivé datasety budou více popsány níže.

3.7.1 MNIST

MNIST (*Modified National Institute of Standards and Technology database*) je datová sada obsahující ručně psané arabské číslice (0-9). Dataset tvoří 70 000 černobílých snímků o velikosti 28×28 s rovnoměrným zastoupením pro každé číslo. Dataset je rozdělen na trénovací (60 000 snímků) a testovacího set (10 000 snímků). Ukázka snímků z datasetu je možno vidět na obrázku 3.11.

Obrázek 3.11: Ukázka snímků z datasetu MNIST.

3.7.2 FashionMNIST

Fashion-MNIST je další populární datová sada pro strojové učení, podobně jako MNIST, ale zaměřuje se na klasifikaci obrázků oděvů a jejich doplňků. Obsahuje 60 000 černobílých obrázků o rozměrech 28×28 pixelů rozdělených do 10 tříd. FashionMNIST je rozdělen na trénovací a testovací sadu, kde trénovací sada obsahuje 50 000 obrázků a testovací sada obsahuje 10 000 obrázků. FashionMNIST byl vytvořen, tak aby sloužil jako realističtější a složitější náhrada datasetu MNIST pro porovnání algoritmů strojového učení. Ukázka snímků z datasetu je zobrazeno na obrázku 3.12.

Obrázek 3.12: Ukázka snímků datasetu FashionMNIST[2].

[1]Převzato z https://www.researchgate.net/figure/Sample-Fashion-MNIST-images_fig3_356601779.

3.7.3 CIFAR-10

CIFAR-10 je datová sada pro klasifikační úlohu, která obsahuje 60 000 barevných snímků o velikosti 32×32, rozdělených do 10 tříd. Dataset je rozdělen na trénovací (50 000 snímků) a testovacího set (10 000 snímků). Jedná se o jeden z nejvíce používaných standardních datasetů pro klasifikační úlohu. Ukázka z datasetu je zobrazena na obrázku 3.13.

Obrázek 3.13: Ukázka snímků datasetu CIFAR-10.

Kapitola 4

Neuroevoluce

Biologická evoluce již dokázala svou sílu při navrhnutí tak komplexního a výpočetně úsporného systému jako je lidský mozek. Proto není divu, že tuto konceptuální ideu se výzkumníci snaží využít u neuronových sítí, které v posledních letech vytváří obrovské pokroky v oboru umělé inteligence. Spojení evoluční idey, tedy evolučních algoritmů a obecně neuronových sítí se nazývá **neurovoluce**. Evoluce umělých neuronových sítí lze použít s trochou představivosti v každém aspektu jejich činnosti, avšak stejně jako u biologického protějšku je tento proces zdlouhavý a výpočetně náročný.

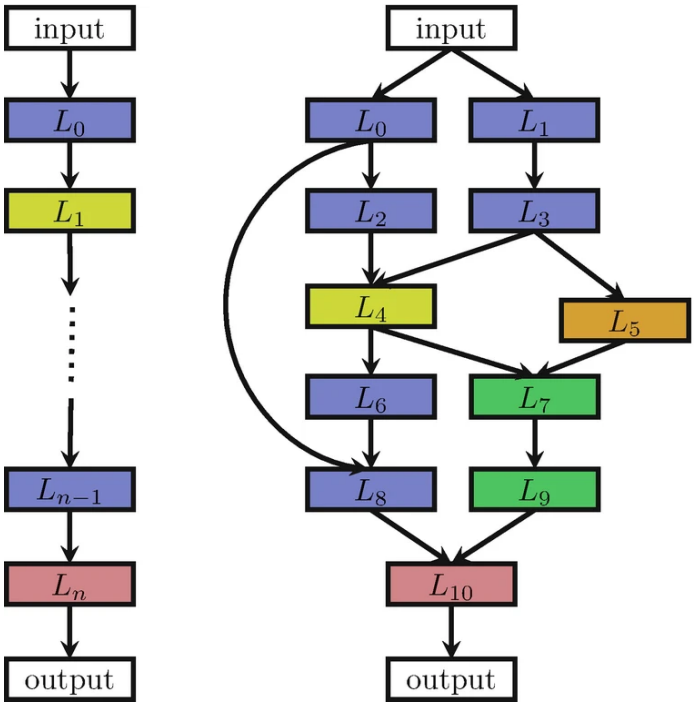
První a asi nejvíce využívaná oblast je návrh architektury sítě (angl. *Neural Architecture Search*, zkráceně **NAS**). Tato oblast byla doposud dominovaná hlavně odborníky z praxe a jejich znalostmi, avšak lidský design architektury může být zatížen právě těmito lidskými vlastnostmi. Evoluční algoritmus poskytuje nezaujatý pohled a systematický způsob vývoje pro tento úkol, což může vést k objevení nové lepší nebo efektivnější architektury.

Další oblastí využití evoluce lze definovat, jako využití evolučního algoritmu na fixní architekturu. Tedy evoluční algoritmus se snaží vylepšit vlastnosti sítě odlišným způsobem než změnou topologie sítě. Může to být nalezení trénovacích vah, popřípadě učícího algoritmu nebo parametrů učícího algoritmu atd. Jedno specifické využití evoluce je návrh nových druhů obecně využívaných komponent, které byly shrnuty v kapitole 3, čím se zabývá i tato diplomová práce a zaměřuje se na návrh nelineárních aktivačních funkcí. Tyto nově navržené typy komponent nemusí být vhodné pro obecné využití, avšak mohou zajistit lepší vlastnosti sítě na konkrétním vyvíjeném datasetu, či použité úloze.

4.1 Evoluce architektury

Architektura je nedílnou součástí neuronové sítě, která ovlivňuje celkovou funkčnost. Výběr správné architektury pro určitý úkol není jednoduchá záležitost, jelikož při zvolení příliš složité architektury dochází k přeučení sítě, což je stav, kdy síť se naučí závislosti ve vstupních datech až příliš dobře a celkově ztrácí schopnost generalizace na dosud neviděných datech. V opačném případě síť nemá potřebnou složitost k naučení skrytých závislostí pro dobrou funkčnost.

NAS metody tento problém řeší a vytváří sadu nástrojů pro automatizovaný návrh architektury. Základní rozdělení metod NAS článku [31] je podle – prohledávacího prostoru, prohledávací strategie a strategie odhadu výkonu. Jelikož se kapitola zaměřuje na neuroevoluci, prohledávací strategie pro generování nových architektur jsou zde zúženy na formy evolučních algoritmů. Nutno dodat, že během let vývoje se v NAS experimentovalo



Obrázek 4.1: Vizuální ukázka dvou rozdílných způsobů tvoření architektury CNN, kde každý barevný blok představuje některou z vrstev zmíněných 3.5. V obrázku nalevo vrstva L_n smí brát svůj vstup pouze z vrstvy L_{n-1}. V obrázku napravo zde takovéto omezení není, což umožňuje vnik *skip* propojení, což rozšiřuje prostor pro vznik složitějších architektur. Obrázek převzat z [10].

s mnoha různými optimalizačními algoritmy jako například posilované učení, bayesovská optimalizace atd.

Definice prohledávacího prostor je jednou z nejdůležitějších částí automatického návrhu, jelikož určuje všechny možné architektury sítí, které mohou být nalezeny. Všeobecně se do definice prohledávajícího prostoru zakomponovává znalosti o doméně a spolu *state-of-the-art* architekturami, což může ulehčit a zrychlit běh algoritmu. Nutno dodat, že návrh vyhledávacího prostoru představuje důležitý kompromis mezi lidskou zaujatostí (ve formě výběru komponent) a efektivitou vyhledávání.

Strategie odhadu výkonu jsou metody způsobu vyhodnocení jednotlivých kandidátních řešení. V jednoduchosti se jedná o trénovaní a validaci architektur na datech, avšak tento úkon je značně výpočetně náročný a tak limituje množství architektur, které mohou být prozkoumány. Proto se hledají alternativy urychlující vyhodnocení architektur například tréning a validaci architektur na podmnožiny dat nebo použití prediktoru fitness.

Další důležitou otázkou při neuroevoluci je výběr způsobu zakódování neuronové sítě. Existují dva používané způsoby – přímé a nepřímé kódování. Přímé kódování tedy znamená, že reprezentace genotypu jedince přesně odpovídá 1:1 mapování na fenotyp. Tedy každá část genu odpovídá nějakém neuronu nebo propojení neuronové sítě. Například průkopníkem přímého kódování a celkově neuroevoluce je algoritmus **NEAT** [28](*NeuroEvolution of Augmenting Topologies*) z roku 2002. Přímé kódování se současnosti nepoužívá z důvodu špatné škálovatelnosti pro hluboké neuronové sítě. Místo toho převládá využití nepřímého kódováni, které zvětšuje abstrakci (kódováni po vrstvách) a umožňuje zakódování hlubokých neuronových sítí v rozumném měřítku.

4.2 Evoluce vah

Trénování hlubokých dopředných neuronových sítí spočívá standardně v optimalizaci chybové funkce upravováním hodnot vah a biasů za použití metody zpětného šíření chyby s gradientním sestupem. S tímto používaným standardem jsou však spojena různá omezení, jako například využití diferencovatelných aktivačních funkcí, nebo problémy při trénování rekurentních neuronových sítí.

Proto jsou zkoumány jiné možnosti, jednou z nich je využití evolučních algoritmů. Výhodou přístupu je práce s množinou kandidátních řešení a výpočet (trénování) sítě je možné velmi efektivně paralerizovat. Tradiční forma evoluce vah spočívá v převedení všech vah modelu ne vektor, který reprezentuje jedince v populaci. Genetickými operátory je populace šlechtěna na požadovanou funkčnost.

Tento způsob však není optimálně škálovatelný pro dnes využívané neuronové sítě, proto se v dnešní době výzkumníci snaží objevit efektivní metody pro zakódování vah neuronových sítí. Jeden z experimentů, který se týká této problematiky v kontextu CNN, je prezentován v článku [11]. Autoři zde představují svůj přístup kódovaní, který zdokonaluje funkčnost evolučního tréninku ve srovnání s tradiční metodou, avšak při srovnáním s gradientními metodami celkově výkonnostně zaostává. Obecně se zatím nepodařilo nalézt evoluční algoritmus, který by předčil gradientní metody při učení hlubokých neuronových sítí.

4.3 Evoluce aktivačních funkcí

Volba aktivační funkce v hlubokém neuronových sítí má velký vliv na proces tréninku a vykonávání požadované úlohy. V dnešní době je nejvíce všeobecně využívaná aktivační funkce ReLU (sekce 3.3). Přestože vniklo četné zastoupení různých alternativ, žádná z nich však nepředčila konzistenci ReLU při její aplikaci v různé úlohách. Tyto alternativy byly navrhnuty člověkem, tak aby nejlépe odpovídaly poznatkům o vlastnostech kvalitních aktivačních funkcí.

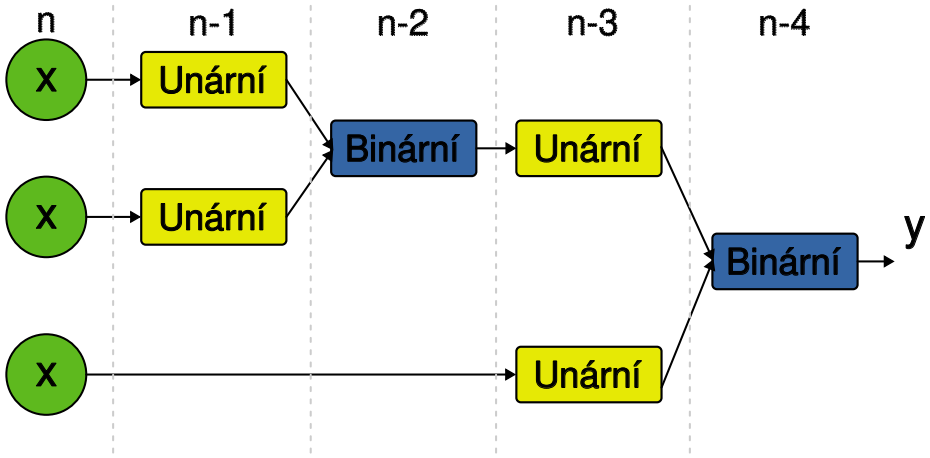
Nicméně díky pokroku v oblasti automatického návrhu neuronových architektur (NAS) a objeveným sítím, které ukázaly, že automatický návrh je efektivní cestou k nalezení architektur překračujících tradiční přístup, se automatický návrh dostává i do oblasti aktivačních funkcí.

Dosavadní práce v oblasti automatizace návrhu aktivačních funkcí spadají do dvou hlavních směrů, především gradientní metody a prohledání prostoru aktivačních funkcí.

Gradientní metody se zaměřují na učitelné parametrické aktivační funkce. Metoda pracuje s funkcemi, jež mají předem definovaný předpis a snaží se nalézt optimální parametry pomocí gradientních metod, které modifikují aktivační funkci [29, 25].

Druhý přístup je založen na tvorbě nových předpisů aktivačních funkcí neboli prohledávání prostoru s kandidátními řešeními. Zejména jsou k tomu využívány metody posilovaného učení a evolučních algoritmů.

Stejně jako u prohledávání architektur NAS je nedílnou součástí automatického návrhu aktivačních funkcí definování prohledávacího prostoru tak, aby prostor obsahoval slibné kandidátní řešení. Důležitou výzvou je opět vyvážení velikosti a vyjadřovací schopnosti prohledávacího prostoru. Obvykle je prohledávací prostor definován jako omezená stromová struktura, což je přirozená reprezentace matematických funkcí. Uzly v tomto stromu jsou unární nebo binární matematické funkce, přičemž listové uzly pak slouží jako vstupy pro aktivační funkce, jak lze vidět na obrázku 4.2. Výsledná stromová struktura, která vznikne,



Obrázek 4.2: Ukázka náhodně vygenerované aktivační funkce z prohledávacího prostoru s unárními a binárními stavebními bloky, kde hloubka stromu je $n = 4$.

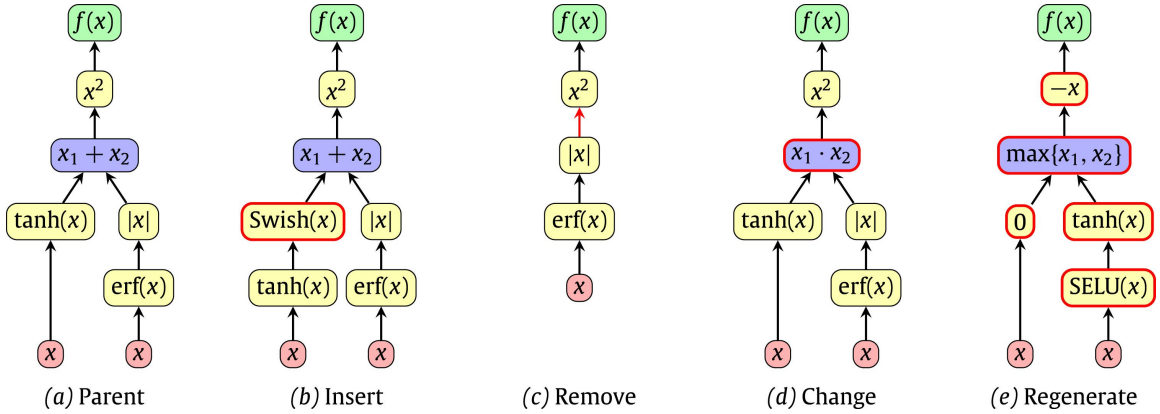
je následně využívána jako výpočetní graf pro generování výstupního skaláru z aktivační funkce.

Jedno z úspěšných uplatnění automatického návrhu aktivačních funkcí, jež využívalo posilované učení, bylo prezentováno v roce 2017 v článku od výzkumníků z Google Brain [27]. Autoři představují několik zajímavých aktivačních funkcí a zejména se však zaměřují na nalezenou funkci nazvanou Swish, definovanou předpisem:

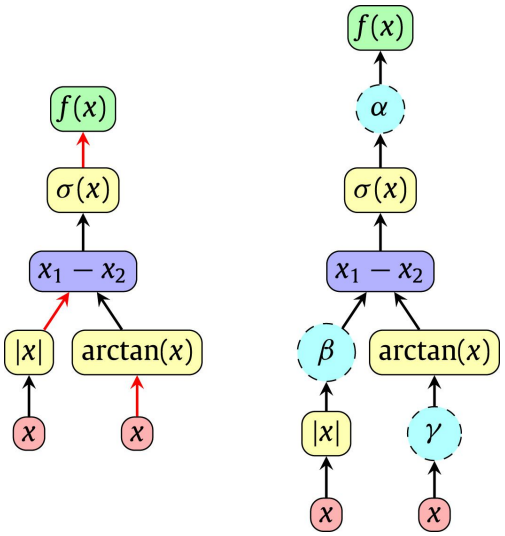
$$f(x) = x \cdot \sigma(\beta x), \tag{4.1}$$

kde β je konstanta nebo trénovací parametr. Při porovnání s ostatními aktivačními funkcemi pouze funkce Swish překonala běžně používané aktivační funkce, jako je ReLU, atd., na několika architekturách na datasetech CIFAR-10, CIFAR-100 a Imagenet. Zde autoři dokázali, že zlepšení architektur pomocí nových aktivačních funkcí jsou možné, ale často vázané pro určitý úkol. Swish získala velkou popularitu v komunitě umělé inteligence a byla začleněna do frameworků pro umělou inteligenci, jako jsou PyTorch a TensorFlow.

Další úspěšné využití automatického návrhu je prezentováno v článku [3], zde autoři prezentují svou techniku PANGAEA (Parametric ActivatioN functions Generated Automatically by an Evolutionary Algorithm), jež kombinuje evoluční algoritmy s gradientní metodou. Jedná se tedy o automatický návrh parametrických aktivačních funkcí, který ke generaci nových jedinců reprezentovaných jako stromová struktura využívá mutačních operátorů, vložení, odstranění, změnu a regeneraci uzlů, které jsou zobrazeny na obrázku 4.3. Po vytvoření nového jedince dochází k parametrizaci aktivační funkce (obrázek 4.4), což umožňuje doladění aktivačních funkcí během tréninku sítě.



Obrázek 4.3: Ukázka reprezentace jedince v PANGAEA (a) a operátorů mutace. Mutace 'vložení': nový uzel je vložen do jedné z hran (b). Mutace 'odstranění': jeden z uzlů grafu je odebrán (c). Mutace 'změna': jeden z uzlů stromu je zaměněn za operátor se stejnou aritou (d). Mutace 'regenerace': každý z uzlů stromu je zaměněn za náhodný operátor (e). Obrázek převzat z [3].



Obrázek 4.4: Ukázka parametrizace jedince v technice PANGAEA. Aktivační funkce je upravena přidáním tří parametrů do hran, čímž vznikne předpis $\alpha\sigma(\beta|x| - tan^{-1}(\gamma x)$. Obrázek převzat z [3].

Kapitola 5

Návrh řešení

Návrh nových aktivačních nelineárních funkcí je nelehký časově náročný úkol. Hlavním důvodem je rozsáhlý téměř nevyčerpatelný prohledávací prostor a další komplexnost do tohoto problému přidává skutečnost, že aktivačních funkce mohou mít různé chování při odlišně zvolených podmínkách.

V této kapitole se nachází popis přístupu prezentovaného v této práci k automatickému návrhu aktivačních funkcí. Pro tento účel jsou vybrány techniky evolučních algoritmů, konkrétně algoritmus kartézské genetické programování, který byl popsán v sekci 2.3. CGP je vhodné pro práci s matematickými výrazy, které reprezentují aktivační funkce. Princip CGP zajišťuje uživatelsky přívětivý výběr a možnost specifikace matematických primitiv, ze kterých jsou aktivační funkce sestaveny, a tím definují prohledávací prostor.

Cílem práce je zkoumání vlivu nových aktivačních funkcí na výkonnost konvolučních neuronových sítí při klasifikační úloze, zejména se zaměřuje na vliv na přesnost klasifikace. Pro tyto experimenty jsou vybrány tři modely konvolučních neuronových sítí s postupně se zvyšující složitostí modelů a datasetů. Prvním z těchto modelů je LeNet-5, který je vybrán pro svou jednoduchou strukturu a rychlou evaluaci. Tento model je v experimentech použit pro klasifikaci vzorů z datasetu MNIST. Další modely jsou model ResNet-10 a WRN-40-10 sekce 3.6, které jsou testovány na datasetech FashionMnist a CIFAR-10, jak jsou postupně napsány. Zmíněné datasety jsou popsány v sekci 3.7.

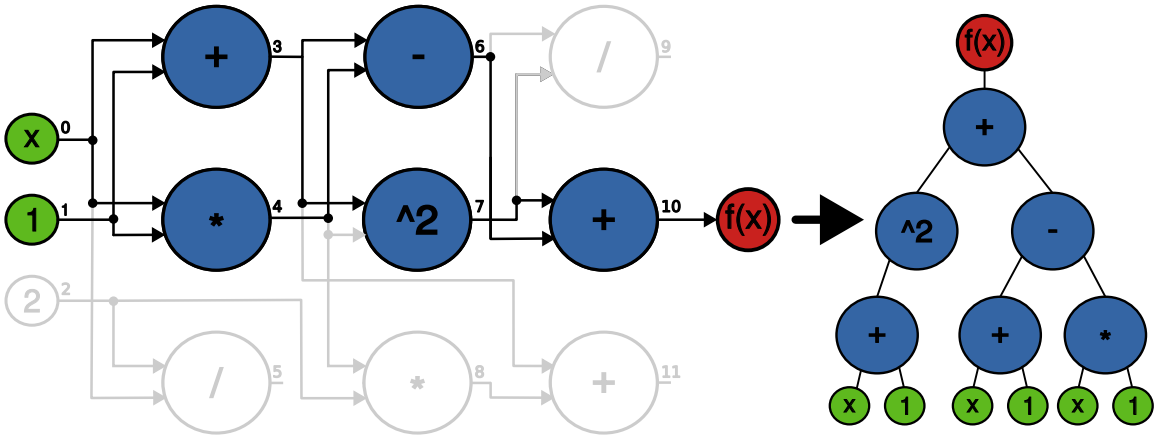
5.1 Návrh aktivačních funkcí pomocí CGP

Pro automatický návrh je vybráno kartézské genetické programování. Navrhované aktivační funkce jsou zde reprezentovány jako acyklické orientované grafy s několika vstupy, kde alespoň jeden z nich reprezentuje mapy příznaků procházející konvoluční neuronovou sítí, na kterých jsou nové kandidátní funkce testovány a ostatní ze vstupů představují zvolené vstupní konstanty. Výpočetní mřížka je základem pro tvorbu kandidátních řešení a současně reprezentuje prohledávací prostor. Tvoří ji výpočetními uzly nabývající unárních nebo binárních matematických operátorů předem definovaných v uživatelské lookup tabulce.

Výsledný acyklický dopředný graf je poté převeden pomocí mapování operátorů z lookup tabulky na výpočetní stromovou strukturu, která slouží k získání matematického předpisu aktivační funkce představující jedince, viz obrázek 5.1.

V implementovaném programu je pro reprezentaci jedinců použito běžné kódování v CGP, jak bylo popsáno v sekci 2.3. Genotyp jedince je reprezentován jako posloupnost $X = (n_{(0,0)}, n_{(0,1)}, n_{(i,j)}, y)$, kde $n_{(i,j)}$ značí n-tici i-tého sloupce a j-tého řádku z mřížky,

n-tice reprezentuje propojení uzlů v mřížce a jeho operátor. Symbol y značí výstupní uzel, ze kterého je očekáván výstupní skalár aktivační funkce.



Obrázek 5.1: Ukázka reprezentace aktivační funkce v CGP, který je tvořen unárními a binárními operátory. Obrázek obsahuje transformaci mřížky na výpočetní graf aktivační funkce $f(x) = (x+1)^2 + (x+1) - (x*1)$.

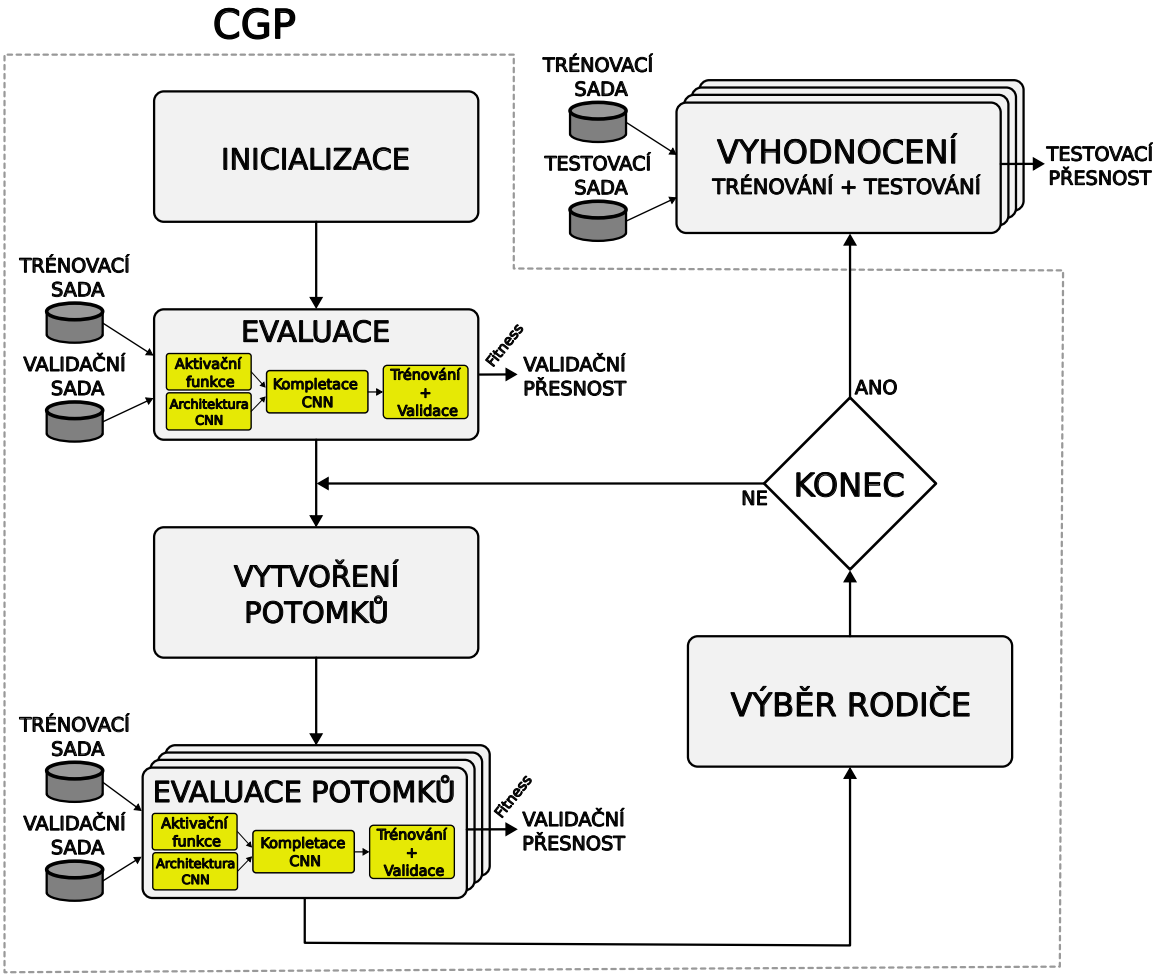
5.1.1 Evoluční návrh

Navržený algoritmus, který je využit pro návrh aktivačních funkcí je zobrazen na obrázku 5.2. Algoritmus využívá prohledávací techniku $1+\lambda$, která je běžně využitá pro kartézské genetické programování.

První krok algoritmu se skládá z úvodní náhodné inicializace populace o velikosti $1+\lambda$. Každý jedinec je převeden na odpovídající aktivační funkci. Tato aktivační funkce je použita namísto standartní aktivační funkce ve zvolené CNN. Po natrénování sítě je stanovena fitness pro kandidátní aktivační funkci jako přesnost klasifikace. Trénování jedinců je po čas evoluce vykonáváno na podmnožině zvolené trénovací datové sady. Zbylá data slouží jako validační část k určení přesnosti, tedy hodnoty fitness kandidátního řešení. Je důležité poznamenat, že pro získání fitness hodnoty kandidátního řešení jsou nejprve všechny aktivační funkce ve vybrané konvoluční neuronové síti nahrazeny kandidátním řešením, a poté proveden trénink po určitý počet trénovacích epoch. Během evolučního návrhu nejsou provedeny žádné další změny v síti nebo v nastavení konfigurace tréninku. Všechny kandidátní řešení mají stejné podmínky vyhodnocení validační přesnosti.

Pro vytváření nových kandidátních řešení je využito elitismu, tedy nejlépe ohodnocený jedinec slouží jako základ pro generování potomků nové generace. Nový jedinci vznikají z kopie rodiče a aplikování několika bodových mutací, které mohou měnit spojení, operátory a výstup mřížky.

Algoritmus je ukončen po dosažení maximálního počtu generací. Po skončení evoluce je nejlépe ohodnocený jedinec vrácen k finálnímu vyhodnocení spolu s dalšími nejlépe hodnocenými řešeními, které jsou uchovány v historii průběhu evoluce (TOP10). Výsledná řešení jsou následně trénována na celém trénovacím datasetu a vyhodnocena na odpovídající testovacím datasetu. Výsledky jsou seřazeny a nejlepší 4 aktivační funkce jsou znovu nezávisle na předchozím tréninku natrénovány a otestovány pro dosažení vyšší statistické spolehlivosti výsledků a omezení šance dosažení výsledků náhodou.



Obrázek 5.2: Schéma evolučního algoritmu pro návrh aktivační funkcí. Míra určující kvalitu kandidátních řešení je stanovena validační přesností.

5.1.2 Mutace

Mutace je jediným evolučním operátorem používaný v implementaci pro vytváření nových jedinců. Funkce je založena na bodové mutaci, která může ovlivnit spojení výpočetní mřížky, funkce uzlů nebo výstupní genom. Každá z těchto mutací má stejnou pravděpodobnost výběru při tvorbě nových potomků. Pro zajištění dostatečné odlišnosti potomků od rodiče existují dva typy této mutace. Jejich realizace se liší v tom, jak je vybrán gen, na který se mutace aplikuje. První způsob spočívá v úpravě aktivního genu, což zajistí přímý vliv na výsledný fenotyp. Druhá z těchto možností zahrnuje náhodný výběr genu, což podporuje neutrální drift, klíčový pro efektivní využití CGP.

Vzhledem k velkému množství duplicitních funkcí nacházejících se v prohledávacím prostoru, jako je například $ReLU(x)$ a $max(x, 0)$, je po aplikaci několika mutací prováděna kontrola, zda nově vzniklá aktivační funkce již nebyla zkoumána během předchozího prohledávání prostoru. Tento proces je zajištěn získáváním výstupního vektoru reprezentujícího aktivační funkci potomka a porovnáním tohoto vektoru s pamětí prohledávání. Pokud je zjištěno, že aktivační funkce již byla dříve zkoumána, je potomek podroben dalším mutacím, dokud není identifikován jako nová funkce.

5.1.3 Náhodné prohledávání

Pro srovnání s evolučního návrhem je implementován a testován algoritmus náhodného prohledávání. Náhodné prohledávání spočívá v nezávislém vytváření nových jedinců, kde každý jedinec je vytvořen generováním celého genotypu. Pro odstranění duplicitních jedinců a pro nejlepší využití výpočetního času je zde opět zaveden mechanizmus kontroly duplicity prohledávaných funkcí, který je zmíněn v sekci 5.1.2. Jedinci jsou vytvářeni v jednotlivých generacích a každý z jedinců je podroben stejně jako u evolučního návrhu tréninku a testování na validační sadě k získání fitness hodnoty jedince. K finálnímu vyhodnocení je opět připuštěno deset nejlépe ohodnocených aktivačních funkcí.

Kapitola 6

Implementace

V této kapitole je stručně popsána implementace realizující návrh zmíněný v předchozí kapitole. Program je implementován v jazyce Python (verze Python 3.10). Pro práci s konvolučními neuronovými sítěmi a matematickými funkcemi realizující kandidátní aktivační funkce byla vybrána knihovna Pytorch[1], která umožňuje jednoduchou a efektivní práci s konvolučními sítěmi a taktéž akceleraci výpočtu na GPU.

Implementace kartézského genetického programování je převzata a upravena z [26]. Hlavní funkčnost programu spočívá v dynamickém vytváření předpisu anonymních funkcí lambda, jež nemají trvalý identifikátor a jsou definovány na místě v programu, kde jsou potřeba. Dále takto definované funkce jsou využity jako aktivační funkce v architektuře při vytváření konvoluční neuronové sítě.

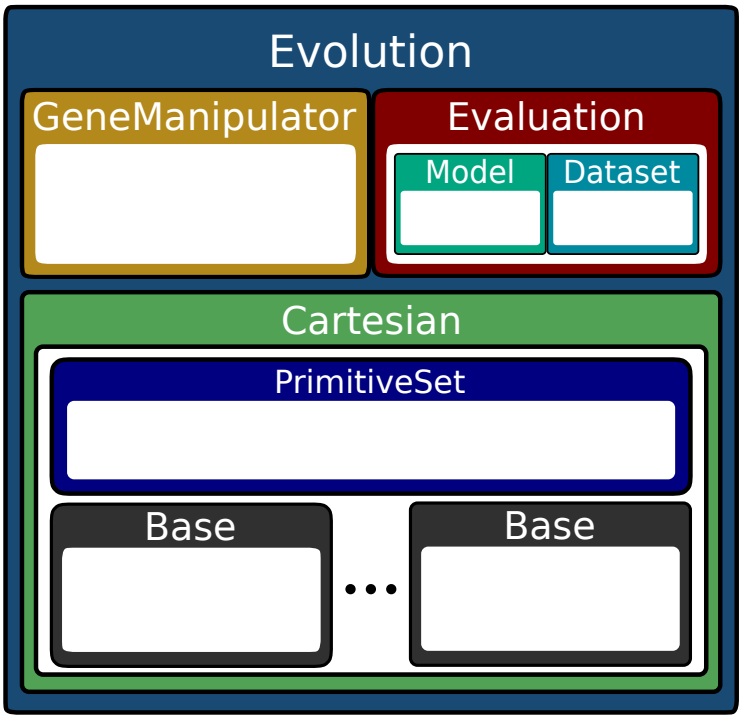
6.1 Struktura programu

Hlavní a nejdůležitější část programu tvoří objekt `Evolution`, který má na starost řízení a průběh evoluce kartézského genetického programování. Objekt se stará také o celkové vyhodnocení nejlepších aktivačních funkcí objektem `Evaluation`. Uvnitř je několik pomocných objektů, které jsou využity při evoluci. Diagram hierarchie jednotlivý objektů je zobrazeno na obrázku 6.1. První z nich je objekt `Base`, který reprezentuje jedno kandidátní řešení. S tímto objektem hlavně interaguje `GeneManipulator`, který má za úkol provádění mutací a převádění genotypu na odpovídající aktivační funkci. Pro převod tento objekt využívá `PrimitiveSet`, který obsahuje mapování na uživatelsky definované matematické primitiva. Poslední důležitým objektem je `Cartesian`, který odpovídá za vstupní definici kartézské mřížky.

6.1.1 Reprezentace jedince

Jedinci jsou reprezentováni třídou `Base`, jak je popsáno návrhu v sekci 5.1, kde je pro kódování jedince zvolena reprezentace ve formě acyklického grafu. Genotyp jedince je zakódován jako c r-rozměrných polí, kde každé pole je o velikosti $1 + max\_arity(PrimitiveSet)$. Číslo 1 značí mapování operátoru, zatímco zbytek výraz označuje maximální aritu operátoru nalezenou ve využité sadě matematických primitiv. Toto slouží k zakódovaní propojení ve mřížce. Symbol c značí počet sloupců a r počet řádků mřížky CGP. Výstup mřížky je oddělen ve své vlastní proměnné.

[1] https://pytorch.org/



Obrázek 6.1: Diagram hierarchie hlavních objektů.

6.1.2 Konvoluční sítě a evaluace

V této sekci je nejdříve popsána implementace datasetů, konvolučních sítí a následně se věnuje implementaci vyhodnocení aktivačních funkcí.

Pro realizaci všech použitých datasetů je využita knihovna Pytorch, která poskytuje všechny námi zvolené datasety pro úlohu klasifikace – MNIST, FascionMNIST, CIFAR-10. Knihovna se stará o stažení potřebných dat a k tomu poskytuje rozhraní pro jejich načítání. K přípravě dat je využita z-skóre normalizace definovaná vztahem 6.1:

$$x_{new} = \frac{(x - \mu)}{\sigma}, \tag{6.1}$$

kde x_{new} označuje nová normalizovaná data a x označuje vstupní data. Symbol μ představuje střední hodnotu dat a σ jejich standardní odchylku. Nad implementací knihovny je vytvořen objekt, který data dále rozdělí do trénovací, validační a testovací sady pro využití při evoluci a finálním vyhodnocení (poměr validační/trénovací sady je jeden ze vstupní parametrů programu). Implementace se nachází ve třídě `Dataset`.

Soubory konvolučních neuronových sítí se nacházejí v adresáři `models`, každý z použitých modelů má oddělený soubor a model WRN-40-4 a ResNet-10, které jsou převzaty z veřejného zdroje[2]. Modely jsou upraveny tak, že při vytváření je nezbytné specifikovat využitou aktivační funkci, aby bylo možné ji dynamicky měnit během evoluce.

Finální vyhodnocovací smyčka konvolučních sítí spolu s vyhodnocením fitness je ve třídě `Evaluation`. K tréninku sítí je využita chybová funkce křížové entropie spolu s optimalizačním algoritmem Adam s učícím krokem 0,01. Optimalizátor Adam je vybrán z důvodu menší náchylnosti na volbu hyperparametrů než jiné optimalizační algoritmy. Při vyhodnocení fitness může nastat situace, že některá navrhnutá aktivační funkce je nestabilní pro trénování,

[2] https://github.com/osmr/imgclsmob

což způsobuje jeho selhání. Pro uspoření výpočetních zdrojů jsou takové aktivační funkce přeskočeny již po první provedené trénovací epoše, trénink a určení fitness se tedy neprovede celé a zanechá se výsledek z první epochy. Stejně je také zacházeno s funkcemi, které po druhé trénovací epoše nepřesáhnou přesnost 0,12, jelikož takové funkce jsou nevhodné pro trénování, a tím síti neumožňují naučit se požadovanou úlohu, nebo by nedosahovaly konkurence schopné fitness hodnoty.

6.2 Definice výpočetních primitiv

Důležitým vstupem programu je definice výpočetních primitiv CGP. Zde si uživatel může určit, s jakými výpočetními uzly bude evoluce aktivačních funkcí probíhat. V kódu 6.1 je předvedena definice několika jednoduchých matematických primitiv. Všechny využité operátory jsou tvořeny jejich Pytorch implementací nebo jejich kombinací tak, aby bylo možné využít jejich knihovní optimalizaci při akceleraci s GPU. Primitiva jsou uložena do proměnné `primitives`, z kterých je vytvořen objekt `PrimitiveSet`.

```
import torch
from primitive import Primitive, Symbol, ConstantValue

# Definice vsupního primitiva s konstatní hodnoutou 0
const = ConstantValue(
            name="c_0",
            value=torch.tensor(0.0)
        )

# Definice binarního  primitiva sčitáni
add = Primitive(
            name="add",
            function=torch.add
            arity=2
        )

# Definice unarního primitiva hyperbolický tangens
tanh = Primitive(
            name="tanh",
            function=torch.tanh,
            arity=1
        )

# Definice vsupní proměnné do aktivační funkce
x = Symbol("x")

primitives = [const, add, tanh, x]
```

Výpis 6.1: Ukázka definice primitiv pro výpočetní mřížku CGP

6.3 Spuštění a běh

V této sekci je popsán postup pro vytvoření prostředí pro běh, základní argumenty a také výstup programu. Pro běh programu je potřeba mít zajištěn Python interpret, nejlépe pro Python3.10. Potřebné knihovny jsou shrnuty v souboru `requirements.txt` a pomocí příkazu:

```
pip install -r requirements.txt
```

je prostředí stáhne a nainstaluje. Hlavní spustitelný soubor programu je `main.py` a obsahuje základní parametry pro spuštění (lze je také vypsat pomocí `-h` nebo `-help`):

- `-m [evo|rand|gt]`, `--mode [evo|rand|gt]` – výber prohledávacího / optimalizačního algoritmu z nasledujících možností:
  - `gt` – testování standardních aktivačních funkcí
  - `evo` – evoluční návrh
  - `rand` - náhodné prohledávání
- `-ds [0|1|2]`, `--dataset [0|1|2]` – výběr použitého datasetu
- `-mdl [0|1|2]`, `--model [0|1|2]` – výběr použité architektury CNN
- `-ep [N]`, `--num_epochs [N]` – počet epoch k celkovému vyhodnocení
- `-evo_ep [N]`, `--num_evo_epochs [N]` – počet epoch k vyhodnocení fitness
- `-gen [N]`, `--num_generation [N]` – počet generací prohledávání
- `-af [0|1]`, `--base_activation [0|1]` – zvolení aktivační funkce při módu GT
- `-ds_r [0-1]`, `-dataset_ratio [0-1]`, – poměr využitých dat
- `-rf [FOLDER]`, `--result_folder [FOLDER]` – zvolení složky ukládání výsledků a průběhu výpočtu

Příklad spuštění evolučního návrhu aktivačních funkcí pomocí příkazu:

```
python main.py -m evo -ds 0 -mdl 0 -ep 30 -evo_ep 3 -gen 250
```

Výstupem programu je poté textový soubor, který obsahuje základní informace o nejlepších aktivačních funkcí výpis 6.2. Další informace o běhu jsou ukládány do souboru `-rf [FOLDER]/logs.pkl`, ze kterého jsou data využita k vizualizaci.

```
##################### Parameters #####################
 Device: cuda
 Dataset name: MNIST
 Train/validation ratio: 0.7
 Dataset ratio: 1.0
 Architecture: LeNet5
 Number of epochs: 30
 Number of evolution epochs: 3
 Number of generations: 250
#################### Top fitness ####################
ID23: sub(erf(add(tanh(x), abs(x))), min(abs(x), c_2))
Fitness: 0.9809444444444444
..
..
################ Results leaderboard ################
#0. ID23: sub(erf(add(tanh(x), add(x, x))), min(add(x, x), c_2))
Results: 0.99279 ....  0.99199
Mean: 0.99174
Std: 0.00156
..
..
#####################################################
Entire run time: 8:13:03.49
```

Výpis 6.2: Ukázka výstupu programu pro evoluční návrh aktivačních funkcí.

Kapitola 7

Experimenty

Tato kapitola obsahuje experimenty, které jsou provedeny s navrženou implementací automatického návrhu aktivačních funkcí. V úvodu kapitoly je popsáno nastavení parametrů experimentů. Dále v kapitole následují výsledky a vyhodnocení experimentů. Na závěr kapitoly je uvedeno možné pokračování práce.

7.1 Nastavení experimentů

Experimenty byly prováděny na superpočítači Karolina prostřednictvím e-INFRA CZ, disponujícím několika grafickými kartami NVIDIA A100, 40 GB RAM, které byly použity pro akceleraci procesu učení CNN. Všechny ostatní výpočty byly prováděny na CPU Intel Xeon-SC 8628, 24jádrový, 2,9 GHz. Díky využití superpočítače bylo možné běh programu několikanásobně urychlit, a tak umožnit testování programu v takovém rozsahu.

Experimenty popsané v této kapitole byly prováděny s následujícími parametry, které jsou uvedeny v tabulce 7.1. Tato tabulka se odkazuje na sadu využitých matematických primitiv, jež definuje prohledávací prostor algoritmu tabulka (7.2). Všechna využitá primitiva v práci mají doménu $\mathbb{R}$, tak aby bylo možné je libovolně skládat. Výběr primitiv byl ovlivněn získanými znalostmi předchozích prací, a to zejména práce [3]. Zde autoři dokázali, že takto definovaný prostor obsahuje kvalitní aktivační funkce, které by pravděpodobně nebyly nalezeny ručně.

7.2 Experimenty

Celkem byly provedeny tři druhy experimentů, každý s dvěma různými sadami primitiv s cílem demonstrovat funkci a výkonnost implementovaného algoritmu. Cílem těchto experimentů bylo ukázat, že program je schopen navrhnout nové aktivační funkce při různé složitosti architektury CNN a obtížnosti testovaného datasetu. Návrh byl vyzkoušen se sadou primitiv popsanou v tabulce 7.2 (dále se nazývá sada primitiv A) a také s její upravenou verzí s vynecháním konstant (dále se nazývá sada primitiv B). Využití sady bez konstant podporuje větší variabilitu v průběhů funkcí při návrhu, protože mutace mají výraznější dopad na celkový tvar funkce. Evoluční návrh s upravenou sadou primitiv je vyzkoušen na podmnožině dat z datasetu o velikost 50% a 20% celkového datasetu. Nově vzniklé aktivační funkce by měly plně využívat vlastnosti architektury modelu, a tím vylepšit celkovou funkci při klasifikaci.

Pro porovnání evolučního návrhu je testována i verze s náhodným prohledáváním aktivačních funkcí. Z důvodu značné výpočetní náročnosti byly vždy provedeny pouze dva běhy evolučního návrhu a jeden náhodného prohledávání pro každý druh experimentu s jednotlivými sadami, ze kterých jsou reportovány výsledky níže.

Pro první experiment byla vybrána velmi jednoduchá CNN, která se dá považovat jako *Hello World* oblasti hlubokého učení. Konkrétně architektura LeNet-5 s testovanou datovou sadou MNIST. Dataset MNIST je velmi jednoduchá datová sada a přesnosti zde dosahují téměř 100% (aktuálně nejlépe dosažený výsledek 99,87%[1]), což znamená, že zde není velký prostor k zlepšení. Nicméně tento experiment vystavuje zkoumaný evoluční návrh do zajímavé pozice, kde se algoritmus musí snažit vylepšit téměř dokonale přesnou CNN.

Další z experimentů je kombinace architektury ResNet-10 s datasetem FashionMNIST, jež je složitější kombinaci předchozích. Finální experiment, a také nejsložitější z nich, je kombinace architektury WRN-40-4 a datasetu CIFAR-10.

Nastavení experimentů	
Parametry evolučního algoritmu	
Počet generací	250/125/100
Velikost populace	4
Počet mutací	3
Parametry evaluace jedince	
Velikost trénovací sady	70 %
Velikost validační sady	30 %
Počet epoch	3/8/18
Parametry CGP	
Počet řádků	3
Počet sloupců	10
L-back	10
Sada primitiv	Tabulka 7.2
Parametry vyhodnocení	
Velikost trénovací sady	100 %
Velikost testovací sady	100 %
Počet epoch	30/72/150

Tabulka 7.1: Vstupní parametry experimentů. Pokud je na řádku tabulky uvedeno více hodnot tak hodnoty značí parametry využité v experimentech, jak jsou postupně napsány za sebou: LeNet-5 & MNIST, ResNet-10 & FascionMNIST, WRN-40-4 & CIFAR-10.

Sada primitiv						
Konstanty	**Unární**				**Binární**	
0	x	$-x$	$ReLU(x)$	$Softplus(x)$	$x+y$	$x-y$
1	x^2	$\|x\|$	$min(x,0)$	$Softsign(x)$	x/y	$x*y$
2	e^x	$\sigma(x)$	$sinh^{-1}(x)$	$erf(x)$	$max(x,y)$	
	$tanh(x)$	$log(\sigma(x))$	$tan^{-1}(x)$		$min(x,y)$	

Tabulka 7.2: Sada primitiv využitá pro experimenty.

[1] https://paperswithcode.com

7.3 Výsledky experimentů

Výstupem z každého běhu automatického návrhu aktivačních funkcí je množina nejlépe fungujících jedinců. Z této množiny jsou vybráni jedinci, kteří dosáhli nejlepší přesnosti při finálním vyhodnocení. Výsledky s oběma sadami primitiv pro jednotlivé experimenty jsou zaznačeny v tabulkách 7.4, 7.6, 7.8. Aktivační funkce, které dosáhly nejlepších výsledků, mají své průběhy zobrazeny v tabulkách 7.5, 7.7, 7.9. Výpočetní časy jednotlivých experimentů s celou datovou sadou jsou zaznačeny v tabulce 7.3.

Ve výsledcích je uveden předpis jedince, který je zjednodušen pro prezentaci výsledků. Tedy jsou vynechány části, které nemají vliv na výstup funkce. Dále je zde reportována dosažená fitness hodnota při návrhu, a také finální přesnost (vše uvedeno v %). Obě tyto reportované hodnoty jsou zaokrouhleny na čtyři desetinná místa a převedeny na procenta.

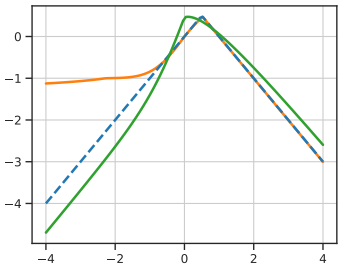
Nejlépe dosažená přesnost experimentu je ve výsledcích označena tučně. Hvězdičky v sloupci přesnosti označují výsledky statistického testu "průměrné zlepšení přesnosti oproti ReLU"testováno jednovýběrovým Welchových t-testem. Znak $*$, značí pokud signifikance $p \leq 0,05$, znaky $**$ značí pokud $p \leq 0,01$, a znaky $***$ značí pokud $p \leq 0,001$.

Doba evoluce experimentů	
Experiment	**Doba výpočtu (hh:mm:ss)**
Exp. 1 – LeNet-5 & MNIST	04:02:19
Exp. 2 – ResNet-10 & FashionMNIST	08:54:37
Exp. 3 – WRN-40-4 & CIFAR-10	11:01:36

Tabulka 7.3: Průměrný čas evolučního návrhu s využitím celého datasetu. Do těchto časů se nezapočítává doba strávená ve finálním vyhodnocení.

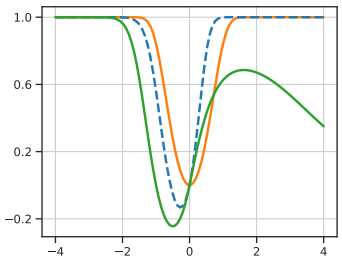
Výsledky experimentu LeNet-5 & MNIST		
Sada primitiv A	**Fitness (%)**	**Přesnost (%)**
Evoluční návrh		
$min(erf(x), min(erf(2-x), tan^{-1}(sinh^{-1}(x))))$	98,25	**99,28**±0,07 ***
$min(x, 1-x)$	97,82	99,26±0,04 ***
$min(erf(x), min(erf(2-x), sinh^{-1}(sinh^{-1}(x))))$	98,21	99,23±0,05 ***
Náhodné prohledávání		
$erf(x^2)$	96,53	99,26±0,04 ***
$Softsign((x+1)*x)$	97,18	99,21±0,07 ***
Sada primitiv B		
Evoluční návrh 50% datasetu		
$tanh(erf(sinh^{-1}(x)) - x^2)$	95,33	99,19±0,08 ***
$tanh(tan^{-1}(x)) + erf(x^2))$	95,32	99,19±0,06 ***
Náhodné prohledávání 50% datasetu		
$tanh((-x)^2 + x/e^{-x})$	94,43	99,27±0,08 ***
$tanh((x^2+x)/e^{ReLU(x)})$	95,58	99,25±0,06 ***
Evoluční návrh 20% datasetu		
$tan^{-1}(\sigma(Softsign(x)))+min(Softsign(x), tanh(x))-\|x\|$	92,38	99,24±0,08 ***
$tanh(tan^{-1}(x) + x^2)))$	88,67	99,23±0,07 ***
Náhodné prohledávání 20% datasetu		
$tan^{-1}(tanh(x)) - tanh(tan^{-1}(x^2))))$	88,96	99,24±0,06 ***
$sinh^{-1}(-x - x^2)$	89,96	99,23±0,06 ***
Standard		
ReLU		99,06±0,09
PReLU		99,09±0,06 *

Tabulka 7.4: Přehled nejlepších evolučně navržených aktivačních funkcí pro experiment LeNet-5 & MNIST. Výsledky přesnosti jsou uváděny jako střední hodnota ± standardní odchylka získána z deseti nezávislých běhů s aktivační funkcí.



Evoluční návrh

- $min(\text{erf}(x), \min(\text{erf}(2-x), \tan^{-1}(\sinh^{-1}(x))))$
- $min(x, 1-x)$
- $tan^{-1}(\sigma(\text{Softsign}(x)))+\min(\text{Softsign}(x), \tanh(x))-|x|$



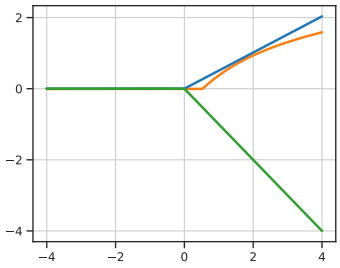
Náhodné prohledávání

- $erf(x^2)$
- $tanh((-x)^2 + x/e^{-x})$
- $tanh((x^2+x)/e^{ReLU(x)})$

Tabulka 7.5: Průběhy nejlepších aktivačních funkcí nalezených v experimentech LeNet-5 & MNIST.

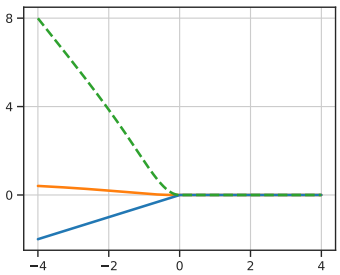
Výsledky experimentu ResNet-10 & FashionMNIST		
Sada primitiv A	**Fitness (%)**	**Přesnost (%)**
Evoluční návrh		
$max(sinh^{-1}(x), 0.5) - tan^{-1}(1 + tan^{-1}(log(\sigma(0.5))))$	90,58	93,35±0,07 **
$max(sinh^{-1}(0), x) * tan^{-1}(1 + tan^{-1}(log(\sigma(0.5))))$	90,79	93,34±0,16 *
Náhodné prohledávání		
$\lvert(min(Softsign(x), 0)^2)^2\rvert$	91,32	**93,43**±0,13 **
$min(x/2, Softsign(0))$	89,82	93,23±0,10
Sada primitiv B		
Evoluční návrh 50% datasetu		
$ReLU(max(max(ReLU(x), e^x), x)) - max(ReLU(x), e^x)$	88,87	93,14±0,24
$ReLU(tanh(e^{ReLU(x)})) - ReLU(x))$	88,82	93,12±0,10
Náhodné prohledávání 50% datasetu		
$min(x, 0) * tanh(min(x, 0))$	88,53	93,17±0,19
$((x - ReLU(x)) + (x - ReLU(x))) * tanh(x)$	87,81	93,17±0,15
Evoluční návrh 20% datasetu		
$-ReLU(x)$	84,75	93,36±0,15 *
$min(x, 0) * tanh(erf(log(\sigma(x)) * x)))$	82,46	93,25±0,18
Náhodné prohledávání 20% datasetu		
$min(x, 0)/e^{(erf(Softplus(sinh^{-1}(x))))}$	84,95	93,09±0,22
$x * tan^{-1}(Softplus(x))$	84,04	92,92±0,11
Standard		
ReLU		93,14±0,17
PReLU		93,01±0,29

Tabulka 7.6: Přehled nejlepších evolučně navržených aktivačních funkcí pro experiment ResNet-10 & FashionMNIST. Výsledky přesnosti jsou uváděny jako střední hodnota ± standardní odchylka získána z šesti nezávislých běhů s aktivační funkcí.



Evoluční návrh

- $max(sinh^{-1}(x), 0.5) - tan^{-1}(1 + tan^{-1}(log(\sigma(0.5))))$
- $max(sinh^{-1}(0), x) * tan^{-1}(1 + tan^{-1}(log(\sigma(0.5))))$
- $-ReLU(x)$



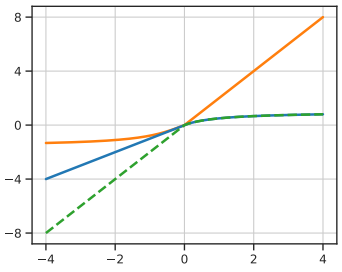
Náhodné prohledávání

- $\lvert(min(Softsign(x), 0)^2)^2\rvert$
- $min(x/2, Softsign(0))$
- $((x - ReLU(x)) + (x - ReLU(x))) * tanh(x)$

Tabulka 7.7: Průběhy nejlepších aktivačních funkcí nalezených v experimentech ResNet-10 & FashionMNIST

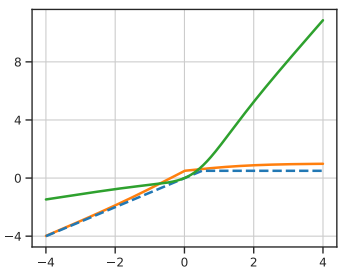
Výsledky experimentu WRN-40-4 & CIFAR-10		
Sada primitiv A	**Fitness (%)**	**Přesnost (%)**
Evoluční návrh		
$min(Softsign(x), min(1,x))$	78,75	84,77±0,32 ***
$min(Softsign(x), x+x)$	78,24	84,72±0,14 **
$tanh(\|Softplus(x)\|) * x$	79,96	84,18±0,73
Náhodné prohledávání		
$min(min(x, \sigma(0)), \sigma(0) + erf(1))$	73,83	84,47±0,26 **
$min(-x, tan^{-1}(1))$	74,18	83,97±0,41
Sada primitiv B		
Evoluční návrh 50% datasetu		
$max(tan^{-1}(x), x+x)$	68,80	**84,95**±0,37 ***
$max(0, max(erf(x), x))$	64,68	83,67±0,66
Náhodné prohledávání 50% datasetu		
$min(x, Softsign(x)$	68,26	84,40±0,45 *
$e^{tanh(x)} * x$	68,60	84,23±0,23 **
Evoluční návrh 20% datasetu		
$min(x - erf(ReLU(x), 0))$	54,15	83,85±0,50
$ReLU(x+x)$	57,10	83,84±0,15
Náhodné prohledávání 20% datasetu		
$min(x, min(0, \sigma(x)))) + \sigma(x)$	56,35	84,63±0,47 **
$x + \|x\|$	54,85	83,69±0,44
Standard		
ReLU		83,33±0,68
PReLU		83,60±0,48

Tabulka 7.8: Přehled nejlepších evolučně navržených aktivačních funkcí pro experiment WRN-40-4 & CIFAR-10. Výsledky přesnosti jsou uváděny jako střední hodnota ± standardní odchylka získána ze čtyřech nezávislých běhů s aktivační funkcí.



Evoluční návrh

- $max(tan^{-1}(x), x+x)$
- $min(Softsign(x), min(1,x))$
- $min(Softsign(x), x+x)$



Náhodné prohledávání

- $min(x, min(0, \sigma(x)))) + \sigma(x)$
- $min(min(x, \sigma(0)), \sigma(0) + erf(1))$
- $e^{tanh(x)} * x$

Tabulka 7.9: Průběhy nejlepších aktivačních funkcí nalezených v experimentech WRN-40-4 & CIFAR-10.

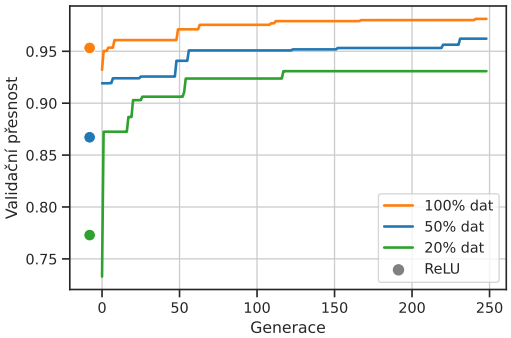
7.4 Zhodnocení experimentů

Z výsledků sady experimentů vyplývá, že evoluční návrh dokáže najít aktivační funkce, které významně zlepšují přesnost klasifikace ve srovnání s ReLU při různých úrovních složitosti modelů a testovaných datasetů. Tuto schopnost lze pozorovat i na průbězích fitness hodnot během nejlepšího běhu evoluce, viz obrázky 7.1, kde evoluční návrh dokáže překonat ReLU během několika generací, což naznačuje možnost nalezení slibných aktivačních funkcí.

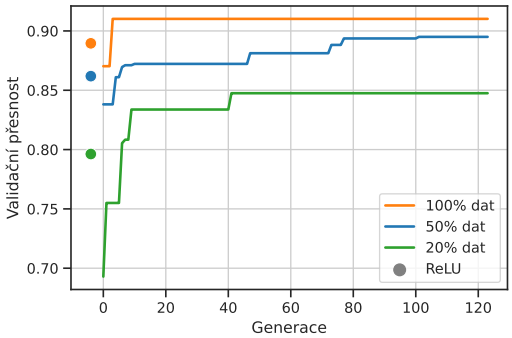
Automatické návrhy prokázaly schopnost nalézt kvalitní aktivační funkce s oběma sadami primitiv, Avšak u sady primitiv B se zmenšením testovaného datasetu je kvalitních aktivačních funkcí méně z důvodu výrazné změny navrhovaných a testovaných podmínek v datasetu, jež vede k dobré funkci při návrhu, ale horší funkci při finálním vyhodnocení. Rovněž zmenšení datasetu přidává další nestabilitu pro testované aktivační funkce.

Z výsledků je také patrné, že dosažení nejlepší fitness hodnoty nemusí nutně znamenat dosažení nejlepší finální přesnosti. Proto je klíčové přehodnocení finálních výsledků a zaměření se pouze na nejlepší dosažené řešení.

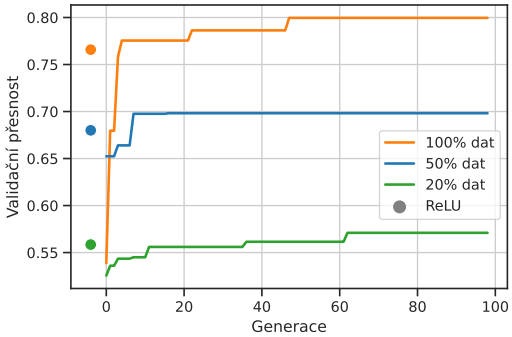
Implementované náhodné prohledávání je také schopno najít kvalitní aktivační funkce vylepšující přesnost oproti běžně využívané ReLU. Nicméně, jak ukazují boxploty na obrázku na obrázku 7.2, metoda evolučního návrhu dosahuje vyšší průměrnou fitness hodnotu z vyhodnocených běhů.



(a) LeNet-5 & MNIST

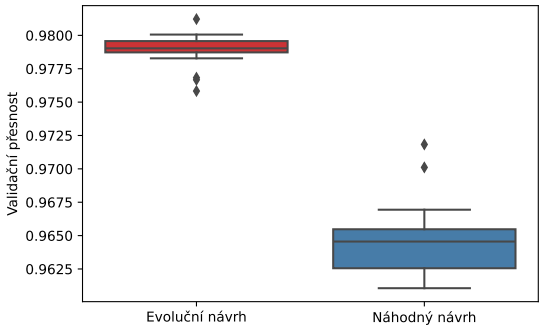


(b) ResNet-10 & FashionMNIST

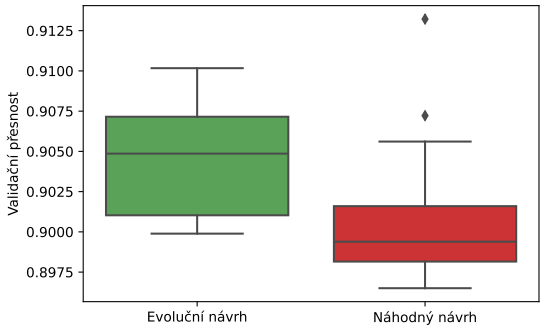


(c) WRN-40-4 & CIFAR-10

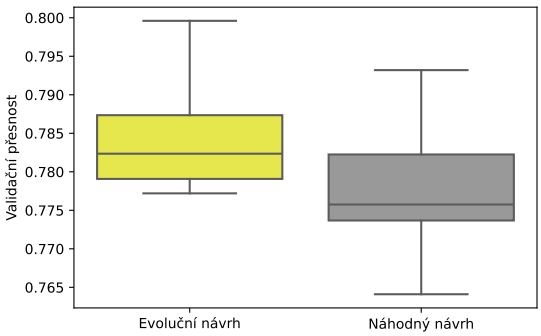
Obrázek 7.1: Průběhy evoluce aktivčních funkcí s využitím různého procenta dat. Barevné body ukazují fitness hodnotu funkce ReLU testovanou na daném procentu dat.



(a) LeNet-5 & MNIST



(b) ResNet-10 & FashionMNIST



(c) WRN-40-4 & CIFAR-10

Obrázek 7.2: Porovnání dosažených fitness hodnot evolučního návrhu a náhodného prohledávání aktivačních funkcí (sada primitiv A, celý dataset).

V prvním testovaném experimentu LeNet-5 & MNIST bylo navrženo několik aktivačních funkcí, které statisticky významně vylepšili přesnost oproti ReLU. Nejlepší z navržených aktivačních funkcí dosáhla průměrné přesnosti 99,28% a zlepšila přesnost o 0,22% ve srovnání s ReLU.

Ve druhém experimentu s ResNet-10 a FashionMnist byly navrženy čtyři aktivační funkce, které přinesly statisticky významné zlepšení v průměrné přesnosti ve srovnání s ReLU. Nejlepší aktivační funkce vzešlá z náhodného prohledávání dosáhla průměrné přesnosti 93,43% a zlepšila průměrnou přesnost o 0,29%. Nejlepší aktivační funkce z evolučního návrhu dosáhla průměrné přesnosti 93,35% a zlepšila průměrnou přesnost o 0,21%.

V posledním experimentu byl testován program s architekturou WRN-40-4 a datasetem CIFAR-10. Podle tabulky 7.10 srovnání metod návrhu aktivačních funkcí s prezentovanou metodou není přímo možné, z důvodu zvolení odlišné architektury CNN pro testování na datasetu CIFAR-10. Jediný z výsledků, který možno implementaci porovnat je výsledek standartního využití WRN-40-4, který dosahuje přesnosti 95,03%, a jež ukazuje, že aktuální řešení zatím nedosahuje srovnatelných výsledků, avšak se jim částečně přibližuje. Hlavními důvody jsou, že implementovaný program přistupuje ke všem architekturám s různými aktivačními funkcemi stejně, aby neprioritizoval žádnou z aktivačních funkcí. Tento přístup má za následek, že trénování těchto jedinců není optimální. Dále nejsou využity žádné další techniky, které by pomohly vylepšit přesnost odhadu, jako je optimalizace hyperparametrů nebo augmentace dat.

V tomto experimentu bylo navrženo sedm aktivačních funkcí, které statisticky zlepšují přesnost odhadu oproti ReLU za těchto podmínek. Nejlepší aktivační funkce z evolučního návrhu dosáhla průměrné přesnosti 84,95% a zlepšila odhad o 1,62% ve srovnání s ReLU.

Srovnání algoritmů pro CIFAR-10		
Metody	**Aktivační funkce**	**Přesnost (%)**
WRN-40-4 [32]	$ReLU(x)$	95.03
RNN[27]	$x * \sigma(\beta x)$	95.50
PANGAEA [3]	$\alpha ReLU(\beta \lvert ReLU(\gamma x) \rvert)$	92.77
EA [2]	$e^{min(erf(x),0)-max(x,0)} * min(tan^{-1}(x^3) * max(\lvert x \rvert, 0), 0)$	94.10

Tabulka 7.10: Srovnání přístupu automatického návrhu aktivačních funkcí.

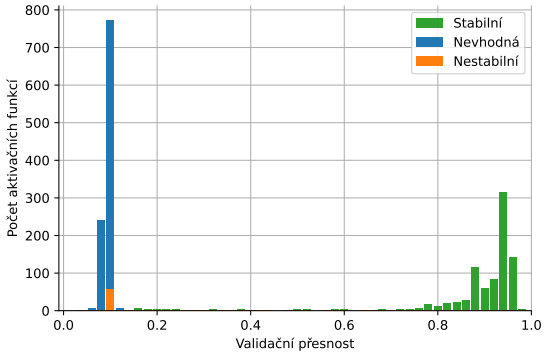
7.5 Pokračování práce

Z provedených experimentů je patrné, že implementovaný program s uživatelsky definovanou sadou primitiv je schopen nalézt aktivační funkce významně vylepšující přesnost odhadu oproti zvolené referenční aktivační funkci ReLU. Díky uživatelské definici sady primitiv je možné jednoduše parametrizovat návrh a tím i jednoduše zkoumat různé definice zajímavých prohledávaných prostorů, které by mohly zahrnovat kvalitní aktivační funkce, jež by jinak nemohly vniknout ručně. Tedy implementovaný program představuje nástroj umožňující další zkoumání v tomto ohledu.

Dále se zde nabízí několik možností pro rozšíření a zkoumaní stávající implementace či celého tématu automatického návrhu aktivačních funkcí. Jednou z možných rozšíření stávající implementace, která by pomohla dosaženým lepších konkurenčních výsledků navrhnutých řešení je využití pokročilých technik učení konvoluční neuronových sítí, jako je využití augmentace dat nebo regularizace.

Jako další zajímavé rozšíření se jeví být kombinace evolučního návrhu s gradientní metodou návrhu, která by umožnila další možnost modifikace nalezených aktivačních funkcí a větší schopnost přizpůsobení aktivačních funkcí pro jednotlivé vrstvy architektury CNN.

Jednou z oblastí výzkumu tohoto tématu, která se zdá být klíčová pro zefektivnění návrhu, je predikce odhadu vhodnosti aktivační funkce. Tímto způsobem lze zrychlit alespoň identifikaci jedinců, kteří nejsou vhodní pro použití v CNN. Jak ukazuje obrázek 7.3 sestavený ze všech dosažených fitness hodnot v evolučního experimentu LeNet-5 & MNIST ze dvou běhů, je téměř 50% zkoumaných aktivačních funkcí nevhodných pro učení CNN, a tedy se i při přeskočení po druhé epoše (viz sekce 6.1.2), jak je implementováno v prezentované práci, plýtvá GPU výkonem na neperspektivních aktivačních funkcích, který by se dal uplatnit lépe.



Obrázek 7.3: Ukázka všech dosažených fitness hodnot ze dvou evolučních běhů experimentu LeNet-5 & MNIST.

Kapitola 8

Závěr

Cílem této práce bylo seznámit se, navrhnout, implementovat a pomocí experimentů otestovat evoluční návrh aktivačních funkcí využitých v konvolučních neuronových sítích. Hlavním zaměřením bylo dosáhnout vylepšení přesnosti odhadu klasifikace obrazu CNN oproti běžně využívané aktivační funkci ReLU. Pro účely evolučního návrhu bylo vybráno kartézské genetické programování.

V první části práce byly podrobně shrnuty základní teoretické znalosti z oblasti konvolučních neuronových sítí, evolučních algoritmů a jejich spojení se zaměřením na aktuální situaci v návrhu aktivačních funkcí potřebné pro pochopení této práce.

Následně bylo provedeno několik experimentů s prezentovanou implementací pro otestování funkčnosti a ověření zamýšlených cílů. Z dosažených výsledků je patrné, že prezentovaný program je schopen navrhnout aktivační funkce, které statisticky významně zlepšují přesnost odhadu zvolených modelů CNN oproti referenční ReLU. To bylo dokázáno ve třech experimentech se zvětšující se složitostí architektury CNN a testované datové sady. První a nejjednodušší experiment LeNet-5 & MNIST dosáhl průměrné přesnosti 99,28% a zlepšil tak přesnost o 0,22% ve srovnání s ReLU. Další z experimentů je ResNet-10 & FashionMnist, který dosáhl průměrné průměrné přesnosti 93,35% a zlepšil průměrnou přesnost o 0.21%. Nejtěžší experiment a také experiment, který dosáhl největšího zlepšení přesnosti je WRN-40-4 & CIFAR-10. Zde byla nalezena aktivační funkce, která dosáhla průměrné přesnosti 84,95% a zlepšila tak odhad o 1,62% ve srovnání s ReLU.

V rámci této práce byl rovněž vyzkoušeno zrychlení návrhu aktivačních funkcí, jež spočívá ve využití menšího množství testovaných dat v části automatického návrhu. Tento přístup se také osvědčil a umožňuje zrychlené nalezení aktivačních funkcí, jež vylepšují přesnost odhadu testovaného modelu na datové sadě.

Literatura

[1] Aghdam, H. H. a Heravi, E. J. *Guide to Convolutional Neural Networks: A Practical Application to Traffic-Sign Detection and Classification*. 1st. Springer Publishing Company, Incorporated, 2017. ISBN 331957549X.

[2] Bingham, G., Macke, W. a Miikkulainen, R. Evolutionary Optimization of Deep Learning Activation Functions. *CoRR*. 2020, abs/2002.07224. Dostupné z: https://arxiv.org/abs/2002.07224.

[3] Bingham, G. a Miikkulainen, R. Discovering Parametric Activation Functions. *Neural Networks*. Elsevier BV. duben 2022, sv. 148, s. 48–65. DOI: 10.1016/j.neunet.2022.01.001. ISSN 0893-6080. Dostupné z: http://dx.doi.org/10.1016/j.neunet.2022.01.001.

[4] Bremermann, H. J. et al. Optimization through evolution and recombination. *Self-organizing systems*. Washington, DC: Spartan. 1962, sv. 93, s. 106.

[5] Brownlee, J. *Clever algorithms : nature-inspired programming recipes*. Lulu, 2011. ISBN 1446785068.

[6] Commons, W. *Biologický neuron*. 2006. Dostupné z: https://commons.wikimedia.org/wiki/File:Neuron_(cesky)-1.svg.

[7] Cybenko, G. Approximation by Superpositions of a Sigmoidal Function. *Mathematics of Control, Signals, and Systems*. 1989, sv. 2, č. 4, s. 303–314. DOI: 10.1007/BF02551274.

[8] Darwin, C., John, M., Clowes, W., Sons, Bradbury et al. *On the origin of species by means of natural selection, or, The preservation of favoured races in the struggle for life*. London John Murray, Albemarle Street 1859. Dostupné z: https://www.biodiversitylibrary.org/item/122307.

[9] Eiben, A. E. a Smith, J. E. *Introduction to Evolutionary Computing*. 2nd. Springer Publishing Company, Incorporated, 2015. ISBN 3662448734.

[10] Elsken, T., Metzen, J. H. a Hutter, F. Neural Architecture Search: A Survey. *J. Mach. Learn. Res.* 2019, sv. 20, s. 55:1–55:21. Dostupné z: http://jmlr.org/papers/v20/18-598.html.

[11] Esfahanian, P. a Akhavan, M. GACNN: Training Deep Convolutional Neural Networks with Genetic Algorithm. *CoRR*. 2019, abs/1909.13354. Dostupné z: http://arxiv.org/abs/1909.13354.

[12] Friedberg, R. M. A Learning Machine: Part I. *IBM J. Res. Dev.* 1958, sv. 2, s. 2–13.

[13] Gong, J., Liu, W., Pei, M., Wu, C. a Guo, L. ResNet10: A lightweight residual network for remote sensing image classification. In: *2022 14th International Conference on Measuring Technology and Mechatronics Automation (ICMTMA)*. 2022, s. 975–978. DOI: 10.1109/ICMTMA54903.2022.00197.

[14] Hebb, D. O. *The organization of behavior: A neuropsychological theory* [Hardcover]. New York: Wiley, červen 1949. ISBN 0-8058-4300-0.

[15] Hornik, K., Stinchcombe, M. a White, H. Multilayer feedforward networks are universal approximators. *Neural Networks*. 1989, sv. 2, č. 5, s. 359–366. ISSN 0893-6080. Dostupné z: https://www.sciencedirect.com/science/article/pii/0893608089900208.

[16] Ioffe, S. a Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In: Bach, F. R. a Blei, D. M., ed. *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*. JMLR.org, 2015, sv. 37, s. 448–456. JMLR Workshop and Conference Proceedings. Dostupné z: http://proceedings.mlr.press/v37/ioffe15.html.

[17] Jagtap, A. D. a Karniadakis, G. E. HOW IMPORTANT ARE ACTIVATION FUNCTIONS IN REGRESSION AND CLASSIFICATION? A SURVEY, PERFORMANCE COMPARISON, AND FUTURE DIRECTIONS. *Journal of Machine Learning for Modeling and Computing*. 2023, sv. 4, č. 1, s. 21–75. ISSN 2689-3967.

[18] Krizhevsky, A., Sutskever, I. a Hinton, G. E. ImageNet Classification with Deep Convolutional Neural Networks. In: Pereira, F., Burges, C. J. C., Bottou, L. a Weinberger, K. Q., ed. *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2012, s. 1097–1105. Dostupné z: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[19] KumarI, A. *Different types of CNN Architectures explained: Examples*. Apr 2022. Dostupné z: https://vitalflux.com/different-types-of-cnn-architectures-explained-examples/.

[20] LeCun, Y., Bottou, L., Bengio, Y. a Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*. Ieee. 1998, sv. 86, č. 11, s. 2278–2324.

[21] Miller, J. a Smith, S. Redundancy and computational efficiency in Cartesian genetic programming. *IEEE Transactions on Evolutionary Computation*. 2006, sv. 10, č. 2, s. 167–174. DOI: 10.1109/TEVC.2006.871253.

[22] Miller, J. F., ed. *Cartesian Genetic Programming*. Springer, 2011. Natural Computing Series. ISBN 978-3-642-17309-7. Dostupné z: https://doi.org/10.1007/978-3-642-17310-3.

[23] Miller, J. F. et al. An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach. In: *Proceedings of the genetic and evolutionary computation conference*. 1999, sv. 2, s. 1135–1142.

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

[25] Molina, A., Schramowski, P. a Kersting, K. Padé Activation Units: End-to-end Learning of Flexible Activation Functions in Deep Networks. *CoRR*. 2019, abs/1907.06732. Dostupné z: http://arxiv.org/abs/1907.06732.

[26] Quade, M. *Cartesian* [https://github.com/Ohjeah/cartesian]. 2020.

[27] Ramachandran, P., Zoph, B. a Le, Q. V. Searching for Activation Functions. *CoRR*. 2017, abs/1710.05941. Dostupné z: http://arxiv.org/abs/1710.05941.

[28] Stanley, K. O. a Miikkulainen, R. Evolving Neural Networks through Augmenting Topologies. *Evolutionary Computation*. 2002, sv. 10, č. 2, s. 99–127. DOI: 10.1162/106365602320169811.

[29] Tavakoli, M., Agostinelli, F. a Baldi, P. SPLASH: Learnable Activation Functions for Improving Accuracy and Adversarial Robustness. *CoRR*. 2020, abs/2006.08947. Dostupné z: https://arxiv.org/abs/2006.08947.

[30] University, S. *CS231n: Convolutional Neural Networks for Visual Recognition* [online]. [cit. 2024-5-15]. Dostupné z: https://cs231n.github.io/.

[31] White, C., Safari, M., Sukthanker, R., Ru, B., Elsken, T. et al. Neural Architecture Search: Insights from 1000 Papers. *CoRR*. 2023, abs/2301.08727. DOI: 10.48550/ARXIV.2301.08727. Dostupné z: https://doi.org/10.48550/arXiv.2301.08727.

[32] Zagoruyko, S. a Komodakis, N. Wide Residual Networks. In: Wilson, R. C., Hancock, E. R. a Smith, W. A. P., ed. *Proceedings of the British Machine Vision Conference 2016, BMVC 2016, York, UK, September 19-22, 2016*. BMVA Press, 2016. Dostupné z: http://www.bmva.org/bmvc/2016/papers/paper087/index.html.

Příloha A

Obsah přiloženého paměťového média

Na přiloženém paměťovém médiu k této diplomové práci se nacházejí všechny zdrojové kódy implementovaného programu, výsledky experimentů a návod na práci s programem. Dále jsou zde přiloženy zdrojové soubory textové části této práce v jazyce LaTeX. Obsah paměťového média je rozdělen následovně.

Ve složce `src/` jsou uloženy zdrojové kódy implementovaného programu v jazyce Python. V adresáři `experiments/` se nacházejí výsledky provedených experimentů a v adresáři `thesis/` zdrojové soubory této práce. Soubor `README.md` obsahuje manuál pro práci s programem. Adresářová struktura paměťového média je následující:

```
./xhladi23
├── src/
├── experiments/
├── thesis/
└── README.md.
```