Středoškolská technika 2017

Setkání a prezentace prací středoškolských studentů na ČVUT

Ukázka použití novelty learningu v kombinaci s objective-based learningem

Antonín Říha

Vyšší odborná škola a SPŠ, Jičín

Pod Koželuhy 100, Jičín

Obsah

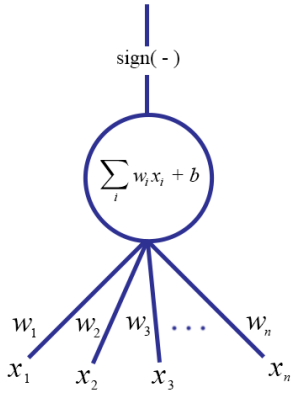
Úvod

Umělá inteligence je aktuálně jedním z nejrychleji se vyvíjejících oborů informatiky. Její úspěchy na mnoha polích ukázaly, že se jedná o jeden z nejužitečnějších oborů. Jelikož mě tento obor vždy fascinoval, rozhodl jsem se, že vytvořím vlastní implementaci umělé inteligence, konkrétně spojení genetického programování s neuronovými sítěmi. Celou tuto práci jsem založil na práci
o NEATu [1], HyperNEATu [2] a ES-HyperNEATu [3]. Práce samotná obsahuje moji vlastní implementaci zmíněných technik a stručné teoretické pojednání o nich samotných.

1 Základní koncepty

1.1 Neuron

Neuron v původním slova smyslu je základní jednotka nervové soustavy zodpovědná za vedení, přijímání a zpracování signálů. V matematice používáme toto označení pro matematický koncept jednotky, která zpracovává vstupy do určitých výstupů. Je definován množinou vstupů s takzvanými váhami jednotlivých připojení. Neuron potom určuje hodnotu svého vstupu sumou jednotlivých vstupních signálů vynásobené váhou konkrétního spojení a jeho výsledek je poté určen matematickou funkcí, typicky sigmoidou. Každý neuron má speciální vstup bias, který ovlivňuje výstup formou vstupu s konstantní hodnotou (Obrázek 1).



x_n – n-tý vstup

w_n – n-tá váha

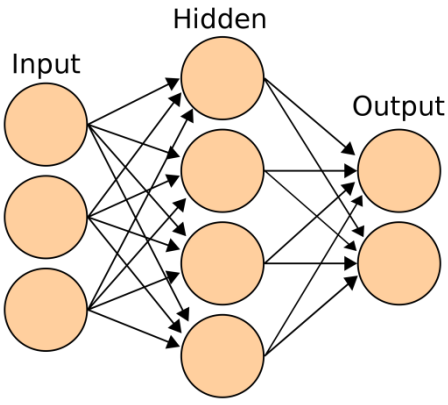
∑ - vnitřní suma vstupů

sign – výstupová funkce v tomto případě signum

Obrázek 1 model neuronu [6]

1.2 Neurální síť

Neurální síť je označení uskupení více neuronů s tím, že výstup minimálně jednoho neuronu je i vstupem jiného neuronu. Zvenčí se neurální síť většinou popisuje jako black box, což znamená, že z množiny vstupů získáme množinu výstupů s tím, že black box má určitý vnitřní stav, který ovlivňuje výstupy. Typická síť se skládá ze tří vrstev, vstupní, výstupní a skryté. Tyto sítě, jakožto i neurony, mezi sebou mohou být libovolně propojeny, aby vytvořili síť vhodnou pro řešení zadaného problému. Typicky se tato síť označuje jako ANN (Artificial Neural Network), jejíž ukázka je na Obrázku 2.



Obrázek 2 model neuronové sítě [7]

1.2.1 CPPN

CPPN je zvláštní typ neurální sítě, který oproti tradiční ANN může mít v každém neuronu jinou vyhodnocovací funkci. Používá se zejména u analyzování či vytváření

geometrických vzorů. Odtud vychází jejich jméno CPPN (Compositional pattern-producing networks – Kompoziční sítě produkující vzor).

1.2.2 Cyklické a acyklické sítě

Je možné sestrojit takovou síť, že výstup neuronu je napojen (nepřímo) na svůj vstup, čímž vzniká cyklická závislost – cyklická síť - která je matematicky neřešitelná (vznikla by nekonečná rekurze). Proto většina sítí taková spojení nedovoluje, a pokud ano, existuje omezený počet znovu aktivací. Síť, která tuto cyklickou závislost neobsahuje, se nazývá acyklická.

1.3 Genetické programování

Genetické programování je technika v programování, která se snaží vytvořit místo programu, který by problém vyřešil, program, který sestaví postup, který problém vyřeší. Základem této techniky je princip evoluce, ve kterém pouze „nejsilnější“ jedinci mohou přežít. Tento přístup se velmi často využívá k vývoji neuronové sítě.

1.4 Strojové učení

Strojové učení je podoblastí umělé inteligence, zabývající se algoritmy a technikami, které umožňují počítačovému systému 'učit se'. Učením v daném kontextu rozumíme takovou změnu vnitřního stavu systému, která zefektivní schopnost přizpůsobení se změnám okolního prostředí. [4]

1.4.1 Učení s učitelem

Jedná se o typ učení, ve kterém jsou poskytnuty programu vstupy a výstupy. Úlohou programu je zjistit spojitost mezi nimi.

1.4.2 Učení bez učitele

Tento typ učení neposkytuje programu žádné hodnocení ani jakoukoliv informaci o výstupu či vstupu, úkolem programu v tomto případě je nalézt spojitost mezi vstupy.

1.4.3 Zpětnovazebné učení

V tomto případě je programu poskytnuta informace jakého výkonu dosáhl, ale nikoliv informace o správném řešení. Typickým příkladem je například úkol ovládání robota v prostředí, ve kterém má sbírat „jídlo“. Robotovi pouze poskytneme informaci o tom, kolik jídla sebral, ale nikoliv však informace o správném postupu či jak daleko od něj byl daleko.

1.5 GPGPU a OpenCL

V současné době je jedním z největších problémů strojového učení vysoká výpočetní náročnost. Pro běžného člověka s běžným hardwarem je proto velmi problematické vycvičit neuronové sítě, které by řešili problém, který uživatel potřebuje. Jedním z možných řešení je samozřejmě optimalizace algoritmu učení, ale tento přístup má jisté hranice, za které se už dostat nemůže. Dalším dostupnějším řešením je využití grafického jádra. Grafické jádro počítače (GPU) je ve srovnání s procesorem stejné cenové kategorie řádově rychlejší, není vzácné, že GPU je i 10x rychlejší, než CPU. Navíc CPU je aktuálně jedinou hardwarovou komponentou, která zvyšuje svou výpočetní sílu rychleji, než to předpovídá Moorův zákon tedy 2x za 18 měsíců. Bohužel v dnešní době není možné využívat GPU stejně, jako CPU. Důvod je takový, že zaprvé GPU získává svoji výpočetní sílu hlavně vysokým počtem jader, tudíž je nutno pracovat s paralelními programy s velkým počtem vláken, za druhé je nutné používat jazyky, které jsou uzpůsobeny GPU. Takové jazyky jsou v dnešní době dva CUDA (NVidia) a OpenCL (AMD). V mém projektu jsem se rozhodl použít OpenCL jelikož je možné používat s jakoukoliv hardwarovou konfigurací.

1.6 Novelty learning

Novelty learning je koncept, který se začal objevovat již od roku 2010 a od té doby jeho popularita roste. Jedná se o koncept, ve kterém se místo konkrétního cíle hledá diverzita. Tento přístup je navržen k překonání problémů se zastavením na lokálním optimu s čímž má většina algoritmů problémy. Nejjednodušší příklad je hledání cesty v bludišti je možné dostat se velmi blízko cíle, a přesto jít naprosto špatnou cestou proto v novelty learningu nehledáme cíl, ale hledáme co nejvíce různých chování, což se může na první pohled zdát zavádějící či až kontraproduktivní ale s rostoucí komplexitou genomů je čím dál těžší dosáhnout rozdílného chování a tímto způsobem můžeme toto hledání podpořit. Díky tomuto přístupu navíc vyvíjíme více univerzální agenty se schopnosti průzkumu, a ne jenom agenty jenž jsou schopni projít například jedno bludiště, které jim zadáme.

1.6.1 Spojení novelty learningu a objective-based learningu

Ve většině případů implementace novelty learningu se v podstatě nahradí objective metrika za novelty většinou za použití archivu předchozích výsledků. Tento přístup má v mnoha případech výhody, a to zejména u problému u nichž je téměř nemožné vyvíjet se naprosto chybným směrem (viz bludiště), ale u více otevřených problémů se často stává, že algoritmus není schopen problém vyřešit. Z tohoto důvodu tato implementace používá algoritmus, který kombinuje oba přístupy. Využívá obě metriky současně s tím, že jednu používá v rámci celé rasy a druhou pouze pro jednotlivce. Tímto přístupem by se měl algoritmus být schopen rozvíjet správným směrem, a tak eliminovat problémy obout přístupů.

2 Techniky učení

2.1 NEAT

NEAT je technika pro vývoj neurálních sítí na které je celá tato práce založená. Její hlavní myšlenkou je snaha začít s co nejjednodušší neurální sítí, která se stává časem složitější. Obsahuje několik konceptů, díky kterým se odlišuje od podobných technik.

2.2 Reprezentace neurální sítě

V NEATu je neurální síť reprezentována několika stavebními jednotkami, které definují několik částí sítě.

2.2.1 Neuron gene (Neuron)

Jedná se o datovou reprezentaci neuronu, která obsahuje informaci o propojených neuronech (vstupy a výstupy) a jeho inovační číslo.

2.2.2 Connection gene (Spojový gen)

Reprezentuje propojení mezi neurony, obsahuje jeho váhu, informaci, jaké neurony propojuje a inovační číslo.

2.2.3 Genom

Obsahuje seznam všech genů, které síť zahrnuje. Jedná se o reprezentaci sítě jako celku pro potřeby vývoje.

2.3 Historické mapování

Je technika představená v NEATu, která nám dovoluje sledovat vývoj jednotlivých genomů. Při provedení jakékoliv akce, která zvyšuje složitost sítě (přidání propojení, přidání neuronu), je novému genu přiřazeno unikátní číslo (inovační číslo), které je poté použito pro vyhledávání shody v síti. Pokud mají dva geny (bez ohledu jestli spojové nebo neurony) stejné inovační číslo, znamená to, že jsou součástí stejné struktury (mohou se lišit vnitřní údaje o váze propojení). Tímto způsobem je možno nalézt jednoduše genomy se stejnými předky a dokonce vyhodnotit, jak moc se od sebe liší. Existuje možnost, že najednou ve dvou rozdílných genomech nastane stejná mutace a přesto jim bude přiřazeno odlišné inovační číslo. Tomuto se v mojí práci bráním tak, že udržuji seznam všech mutací, které se v dané generaci odehrály a pokud zachytím dvě stejné, jednoduše jim přiřadím stejné číslo.

2.3.1 Hledání odpovídajících struktur v genomech

Pokud chceme v NEATu zkombinovat či zjistit podobnost dvou genomů, můžeme velmi jednoduše najít odpovídající struktury díky technice historického mapování. Při srovnávání genomů jednoduše najdeme geny, jejichž inovační čísla si odpovídají. Tyto geny označíme jako odpovídající geny. Poté můžeme jednoduše zjistit podobnost dvou genomů bez nutnosti provádění výpočetně náročných analýz struktury.

2.4 Reprodukce/mutace genomů

Produktem každé generace je populace s určitým množstvím genomů, aby se tyto genomy mohly přibližovat cíli vývoje. Je nutné je oproti svým předkům nějak změnit. Toho může být dosaženo mnoho způsoby, ze kterých se podle určitých pravděpodobností náhodně vybere.

2.4.1 Asexuální reprodukce/mutace

2.4.1.1 Změna vah propojení

Tato nejjednodušší a zároveň nejběžnější mutace upraví váhy náhodně zvolených spojení na náhodné hodnoty.

2.4.1.2 Přidání neuronu

Tato mutace vybere náhodné propojení, které odebere a na jeho místo vloží neuron. Ten je propojen s neurony, které spojovalo původní propojení. Aby dopad způsobený touto mutací nebyl tak drastický, je váha původního spojení použita ke spojení nového neuronu s původním zdrojovým. Váha spojení nového neuronu a původního cílového neuronu je nastavena na maximální hodnotu.

2.4.1.3 Přidání propojení

Při této mutaci vybereme náhodně dva neurony, které propojíme. Důležité u této mutace, pokud nechceme vytvořit cyklickou síť, je prověřit, jestli se tímto spojením síť nezacyklí.

2.4.1.4 Odebrání propojení

Tato mutace vybere náhodné propojení, které potom odstraní. Po odstranění zkontroluje, jestli neuron, který k ní byl připojen, má alespoň jeden vstup a výstup (pokud jde o neuron skryté vrstvy). Pokud tato spojení nemá, je z genomu také odebrán.

2.4.2 Sexuální reprodukce

Při sexuální reprodukci se zvolí dva náhodné genomy jedné rasy, u kterých potom vyhledáme všechny geny se stejným inovačním číslem. Zbytek genů zahrneme podle náhodné šance. Váhy všech spojových genů zprůměrujeme od obou rodičů (u odpovídajících genů). Občas můžeme vybrat
i genomy z různých ras, ale šance, že se to stane, je mnohonásobně nižší.

2.5 Minimalizace složitosti genomu

U většiny podobných systémů, pracujících s neurálními sítěmi, pracujeme s již komplexním návrhem sítě od první generace. U NEATu začínáme s co nejjednodušší sítí. Nejčastěji bez žádného neuronu ve skryté vrstvě. Pouze s několika propojeními vstupních a výstupních neuronů. Díky tomu je NEAT schopen hledat řešení, která jsou blíže optimálnímu.

2.6 Dělení genomů do ras

Vzhledem k možnosti jednoduše zjistit podobnost genomů a díky historickému mapování obsahuje NEAT koncept ras. Rasa je skupina genomů, které jsou si nejvíce podobné s tím, že velikost skupin se může měnit, ale jejich počet je předem stanoven. Důvodem jejich přítomnosti je to, že při reprodukci může vzniknout struktura, která může být užitečná pro vyřešení problému. Zpočátku ale může být zbytečná a může dokonce zhoršovat výkon, proto NEAT zahrnuje sdílení úspěchu řešení problému v celé rase, čímž se ochrání tyto zatím neoptimalizované struktury.

2.7 Výhody přístupu

Hlavní výhodou samotného NEATu je jeho relativně nízká výpočetní náročnost a je ideální pro takové experimenty, které pracují s množstvím dat, které se nedají logicky uspořádat v prostoru (neexistuje mezi nimi žádný geometrický vztah).

2.2 HyperNEAT

2.2.1 Obecná charakteristika

HyperNEAT je metoda, která k vytváření neuronových sítí přistupuje jinak. Nebere vstupy pouze jako množinu měnících se hodnot, ale vyžaduje i stanovení jejich pozice v n rozměrném prostoru, od čehož pochází název HyperNEAT – Hypercube NEAT. Hypercube je matematický koncept, který definuje n rozměrný pravidelný objekt (n = 0 => bod, n = 1 => úsečka, n = 2 => čtverec...). V HyperNEATu je pozice stanovená v intervalu <-1;1> na každé z prostorových os. Díky tomu, že využívá CPPN, může tuto informaci využít k vyhledávání geometrického vztahu mezi vstupy a výstupy. V této metodě, mírně upravená verze NEATu, vytvoří CPPN, která je potom aplikována na všechny potencionální propojení. Pokud je hodnota výstupu z CPPN vyšší než předdefinovaný práh, je spojení vyjádřeno a zahrnuto do výsledného genomu. Tento způsob dekódování sítě se nazývá nepřímý, jelikož výsledná síť může být jinak složitá, než původní síť, ze které vychází.

2.2.2 Získání váhy propojení

Pro získání váhy propojení se používá vygenerovaná CPPN. Počet vstupů CPPN je určen jako x=2*n, kde x je počet vstupů a n je počet rozměrů hypercube, ve které problém řešíme. Váhu z této sítě získáme tak, že pozici genů, mezi kterými chceme zjistit váhu spojení, zadáme jako vstupy do sítě, s tím, že výstup nám určuje váhu potencionálního propojení.

2.2.3 Skryté neurony

Jedním z hlavních nedostatků HyperNEATu je, že se může dostat do situace, ve které nepůjde efektivně zmapovat vstupy k výstupům bez neuronů ve skryté vrstvě. Jelikož ale HyperNEAT nespecifikuje metodu, jak zjistit umístění takových neuronů, je jediná možnost, aby uživatel předdefinoval pozici skrytých neuronů spolu se vstupy a výstupy.

2.2.4 Možnost rozšíření

Další výhodou, oproti NEATu, je možnost měnit počet a pozici vstupů v průběhu simulace. Jelikož vygenerovaná síť CPPN definuje jenom geometrický vztah dvou neuronů v hypercube, není problém přidávat další. Pokud dodržíme stejná pravidla při umisťování, měla by se funkčnost téměř nezměnit a v některých případech mírně snížit.

2.3 ES-HyperNEAT

2.3.1 Obecná charakteristika

ES-HyperNEAT je rozšíření k technice HyperNEAT, které se snaží řešit problém s umisťováním skrytých neuronů za použití stejných informací jako tradiční HyperNEAT. Používá tedy také CPPN a jedná se též o nepřímé dekódování.

2.3.2 Zjištění pozice skrytých neuronů

ES-HyperNEAT nahlíží na problematiku opět trochu jinak. Snaží se nalézt body v hypercube (v mé implementaci pouze 2 rozměrné), do kterých by mělo smysl vytvořit spojení. Pokud se algoritmus rozhodne takové spojení vytvořit, musí vytvořit i neuron na pozici, se kterou se spojuje. Spojení má cenu vytvořit pouze za předpokladu, že je v oblasti dostatek informací. Pro to, aby spojení bylo označeno za podstatné, musí být váha potencionálních propojení v okolí dostatečná. Problémem u této metody je, že množství propojení je teoreticky nekonečné. V praxi bude ovlivněné pouze přesností použitého datového typu. Další věcí, kterou si je nutné uvědomit je, že u každého problému existuje krajní hustota propojení, za kterou nemá smysl zvyšovat počet propojení. Jelikož bychom tím

nezískali žádné nové informace. Je tedy nutné vybrat algoritmus, který by byl schopen rozhodnout o různé hustotě spojení podle míry přítomnosti informací (heterogeneity). Takovým algoritmem je například quadtree.

2.3.2.1 Quadtree

Quadtree je algoritmus navržený pro dělení dvou rozměrné plochy (většinou čtverce) na čtyři stejně velké části, které se dále mohou stejným způsobem dělit do libovolné hloubky.

2.3.2.2 Dělení podle heterogeneity

Dělení quadtree je prováděno, dokud míra informací definována vztahem $v = \frac{1}{k}\sum_{1}^{k}(w - w_i)^2$, kde k je počet dílů v quadtree, w je průměrná váha spojení všech dílů a w_i je váha spojení s dílem quadtree. Pokud tato hodnota překročí předdefinovaný práh je quadtree dále dělen.

3 Grafické rozhraní a implementace

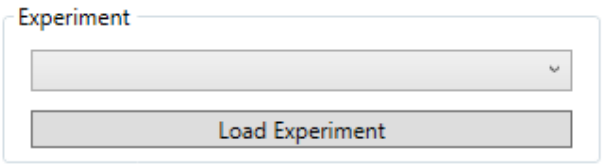
V této kapitole je ukázáno grafické rozhraní aplikace, realizace experimentů, které byly vysvětleny v předcházející kapitole. Na závěr kapitoly je ukázka implementace vybraného skriptu aplikace.

3.1 Popis GUI

GUI je složeno z několika částí, z nichž většina má čistě informační funkci.

3.1.1 Experiment

V této části je možné pouze vybrat a nahrát experiment (Obrázek 3).

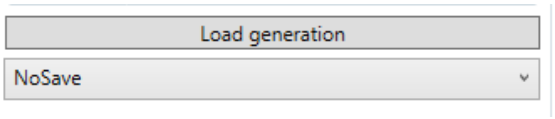


Obrázek 3 GUI experiment

3.1.2 Ukládání a nahrávání

V této části můžeme nahrát generaci genomů a zvolit metodu ukládání generací (Obrázek 4). Máme na výběr několik možností:

- No save – generace se neukládájí
- Save Next – uloží se pouze příští generace, nastavení se poté vrátí do stavu No Save
- Save Last – ukládá pouze nejnovější generaci, předchozí generace maže
- Save All – ukládá každou generaci



Obrázek 4 GUI load a save

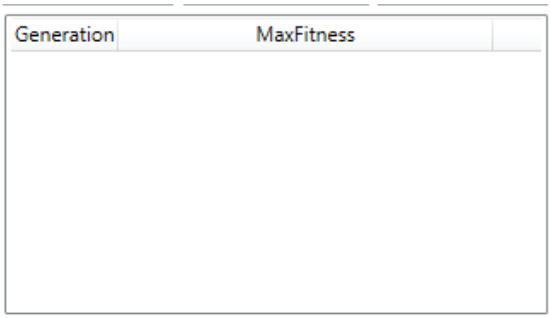
3.1.3 Řízení programu

Tato část umožňuje řízení celého procesu učení. Tlačítko start spustí evoluci nového experiment, nebo pokud byl program pozastaven ho znovu spustí. Pause pozastaví program (pozastaví ho až po skončení aktuální generace). Reset pozastaví experiment a připraví program pro další experiment.

Obrázek 5 GUI population

3.1.4 Tabulka generací

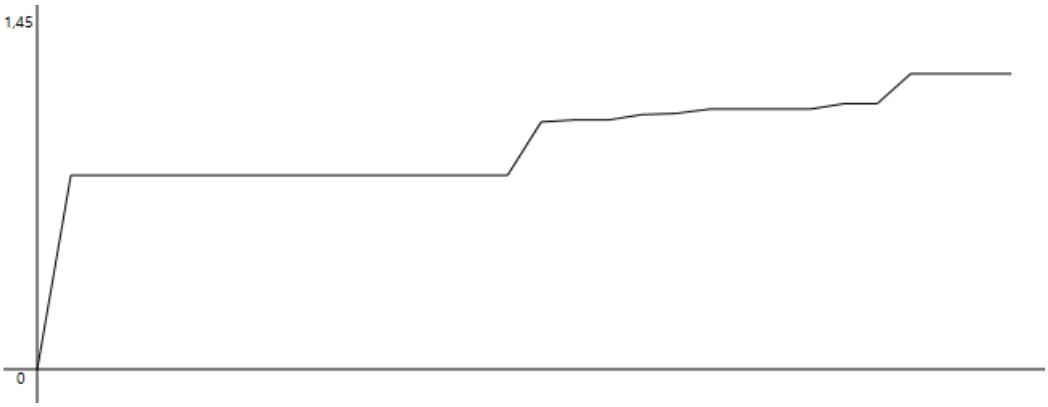
Zde se vypisuje seznam generací historicky seřazeny, vypisuje se tu pouze číslo generace a nejvyšší dosažená úspěšnost.



Obrázek 6 GUI tabulka generací

3.1.5 Graf úspěšnosti

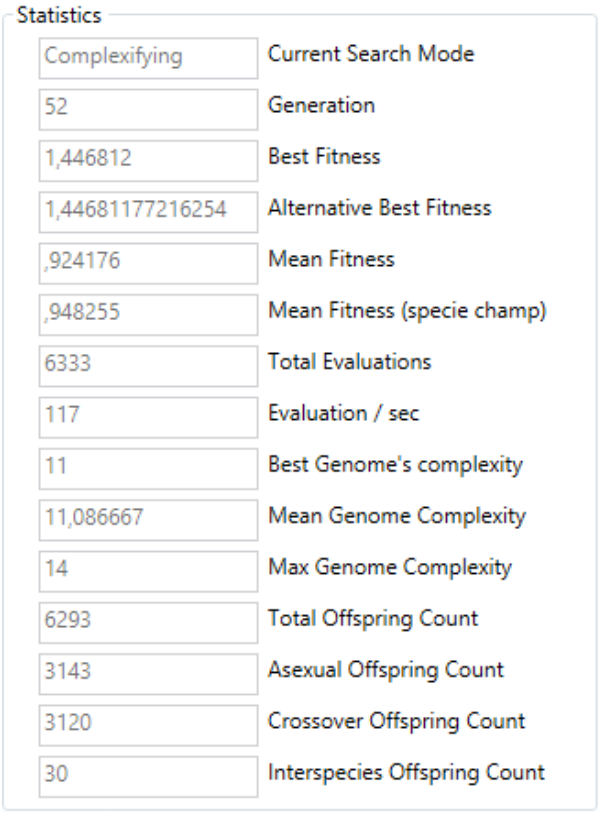
Pod tabulkou se vykresluje jednoduchý graf vývoje úspěšnosti nejlepšího genomu. Na ose X je vyznačena generace a na ose Y úspěšnost (Obrázek 7).



Obrázek 7 GUI graf úspěšnosti

3.1.6 Statistiky vývoje

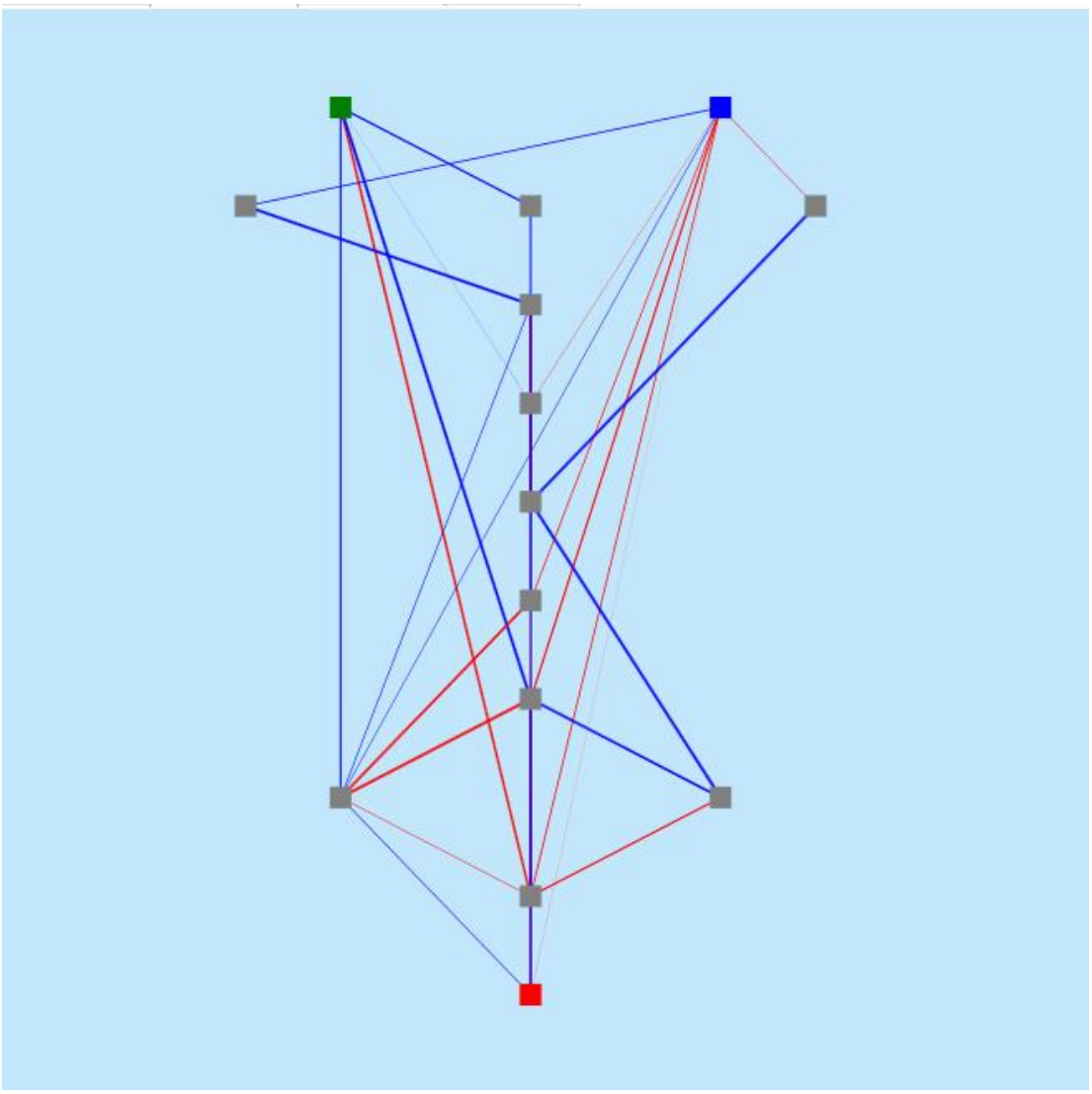
Tato část, zobrazená na Obrázku 8, zobrazuje všechny důležité statistické údaje o vývoji.



Obrázek 8 GUI statistiky vývoje

3.1.7 Genome view

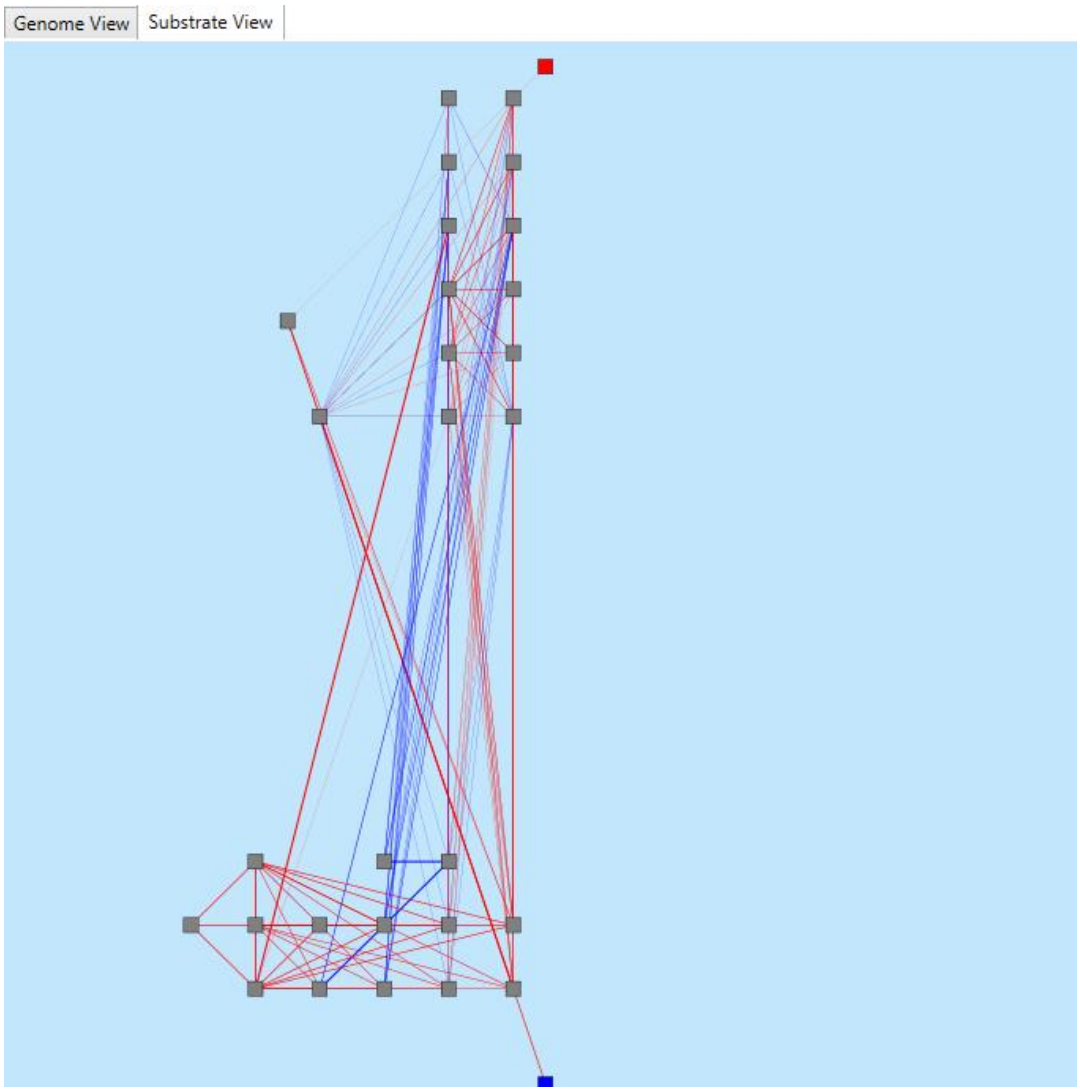
Zobrazuje grafickou reprezentaci neuronu (Obrázek 9). Modré čtverce označují vstupy, červené výstupy, šedé skryté neurony a zelený bias. Propojení, jejichž váha je záporná, jsou vykreslovány červeně. Kladná váha je značena modře.



Obrázek 9 GUI genome view

3.1.8 Substrate view

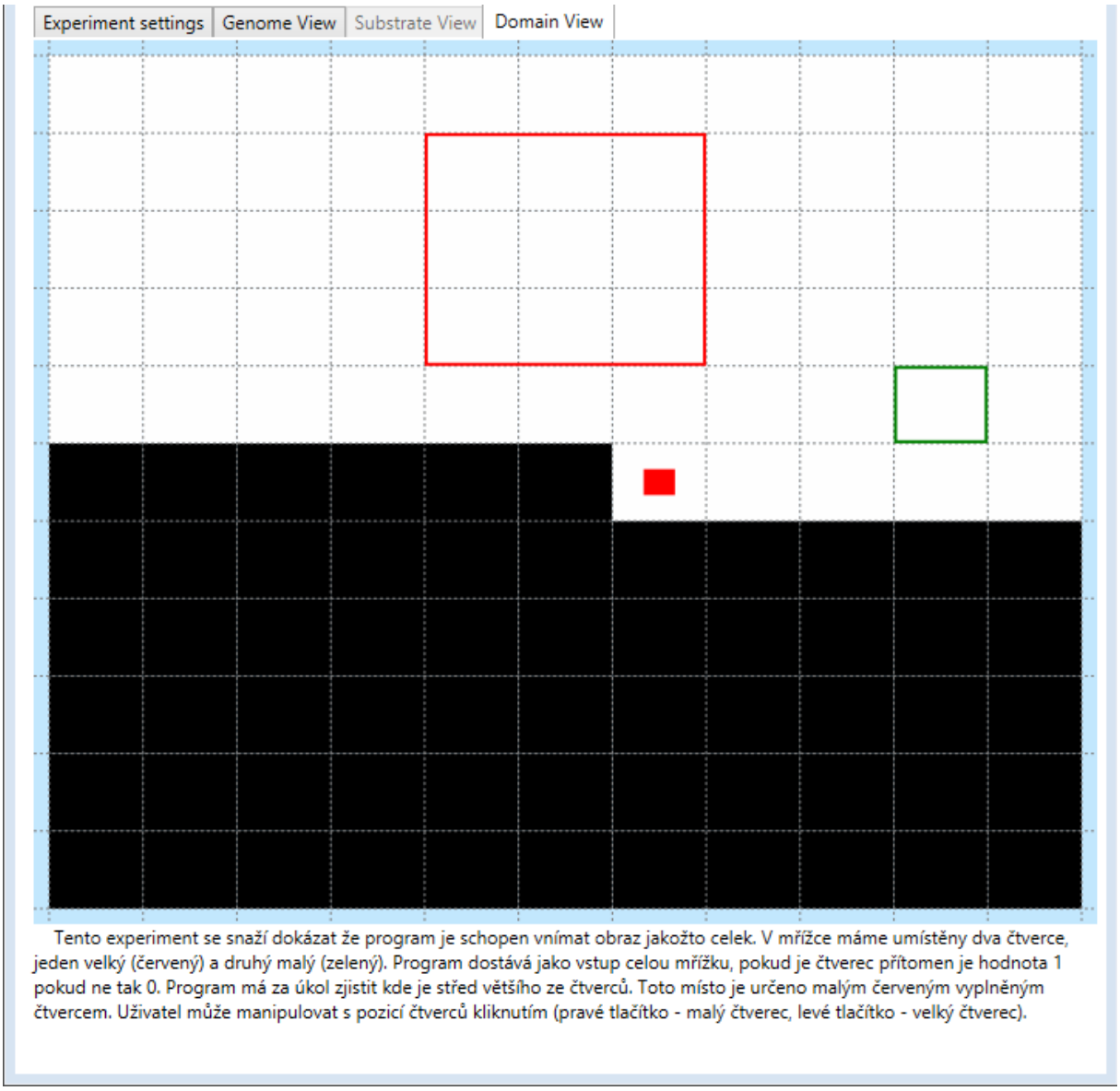
Zobrazuje grafickou reprezentaci substrátu (hypercube s neurony). Funguje pouze při použití techniky ES-HyperNEAT (Obrázek 10). Označení jsou stejná jako u genome view. Jediný rozdíl je, že z důvodu přehlednosti se nezobrazují spojení s biasem.



Obrázek 10 GUI substrate view

3.1.9 Domain view

Tato část, která je na Obrázku 11, zobrazuje grafickou reprezentaci právě probíhajícího experimentu. V některých případech je tato část interaktivní. V dolní části jsou uvedeny detaily experimentu a tpůsob, jak je uživatel může ovlivnit.



Obrázek 11GUI Domain view

3.2 Implementace

Program je napsaný v jazyce C# a používá kromě standartního frameworku .NET pouze dvě mnou nenapsané třídy, FastRandom a ZigguratGaussianSampler [5], které jsem pouze upravil pro svoji potřebu. Je rozdělen do tří projektů, první obsahuje celý evoluční algoritmu – NeatLib, druhý definuje experimenty – NeatExperiments a poslední obsahuje GUI – NeatGUI. Experimentálně je také do programu přidána implementace neuronových sítí pomocí OpenCL a tedy vyžití GPGPU (General Purpose programming on Graphical Processing Unit). Tato implementace se v současné době stále testuje a je možná použít pouze u acyklických sítí.

Závěr

V mé práci jsem vytvořil vlastní implementaci algoritmů pro vytváření neuronových sítí pomocí technik NEAT, HyperNEAT a ES-HyperNEAT. Při mé práci jsem navrhoval systém tak, aby bylo jednoduché ho znovu použít, či upravovat pro další rozšiřování. V budoucnu bych rád svou práci dále rozvíjel, hlavním směrem, kterým bych se rád vydal, je přenesení většiny výpočtů na grafickou kartu a mnoho jiných způsobů optimalizace, například technika pro optimalizaci ES-HyperNEATu [8].

Seznam použité literatury

[1] Evolving Neural Networks through Augmenting Topologies [online]. Massachusetts Institute of Technology, 2002 [cit. 2016-03-03]. Dostupné z: http://nn.cs.utexas.edu/downloads/papers/stanley.ec02.pdf

[2] *A Hypercube-Based Indirect Encoding for Evolving Large-Scale Neural Networks* [online]. School of Electrical Engineering and Computer Science University of Central Florida 4000 Central Florida Blvd. Orlando, FL 32816-2362 USA, 2009 [cit. 2016-03-03]. Dostupné z: http://eplex.cs.ucf.edu/papers/stanley_alife09.pdf

[3] *An Enhanced Hypercube-Based Encoding for Evolving the Placement, Density and Connectivity of Neurons* [online]. University of Central Florida Orlando, FL 32816-2362 USA, 2012 [cit. 2016-03-04]. Dostupné z:

http://eplex.cs.ucf.edu/papers/risi_alife12.pdf

[4] Strojové učení. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2016-03-02]. Dostupné z: https://cs.wikipedia.org/wiki/Strojov%C3%A9_u%C4%8Den%C3%AD

[5] Heliosphan [online]. [cit. 2016-03-02]. Dostupné z:

http://heliosphan.org/index.html

[6] Neurons, as an Extension of the Perceptron Model. *Math ∩ Programming* [online]. 2012 [cit. 2016-03-04]. Dostupné z:

http://jeremykun.com/2012/12/09/neural-networks-and-backpropagation/

[7] Artificial neural network. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2016-03-04]. Dostupné z: https://en.wikipedia.org/wiki/Artificial_neural_network

[8] E*nhancing ES-HyperNEAT to Evolve More Complex Regular Neural Networks* [online]. Orlando, FL 32816, USA, 2011 [cit. 2016-03-02]. Dostupné z:

http://eplex.cs.ucf.edu/papers/risi_gecco11.pdf

Seznam obrázků