Mendelova univerzita v Brně
Provozně ekonomická fakulta

Řízení autonomního agenta pomocí neuroevoluce

Diplomová práce

Vedoucí práce:
Ing. Jiří Lýsek, Ph.D.

Bc. Martin Hnátek

Brno, 2018

Rád bych zde poděkoval svému vedoucímu Ing. Jiří Lýskovi, Ph.D. za jeho cenné rady a čas, který mi věnoval při řešení této práce.

Čestné prohlášení

Prohlašuji, že jsem práci: **Řízení autonomního agenta pomocí neuroevoluce** vypracoval samostatně a veškeré použité prameny a informace uvádím v seznamu použité literatury. Souhlasím, aby moje práce byla zveřejněna v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a v souladu s platnou *Směrnicí o zveřejňování vysokoškolských závěrečných prací.*

Jsem si vědom, že se na moji práci vztahuje zákon č. 121/2000 Sb., autorský zákon, a že Mendelova univerzita v Brně má právo na uzavření licenční smlouvy a užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

Dále se zavazuji, že před sepsáním licenční smlouvy o využití díla jinou osobou (subjektem) si vyžádám písemné stanovisko univerzity, že předmětná licenční smlouva není v rozporu s oprávněnými zájmy univerzity, a zavazuji se uhradit případný příspěvek na úhradu nákladů spojených se vznikem díla, a to až do jejich skutečné výše.

V Brně dne 11. prosince 2018 .....

Abstract

Autonomous agent control using neuroevolution

Abstrakt

Řízení autonomního agenta pomocí neuroevoluce

Tato práce se zabývá trénováním autonomního agenta - auta s pomocí algoritmu neuroevoluce. Toto zahrnuje tvorbu simulačního prostředí pro agenta, vhodným návrhem agenta (senzorů a řízení) a také návrhem fitness funkce

Obsah

1 Úvod a cíl práce **9**
1.1 Úvod do problematiky . . . 9
1.2 Cíl práce . . . 9

2 Neuronové sítě **10**
2.1 Neuron . . . 10
2.2 Vrstva . . . 10
Aktivační funkce . . . 10
Linearní funkce . . . 10
Sigmoid . . . 11
Tanh . . . 11
RELU . . . 12
Rekurentní neurony . . . 12

3 Genetické algoritmy **14**
3.1 Princip . . . 14
3.2 Kódování . . . 14
3.3 Křížení . . . 15
3.4 Mutace . . . 15
3.5 Selekce . . . 15
3.6 Využití . . . 15

4 NEAT **16**
4.1 Genotyp a fenotyp . . . 16
4.2 Mutace . . . 16
4.3 Křížení . . . 17

5 Použité technologie **19**
5.1 Docker . . . 19
5.2 Vue . . . 19
5.3 Node.js . . . 19
5.4 Bull . . . 19
5.5 Postgre . . . 19
5.6 PIXI.js . . . 19
5.7 CES . . . 20
Komponenta . . . 20
Entita . . . 20
System . . . 20

6 Vlastní práce **21**

7 Simulace **22**
7.1 Části ECS použité v simulaci 22
Entity 23
Komponenty 23
Systémy 23
7.2 Fitness funkce 23
Agent 24

8 Serverová část **26**
8.1 Výpočetní cluster 26
Docker swarm 26
8.2 Průběh vyhodnocování 27

9 Klientská část **28**

10 Experimenty **29**
10.1 Nekonečná silnice ve tvaru I 29

11 Možná vylepšení **30**

12 Závěr **31**

13 Reference **32**

Přílohy **33**

A CD se zdrojovým kódem **34**

Seznam obrázků

1 Příklad křížení 15
2 Příklad mutace 15
3 Genotyp a fenotyp u algoritmu NEAT převzato z (STANLEY, Kenneth O, Risto MIIKKULAINEN., 2002, s. 9) 16
4 Mutace v NEAT převzato z (STANLEY, Kenneth O, Risto MIIKKULAINEN., 2002, s. 10) 17
5 Křížení v NEAT. Převzato z (STANLEY, Kenneth O, Risto MIIKKULAINEN., 2002, s. 12) 18
6 Neuronová síť agenta 25
7 Schéma distribuovaných výpočtů 27
8 Fitness agenta v průběhu času 29

Seznam tabulek

1 Použitý hardware 26

1 Úvod a cíl práce

1.1 Úvod do problematiky

S růstem výpočetního výkonu a rozvojem **gpugpu** (paralelizace výpočtů na grafické kartě) se neuronové sítě ukázaly jako mocný nástroj pro řešení složitých problémů na které standardní metody umělé inteligence nestačily.

Další oblastí umělé inteligence, které nárust masivní paralelizace prospěl, jsou metaheuristiky, které mohou s nárůstem výpočetního výkonu v rozmném čase pokrýt stále větší stavový prostor a jsou tedy schopné rychle řešit stále složitější problémy.

Neuroevoluce propojuje oba přístupy a využívá je ke generování topologii neuronových sítí a nastavení jejích vah. Výsledkem je neuronová síť, jejiž architektura lépe popisuje daný problém a lze jí aplikovat i na problémy na které by klasické neuronové sítě aplikovat nešly. Jedná se například o problémy, u kterých je těžké získat trénovací data a nelze tedy neuronovou síť natrénovat klasickými metodami jako je backpropagace.

1.2 Cíl práce

Cílem této práce je navrhnout a vytvořit simulační prostředí pro autonomního agenta. Poté na simulovaném prostředí provést serii experimentů, které slouží k zjištění limitů učících schopností algoritmu NEAT.

2 Neuronové sítě

Neuronové sítě jsou model strojového učení, který je volně založený na principu zvířecího mozku (PATTERSON, Josh. 2017, s. 41).

2.1 Neuron

Neuron je základní výpočetní jednotka neuronových sítí, která je definovaná jako suma všech jejích vstupů a aplikace aktivační funkce.

$$\sigma(\Sigma_{i=0}^{N}\theta_i \cdot x_i + b)$$

Kde:

1. σ - aktivační funkce
2. θ - skrytá váha pro daný vstup
3. b - bias neuronu

2.2 Vrstva

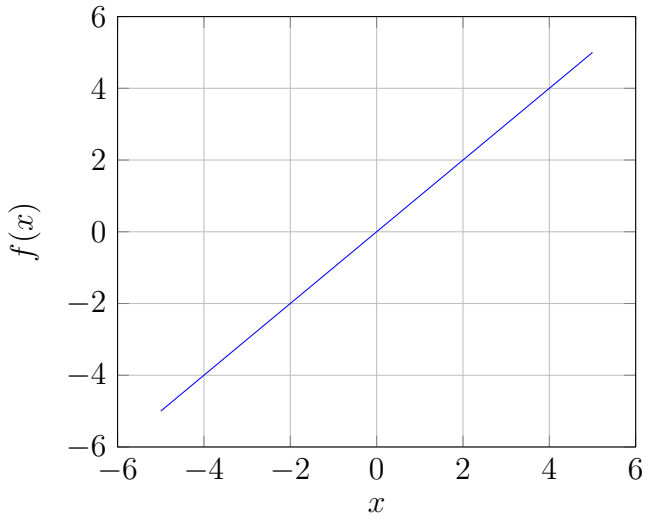
Vrstva je skupina neuronů se stejnou aktivační funkcí.

Aktivační funkce

Aktivační funkce se používá pro definování výstupu a zavedení nelinearity. Bez nich by byla neuronová síť schopna aproximovat pouze n-dimenzionální rovinu. (PATTERSON, Josh. 2017, s. 65)
Dalším využitím je omezení výstupních hodnot. Například aktivační funkce sigmoid se s oblibou používá u výstupní vrstvy neuronových sítí určených ke klasifikačním problémům, protože je to relace $\mathbb{R} \rightarrow \{0..1\}$, která se dá jednoduše jako "jistota"neuronu, že se jedná o výstup, který neuron reprezentuje. Podobně se dá uvažovat i o funkcích jako je například softmax a tanh, které také najdou hojné využití u klasifikačních problémů.

Linearní funkce

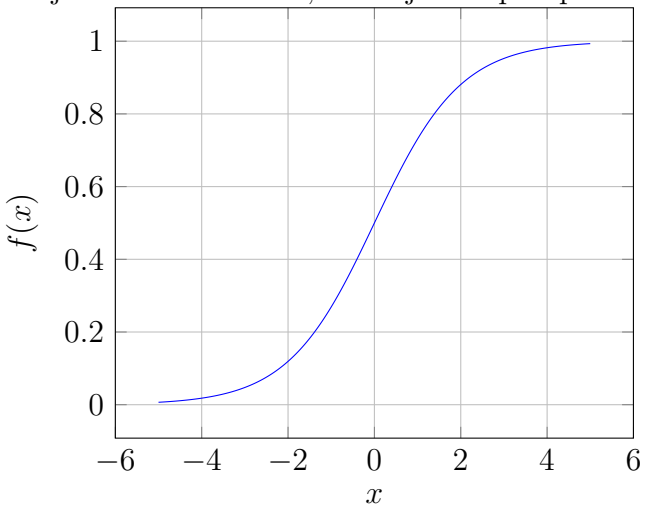
Vrací vstup, tak jak je. Využití najde především u vstupní vrstvy neuronové sítě a u neuronových sítí, které řeší regresní typy úloh.



$$f(x) = x$$

Sigmoid

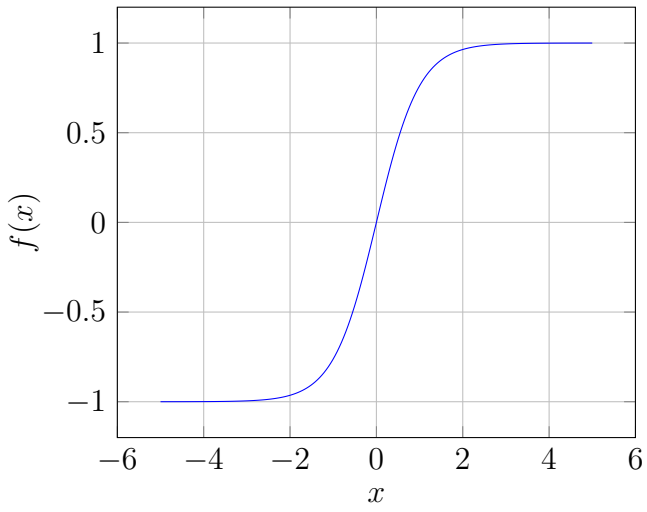
Sigmoid je aktivační funkce, která je schopná potlačit extrémní hodnoty



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

Tanh

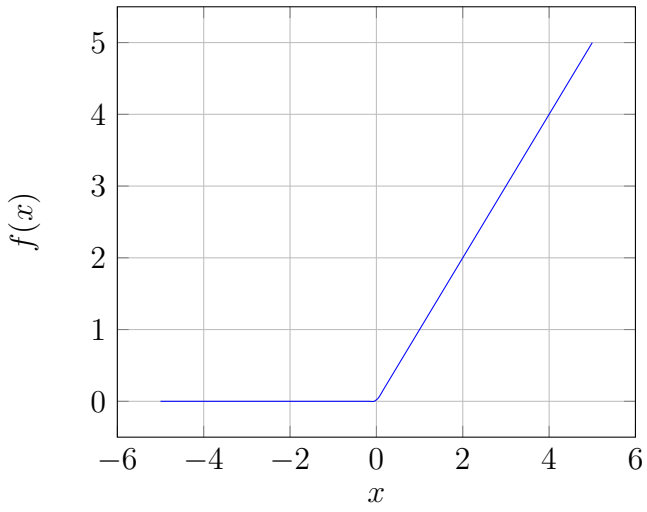
Tanh je funkce obdobná sigmoidu. Hlavní rozdíl mezi ní a sigmoidem je ten, že její obor je v rozmezí -1 a 1 hodí se proto i pro záporná výstupy, které vyžadují záporná čísla. (PATTERSON, Josh. 2017, s. 67)



$$f(x) = tanh(x)$$

RELU

RELU je aktivační funkce, která je podobná lineární aktivační funkci s tím rozdílem, že pokud vstupní hodnota nepřesáhne určitého prahu výstupem je 0. Její hlavní výhodou je to, že zabraňuje problémům s takzvaným explodujícím gradientem (PATTERSON, Josh. 2017, s. 69)



$$f(x) = \begin{cases} x >= 0, & x \\ x < 0, & 0 \end{cases}$$

Rekurentní neurony

Speciální druh neuronů, který si dokáže zapamatovat a reagovat na sekvenci vstupů. Rekurentní neurony značně rozšiřují možnosti neuronových sítí. S nimi je možné řešit

složité úlohy typu strojového překladu nebo sémantické vyhodnocování textu.

V praxi se setkáváme s dvěma druhy neuronů a to LSTM a novější GRU. LSTM a GRU jsou si velmi podobné s jedním podstatným rozdílem. U GRU bylo prokázáno, že je značně rychlejší.

3 Genetické algoritmy

Genetické algoritmy slouží k řízenému prohledávání stavového prostoru založené na teorii evoluce.

3.1 Princip

Základní myšlenka spočívá ve vygenerování náhodných jedinců (řešení problému) a jejích postupné zlepšování s pomocí operací křížení a mutace. Proces zlepšování genomů probíhá na základě jeho hodnocení (fitness).

Samotný algoritmus pak lze rozdělit na následující kroky (MITCHELL, Melanie., 1996, s. 12)

1. Vygeneruj náhodnou populaci o n chromozomech
2. Pro každý chromozom spočítej jeho fitness
3. Opakuj n krát
   a) Vyber pár chromozomů na základě jejích ohodnocení (selekce)
   b) S určitou pravděpodobností p_c rozděl pár chromozomů na náhodném místě a jejích spojením.
   c) S určitou pravděpodobností mutuj daného jedince
4. Nahraď současnou populaci populací, která vznikla předchozím krokem
5. Jdi na krok 2

3.2 Kódování

Někdy se mu též říká genotyp je způsob zápisu řešení problému. Je důležité zároveň uvést pojem fenotyp který označuje vlastní řešení úlohy.

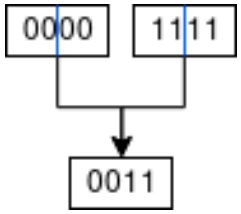
Existuje mnoho různých kódování a každý má své výhody a nevýhody. Většinou se snažíme vybírat kódování, které dobře reprezentuje daný problém. Zde je seznam několika nejpoužívanějších kódování (HYNEK, Josef., 2008, s. 42-43):

1. Binární - řetězec bitů, který může například reprezentovat jednu nebo více numerických hodnot.
2. Reálná čísla - Jedno nebo více reálných čísel
3. Kombinatorické - Může být například seznam čísel označujících

3.3 Křížení

Křížení je operátor, který kombinuje dva jedince do jednoho. Jedinec, který vzniká přebírá genetickou informaci od obou jedinců v určitém poměru. Jeho implementace je závislá na použitém kódování.

Příkladem může být obrázek 1 ve kterém je ilustrováno tzv. jednobodové křížení (HYNEK, Josef., 2008, s. 50) při kterém dochází k rozdělení obou genomů ve stejném bodě a spojením vzniklých podřetězců do nového genomu.



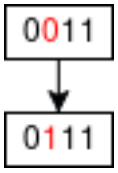
Obrázek 1: Příklad křížení

3.4 Mutace

Mutace náhodně modifikuje chromozom a zavádí tak do populace variaci. Díky tomuto se může algoritmus dostat z lokálního optima.

Existují různé varianty mutace, která se provádí v závislosti na daném kódování.

Například u binárního kódování se může jednat o náhodnou inverzi některého z bitů genomu viz obrázek 2.



Obrázek 2: Příklad mutace

3.5 Selekce

Selekce je proces výběru dvou jedinců na něž jsou později aplikovány genetické operátory jako je křížení a mutace.

3.6 Využití

Genetické algoritmy naleznou využití v mnoha optimalizačních úlohách. Příkladem může být experiment, který zahrnoval použití gramatické evoluce pro vytvoření regresních modelů pro datasety CASY3 a CASY5 (LÝSEK Jiří, ŠŤASTNÝ Jiří, 2014, s. 2-9).

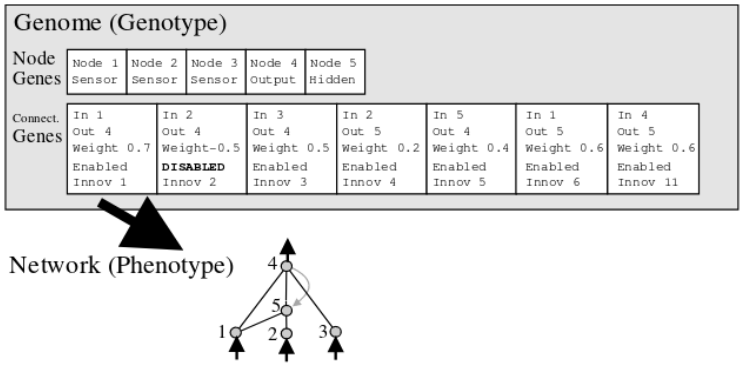
4 NEAT

NEAT (neuroevolution of augmenting topologies) kombinuje neuronové sítě s genetickými algoritmy. Hlavní výhodou této metody je, že generuje jak topologii neuronové sítě, tak její váhy. Výsledkem může být neuronová síť jejíž rozložení lépe popisuje řešený problém.

Algoritmus probíhá stejně jako běžný genetický algoritmus. Rozdílem je použitý fenotyp a operátory mutace a křížení, které se nad ním provádí.

4.1 Genotyp a fenotyp

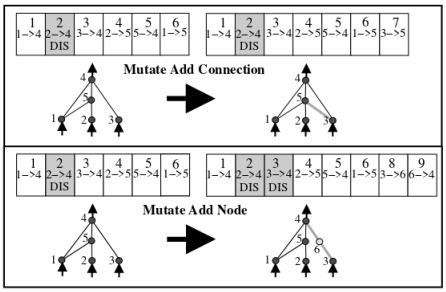
Genotyp a fenotyp je ilustrován na obrázku 3. Genotyp obsahuje jak údaje o jednotlivých neuronech, tak informace o topologii sítě. Metadata, která se týkají topologie neuronové síti jsou údaje o spojení (IN, OUT), váha samotného spojení (weight), informace týkající se toho, zda je spojení použito v fenotypu (ENABLED/DISABLED) a inovační skore. Inovační skore je číslo, které reprezentuje pořadí ve kterém se daný gen objevil v fenotypu (STANLEY, Kenneth O, Risto MIIKKULAINEN., 2002, s. 9).



Obrázek 3: Genotyp a fenotyp u algoritmu NEAT převzato z (STANLEY, Kenneth O, Risto MIIKKULAINEN., 2002, s. 9)

4.2 Mutace

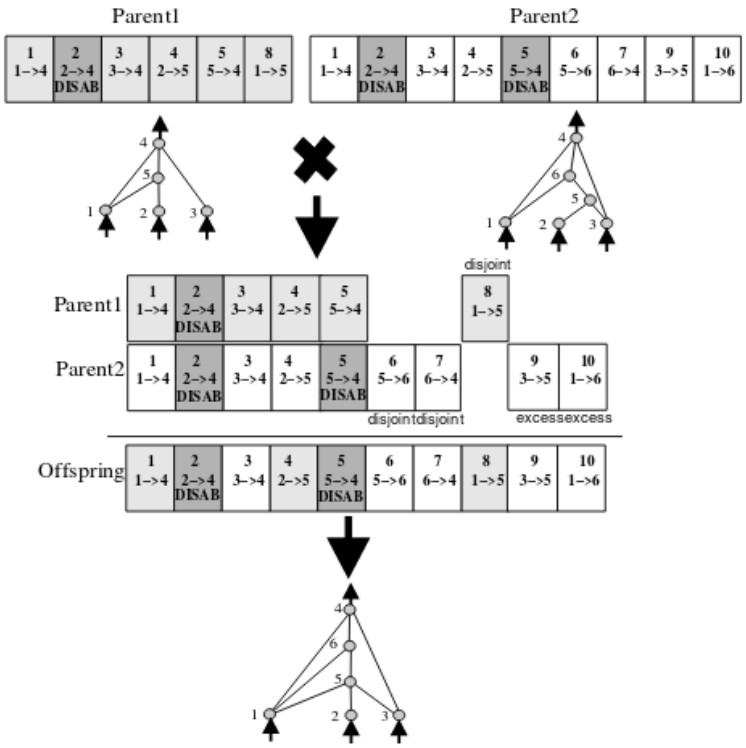
Jak jíž bylo zmíněno nad fenotypem lze provádět různé operace včetně operátoru mutace, který provede náhodnou změnu fenotypu s nadějí, že změna povede k zlepšení řešení. V případě algoritmu NEAT je operátor mutace ilustrován na obrázku 4. Na obrázku je vidět, že mutace může buď přidat další spojení nebo další neuron. V případě, že je přidávan nový neuron je mu přiřazeno náhodné spojení, které je označeno znakem DISABLED (STANLEY, Kenneth O, Risto MIIKKULAINEN., 2002, s. 10).



Obrázek 4: Mutace v NEAT převzato z (STANLEY, Kenneth O, Risto MIIKKULAINEN., 2002, s. 10)

4.3 Křížení

Posledním operátorem používaným při algoritmu NEAT je operátor křížení. Operátor křížení ilustruje obrázek 5. Křížení probíhá na základě inovačního čísla. Spojení se stejným inovačním číslem jsou náhodně zděděny z rodičovských genů. Je tu také šance na převzetí genů, které vytváří spojení nenacházející se v jednom z rodičů. Tyto geny jsou převzaty pouze z rodiče, který ma větší fitness. Při křížení je také náhodná šance, která může převzatý genom vypnout (STANLEY, Kenneth O, Risto MIIKKULAINEN., 2002, s. 12).



Obrázek 5: Křížení v NEAT. Převzato z (STANLEY, Kenneth O, Risto MIIKKULAINEN., 2002, s. 12)

5 Použité technologie

Tato kapitola poskytuje stručný přehled technologii použitých při řešení diplomové práce.

5.1 Docker

Docker je nástroj, který umožňuje vytvářet tzv. kontejnery. Kontejner poskytuje izolované softwarové prostředí ve kterém lze spouštět jednu nebo více aplikací. V praxi to umožňuje snadné nasazení libovolné aplikace bez ohledu na aktuální konfiguraci hostitelského systému.

5.2 Vue

Je populární javascript framework pro snadnou tvorbu uživatelských rozhraní.

5.3 Node.js

Nodejs je open-source interpret jazyku javascript. Využití najde při psaní serverových aplikací v javascriptu a jiné případy, kdy je třeba spustit kód napsaný v javascriptu mimo prohlížeč. Lze ho využít například pro spouštění testů při CI (continuous integration) nebo pro tvorbu vysoce serverových aplikací v javascriptu.

Narozdíl od běžného javascriptu obsahuje rozšířenou standardní knihovnu pro snadnou tvorbu serverových aplikací. Tato standardní knihovna umožňuje například javascript kódu prácí se soubory na hostitelském systému, což je něco, co není z bezpečnostních důvodů v klasickém javascriptu, který běží v prohlížeči možné.

5.4 Bull

Bull je knihovna pro Node.js, která poskytuje frontu úkolů založenou na populární in-memory databázi redis.

5.5 Postgre

Populární databázový systém, který nabízí robustní open source alternativu i ke komerčním řešením.

5.6 PIXI.js

Grafická knihovna pro snadné vykreslování nad html 5 canvasem. Obaluje jak klasické vykreslovací api, tak modernější webgl.

5.7 CES

Je knihovna, která implementuje tzv. ECS (entity component system). ECS se používá především ve hrách a nabízí určitý způsob, jak se dívat na herní objekty a logiku. Myšlenka je taková, že vše lze rozdělit do níže uvedených podsekcí.

Komponenta

Komponenty bývají jednoduché datové struktury, které vyjadřují vlastnosti entity u níž jsou přiřazeny.

Entita

K entitě se dá přiřadit jedna nebo více komponent (vlastností). Entita tak může představovat libovolný herní objekt.

Příkladem může být vozidlo, které může být reprezentováno složením fyzikální, grafické a ovládací komponenty. Při správné implementaci níže zmiňovaných systému lze pak na jakoukoliv entitu, která má tyto komponenty nahlížet jako na auto.

Je důležité si uvědomit, že na rozdíl od klasického systému dědičnosti je možné entity snadno rozšířit tak, že do nich přidáme další komponenty. Můžeme například k entitě auta přiřadit komponentu životy a udělat ho tak zranitelným.

System

Systém provádí určité akce nad entitami, které mají dané komponent. Chceme-li například propojit grafickou reprezentaci s fyzikálním enginem, můžeme si napsat systém, který po kroku fyzikálního enginu vyhledá všechny entity, které mají grafickou a fyzikální komponentu upraví pozice a rotaci všech grafických objektu tak, aby byla identická s pozicí a rotací fyzikálních objektů, které jsou k ním přiřazeny.

6 Vlastní práce

Vlastní práce se skládá ze dvou částí klientská část, která slouží k vizualizaci algoritmu a zobrazení výsledků ze serverové části. Serverová část pro maximální urychlení simulace.

7 Simulace

Simulace je realizovaná jako knihovna pro Node.js, lze jí tedy použít jak u klientské částí, tak u serverové části. Poskytuje kompletní fyzikální simulaci agenta, prostředí ve kterém se pohybuje, jeho ovládání a výpočet fitness funkce. Součástí simulačního prostředí je také kód pro její vizualizaci.

Při návrhu simulace bylo dbáno především na následující kriteria:

1. Rychlost - Simulace musí být, co nejrychlejší s ohledem na to, že jich bude třeba pustit tisíc pro vyhodnocení jediné generace
2. Nenáročnost - Simulace musí běžet na různém hw v různých konfiguracích (viz sekce 8.1)
3. Robustnost - Simulace by si měla poradit s neočekávanými vstupy, jako je třeba **NaN**, který vychází z neuronové sítě
4. Stejné prostředí - Simulace musí poskytovat stejné prostředí pro všechny jedince
5. Portabilita - bylo třeba zajistit, aby šlo kód rozběhnout v různých platformách v různých konfiguracích.

Rychlost byla zajištěna implementací profilovacího programu (**benchmark.js** ve složce simulation), který spouští simulaci na předem připravené populaci jedinců. Výstupem je pak doba, za jakou jí vyhodnotil na jednom jádře procesoru. Tento údaj byl pak používán při implementaci simulace pro orientační představu, jak moc případné změny v kódu ovlivňují rychlost samotné simulace. Dále byla simulace podrobena občasnému profilování v klientské částí s pomocí vývojářských nástrojů prohlížeče chrome, na kterém simulace jede nejlépe.

Nenáročnost, která souvisí s rychlostí pak byla zajištěna tím, že bylo v průběhu psaní kódu dbáno na to, aby v průběhu simulace nedocházelo k přebytečným alokacím, které by nejen že mohli způsobit přebytečný nárůst požadované paměti ale způsobovali by také nepředvídatelné zpomalení, které s sebou přináší jazyk využívající garbage kolektor.

Robustnost je podrobněji vysvětlená v sekci 7.2 a popis toho, jak bylo dosaženo stejných podmínek pro všechny agenty lze nalézt v návrhu 7.1 především v popisu RoadManageru.

7.1 Části ECS použité v simulaci

Simulace je implementovaná v duchu ECS (**Entity component system**) podrobný popis lze nalézt v sekci 5.7. Není tedy žádným překvapením, že se všechny komponenty nalezené v simulaci dají rozložit na systémy, komponenty a entity. Pro lepší představu o implementaci je níže uveden přehled všech systému, entit a komponent použitých v simulaci.

Entity

Simulace obsahuje následující entity:

PhysicsGroup Seskupuje fyzikální entity do jedné pro snadnou manipulaci s nimi.

RoadPart Entita, která obaluje jednu nebo více překážek tak, aby se s nimi dalo snadno pohybovat používá se pro tvorbu složitějších dílů vozovky.

Car Reprezentuje samotné vozidlo obsahuje jak jeho grafickou reprezentaci, tak kompletní logiku a fyzikální model.

Komponenty

Simulace obsahuje následující komponenty: **Car** obsahuje všechny potřebné informace o agentovi. Toto zahrnuje vše od neurnové sítě, která je použitá pro jeho řízení po ovládání jednotlivých kol agenta.

Graphics komponenta, která obsahuje grafické informace pro **Pixi.js**.

Physics komponent, která obsahuje fyzikální entity pro **P2.js**

Systémy

Simulace obsahuje následující systémy:

Car systém, který se stará o ovládání agenta a částečně o vyhodnocování jeho fitness.

Graphics grafický systém, který slouží především k překreslování entit s pomocí **PIXI.js**

Physics krokuje fyzikální engine a synchronizuje grafickou reprezentaci s reprezentací fyzikální

RoadDirector Road director se stará o generování nekonečného prostředí pro agenta. Děje se tak na základě předefinovaných dílu vozovky z nichž každý zaplňuje celou obrazovku simulace. V případě, že agent dorazí až na konec obrazovky je mu určen nový navazující dílek. Agent je pak přehozen na opačnou stranu obrazovky a zároveň je vyměněn díl na kterém se nachází. Hlavní výhodou tohoto přístupu je to, že agent může jezdit po vozovce donekonečna bez starosti o to, že by se dostal na limit fyzikálního enginu (přetečení pozice fyzikálního objektu). Další nesporná výhoda tohoto přístupu je úspora paměťových nároků, kterou by s sebou nesla definice větší mapy a možnost generování náhodných map pro testování agenta.

7.2 Fitness funkce

Fitness funkce je důležitou součástí simulace, která zásadně ovlivňuje chování výsledných agentů a je tedy nutné jí volit vhodně. Je nutné, aby funkce agenta motivovala ke správné činnosti.

Po několika pokusech a konzultaci s vedoucím práce byla jako metrika úspěchu agenta zvolena celková vzdálenost, kterou je agent schopný překonat v průběhu

jedné generace. Výpočet je realizován s pomocí RoadDirectoru, který si při každém přechodu zaznamená bod, ve kterém se po přesunu agent nachází. Výsledná fitness je pak součet uražených vzdáleností pro každou místnost. Road direktor si pro každou obrazovku uchovává vzdálenost, kterou agent v dané obrazovce překonal. Výsledným fitness je pak součet všech vzdáleností na všech obrazovkách

Agenta je ovšem kromě motivace třeba také penalizovat za akce, které jsou nepřípustné. V případě simulace se jedná především o kolizi s překážkou, za což je agent penalizován předčasným ukončením simulace a nemožností tedy zvýšit svoji fitness.

Agent

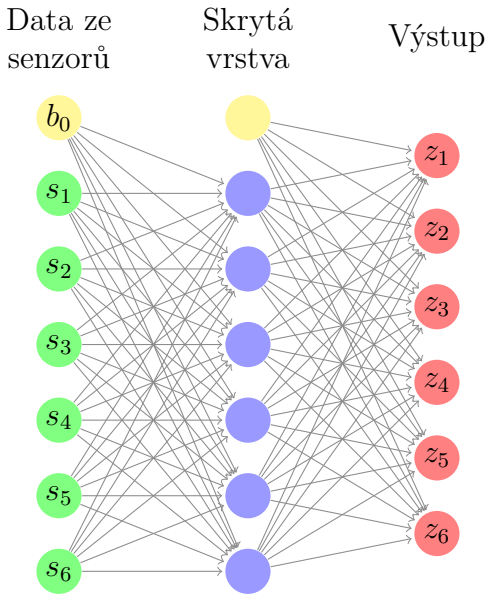
Definice samotného agenta zásadně ovlivňuje výsledek simulace, protože stanovuje vstupy a výstupy do a z neuronové sítě.

Samotným agentem je auto, které je vybaveno 6 vzdálenostními senzory. Měření těchto senzorů je normalizováno (maximální vzdálenost měřícího paprsku je 800 m) a předáno jako vstup do neuronové sítě.

Agent se poté každý snímek s pomocí neuronové sítě 6 rozhoduje, jakou akci podnikne. Má následující možnosti:

1. Ovládání volantu
   a) z_1 Otočení volantem o určitý počet stupňů doleva
   b) z_2 Otočení volantem o určitý počet stupňů doprava
2. Rychlostní stupně
   a) z_3 - zpátečka
   b) $z_4 - z_6$ - rychlosti dopředu

Ovládání volantu i volba rychlostního stupňů probíhá zároveň a to tak, že se vždy z dané skupiny neuronů vybere ten, který má největší hodnotu. Tento přístup je identický tomu, který se používá například u neuronových sítí pro klasifikaci.



Obrázek 6: Neuronová síť agenta

8 Serverová část

Serverová část vyhodnocuje jednotlivé jedince distribuovaně s pomocí fronty úkolů. Frontu poskytuje knihovna **bull**, která používá **redis** pro správu údajů o jednotlivých úkolech.

Cílem byl návrh robustního systému, který v ideálním případě rozloží výpočetní zátěž mezi jednotlivé uzly rovnoměrně. Dalším požadavkem byla možnost odpojení kdykoliv kteréhokoliv z počítačů, jelikož ne všechny bylo možné nechat běžet přes noc.

8.1 Výpočetní cluster

Ukázalo se, že vyhodnocování simulace zabírá neúměrné množství času a to i na nejvýkonnějším dostupném počítači. Například vyhodnocení jedné generace populace o 1024 jedincích zabralo 290 s na nejsilnějším dostupném pc. Z tohoto důvodu bylo rozhodnuto o distribuce výpočetní zátěže mezi více počítačů. Byl vytvořen výpočetní cluster se specifikací popsanou v tabulce 1.

Procesor	RAM	Počet	Architektura
S5P6818 Octa core	1 GB	2	arm64
Broadcom BCM2837B0 quad-core	1 GB	1	arm32
Phenom X4 965	8 GB	1	x64
Intel Core i5-2300	4 GB	1	x64
AMD A4-4300M	4 GB	1	x64
Intel atom x5-Z8350	2 GB	1	x64
Cortex-A5	1 GB	1	armv7l

Tabulka 1: Použitý hardware

Docker swarm

Pro snadnou distribuci a správu byly všechny počítače zorganizovány do docker swarmu. Docker swarm obsahoval jednoho managera (Broadcom BCM2837B0 quad-core), který zároveň spouštěl klientskou aplikaci a další služby:

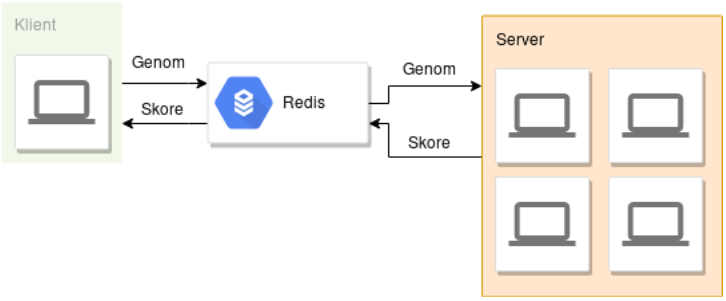
1. **Portainer** pro správu clusteru
2. **Arena** webové ui pro správu **bull**
3. **redis** používaný knihovnou **bull**

Použití docker swarmu umožňuje především snadné nasazení a správu zpracovávajících procesů. Zároveň zajišťuje, že všechny instance zpracovatelů mají unifikovanou konfiguraci, což je zvláště důležité pro dosažení konzistentních výsledků.

8.2 Průběh vyhodnocování

Serverová část pracuje dle diagramu 7, kde je vidět, že klient zadává do fronty úkoly (genom a nastavení simulace). Jednotliví zpracovatelé (počítače v clusteru), kteří si je z ní vyberou, jednotlivé genomy vyhodnotí a hodnotu fitness funkce pošlou zpět na klienta.

Jakmile klient dostane všechny hodnoty zpět provede na populaci genetický algoritmu (mutace, křížení, …) a poté je nová generace poslána znovu na vyhodnocení.



Obrázek 7: Schéma distribuovaných výpočtů

Tento přístup má několik výhod a to:

1. Robustnost - Pokud jeden nebo více zpracovatelů selže (je například odpojen ze sítě) je možné pokračovat ve vyhodnocování (neúspěšný úkol lze vrátit zpátky do fronty). Toto v kombinaci s výše zmíněným docker swarmem znamená, že jakýkoliv výpočetní uzel lze kdykoliv vypnout a po znovu zapojení do sítě si načte nejnovější konfiguraci a začne znovu vyhodnocovat bez potřeby jakékoliv manipulace s jakoukoliv částí swarmu.
2. Dobré rozložení zátěže - Jelikož si zpracovatel vytahuje úkoly z fronty, je vždy optimálně zatížen, a není třeba řešit rozložení mezi různě výkonnými a zatíženými počítači.
3. Škálovatelnost - problém lze škálovat až do doby, kdy počet procesorů nepřesáhne počet potřebných simulací. Chceme-li tedy vypočítat generaci o tisíci jedincích můžeme na ně nasadit až tisíc procesorů.

Lze i namítnout, že se zde projevuje určitá režie při síťové komunikaci se serverem, což může být zdrojem určitého zpomalení. Nicméně se toto zpomalení neprojevilo v průběhu testování clusteru. Právě naopak bylo naměřeno 10 násobné zrychlení oproti výpočtu na jednom počítači.

9 Klientská část

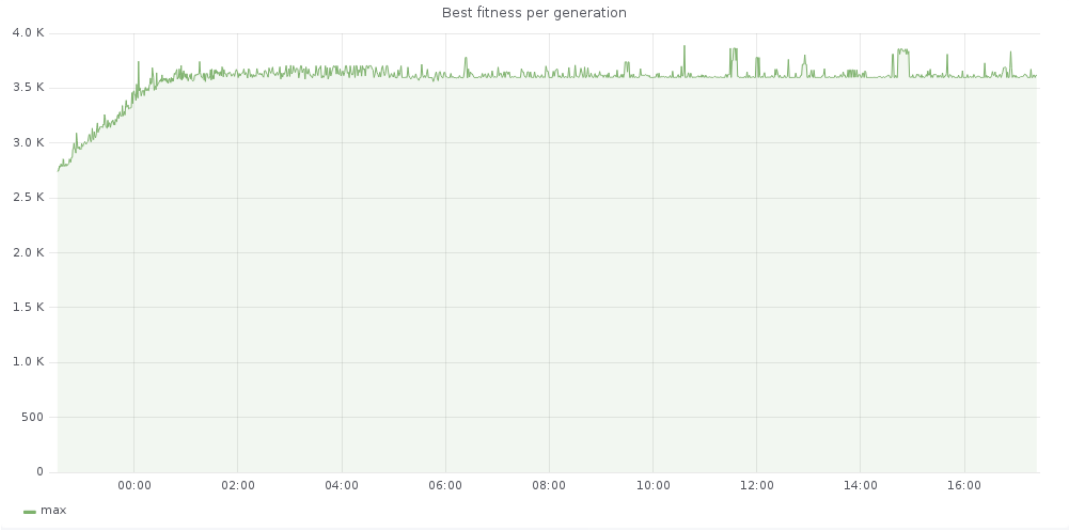
Klientská část byla navržena tak, aby byla schopná vizualizovat průběh algoritmu NEAT a zároveň měla možnost znovu vyhodnocení existujících genomů vygenerovaných serverovou částí. První požadavek vznikl na základě konzultace s vedoucím, který chtěl algoritmus neurovoluce demonstrovat v hodinách předmětu VUI2. Druhý požadavek vznikl z důvodu potřeby vizualizace řešení, které generoval sever.

10 Experimenty

Po návrhu simulačního prostředí byl agent vyzkoušen v několika situacích se stupňující se obtížností. Každá simulace probíhala s 1000 jedinci po 2000 generací. Ačkoliv je pravděpodobné, že by delší doba evaluace by pravděpodobně vyústila v lepší výsledky její výpočet v různých konfiguracích se ukázal jako příliš časově náročný navíc empirické pozorování ukázalo, že tato konfigurace poskytuje dostatečně dobré výsledky za snesitelný čas.

10.1 Nekonečná silnice ve tvaru I

Agent byl umístěn do nekonečné rovné silnice ve tvaru I. Cílem bylo pozorovat, zda se agent bude schopný naučit řídit rovně. Agent po tisící generací dosáhl fitness 3 500 a naučil úspěšně kývavým pohybem udržet uprostřed vozovky.



Obrázek 8: Fitness agenta v průběhu času

11 Možná vylepšení

Tato kapitola se bude zabývat možnými vylepšení současného řešení.

12 Závěr

13 Reference

[BUDUMA, Nikhil 2017] BUDUMA, Nikhil. *Fundamentals of deep learning: designing next-generation machine intelligence algorithms.* Sebastopol: O'Reilly, 2017. ISBN 978-149-1925-614..

[PATTERSON, Josh. 2017] PATTERSON, Josh. *Deep learning : a practitioner's approach Deep learning : a practitioner's approach. 1.* Beijing ; Boston ; Farnham ; Sebastopol ; Tokyo: O'Reilly, 2017. ISBN 978-1-491-91425-0..

[MITCHELL, Melanie., 1996] MITCHELL, Melanie. *An introduction to genetic algorithms.* Cambridge: Bradford Book, c1996. ISBN 0-262-13316-4..

[HYNEK, Josef., 2008] HYNEK, Josef. *Genetické algoritmy a genetické programování.* Praha: Grada, 2008. Průvodce (Grada). ISBN 978-80-247-2695-3..

[LÝSEK Jiří, ŠŤASTNÝ Jiří, 2014] LÝSEK, Jiří a ŠŤASTNÝ, Jiri. (2014). *Automatic discovery of the regression model by the means of grammatical and differential evolution.* Agricultural Economics (AGRICECON). 60. 546-552. 10.17221/160/2014-AGRICECON. .

[STANLEY, Kenneth O, Risto MIIKKULAINEN., 2002] STANLEY, Kenneth O. a Risto MIIKKULAINEN. *Evolving Neural Networks through Augmenting Topologies.* In: Evolutionary Computation [online]. 2002, 10(2), s. 99-127 [cit. 2018-12-08]. DOI: 10.1162/106365602320169811. ISSN 1063-6560. Dostupné z: http://www.mitpressjournals.org/doi/10.1162/106365602320169811.

Přílohy

A CD se zdrojovým kódem