STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 18: Informatika

Implementace a rozbor algoritmu NEAT pro problémy reinforcement learningu

Antonín Říha
Královehradecký kraj

Červená Třemešná 2018

STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 18: Informatika

Implementace a rozbor algoritmu NEAT pro problémy reinforcement learningu

Implementation and analysis of NEAT algorithm for problems of reinforcement learning

Autoři: Antonín Říha
Škola: Vyšší odborná škola a Střední průmyslová škola, Jičín
Kraj: Královehradecký kraj
Konzultant: Mgr. Ondřej Kořínek, Ph. D.
Červená Třemešná 2018

Prohlášení

Prohlašuji, že jsem svou práci SOČ vypracoval/a samostatně a použil/a jsem pouze prameny a literaturu uvedené v seznamu bibliografických záznamů.

Prohlašuji, že tištěná verze a elektronická verze soutěžní práce SOČ jsou shodné.

Nemám závažný důvod proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších předpisů.

V Červené Třemešné dne 12. 3. 2018 ……

Antonín Říha

Anotace

Tato práce se zabývá algoritmy neuroevoluce pro problémy reinforcement learningu. Tyto algoritmy se postupně ukazují jako dobrá alternativa ke klasickému deep learningu. V rámci práce byla implementována jedna z verzí takového algoritmu. K tomuto algoritmu bylo připojeno několik rozšíření a navrhnuto i několik vlastních. V rámci práce bylo zpracováno mnoho témat souvisejících s tématikou a implementace byla otestována na problémech reinforcement learningu.

Klíčová slova

Neuroevoluce; umělá inteligence; neurální sítě; reinforcement learning

Annotation

This paper desribes algorithms of neuroevolution for problems of reinforcement learning. These algorithms have proven themselfs to be a good alternative for classical approach of deep learning. In this paper was implemented one version of such algorithm. To this algorithm was added few extensions and proposed some nwe ones. This project described many topics concerning neuro evolution and tested its implementation on problems of reinforcement learning.

Keywords

Neuroevolution, artificial intelligence, neural networks, reinforcement learning

Obsah

1 Úvod

V posledních letech, zejména díky nárůstu výpočetní síly a množství dat, se obor strojového učení/machine learning (ML) dostává do popředí informatiky. V pouhých několika letech byl schopný udělat velký pokrok, na do té doby neřešitelné problémy (AlphaGO [1], Dota 2 [2],…). Tyto úspěchy způsobily obrovský rozvoj oboru a přinesly spoustu různých přístupů k řešení problémů. Aktuálně nejrozšířenějším z nich je tzv. deep learning, který slaví velký úspěch na mnoha problémech (optické rozpoznávání znaků, rozpoznávaní hlasu, medicína, …), ale poslední dobou se začíná často používat i jiný přístup, který je založený na genetickém programování. Jedná se o tzv. neuroevoluci, která se ukázala být jako dobrá alternativa deep learningu, v některých problémech se dokonce ukázala jako lepší volba (zejména v problémech posilovaného učení).

Tato práce se zabývá algoritmem NEAT [3], který je příkladem jednoho z algoritmů neuroevoluce. Tento algoritmus přináší několik nových myšlenek, zejména rozdělení genomů na rasy a neuroplasticitu (schopnost měnit topologii vyvíjené neurální sítě). Tento algoritmus se ukázal jako dobrá alternativa pro komplexní problémy posilovaného učení a inspiroval autora algoritmu k jeho dalšímu rozšíření.

Prvním takovým rozšířením je HyperNEAT [4] a na něj navazujícím ES-HyperNEAT [5], které se snaží řešit problém algoritmu s vývojem komplexních struktur pomocí tzv. nepřímého dekódování. Tento přístup se ukázal být velmi užitečným zejména, pokud je nutné zpracovávat vstup v kartézském systému souřadnic. To těmto přístupům umožňuje vynikat v některých úlohách, které například pracují s vizuálním vstupem.

V práci byly zpracovány některé problémy genetických algoritmů a byl navrhnut přístup, který se je snaží adresovat pomocí kombinace novelty a fitness search. Tento přístup (CNAOS-NEAT) se ukázal jako dobrá alternativa ke klasickým přístupům zejména v některých typech deceptivních domén.

V práci bylo navrženo několik experimentů za účelem dokázání schopností algoritmu v různých typech úloh. Experimenty samotné se zaměřují na různé problémy a snaží se dokázat různé schopnosti testovaných algoritmů a zároveň umožňují interaktivní sledování průběhu vývoje.

2 UMĚLÁ INTELIGENCE

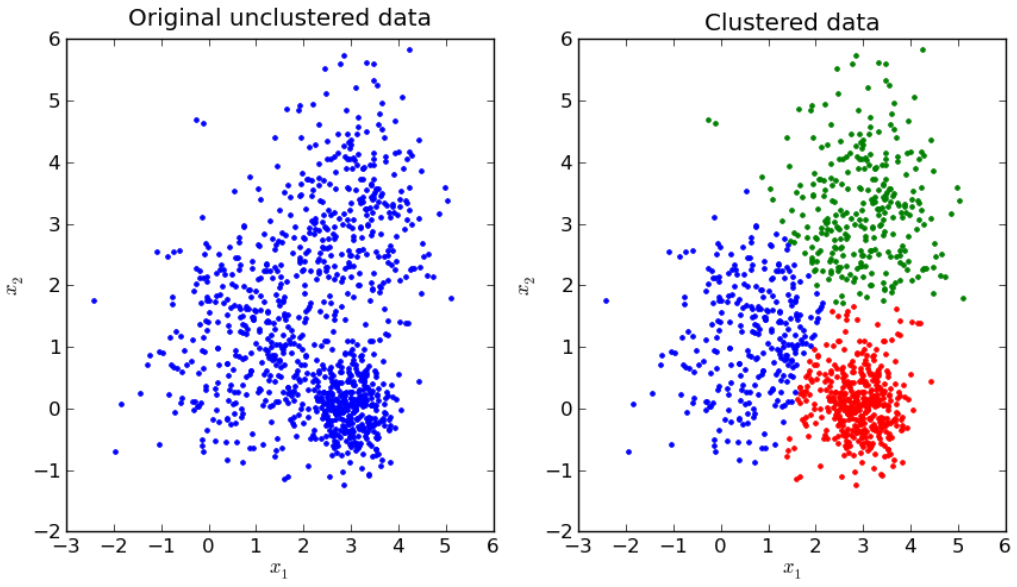
V následující kapitole budou popsány teoretické základy dnešního přístupu k umělé inteligenci a strojovému učení. Budou zde vysvětleny rozdíly mezi jednotlivými druhy učení a také zde bude ukázáno, které typy problémů používají daný přístup.

2.1 Unsupervised learning (učení bez učitele)

Tento přístup vychází z toho, že v datech jsou hledány určité spojitosti nebo vzory. Hlavním rozdílem oproti ostatním přístupům (viz níže) je absence vyhodnocování. Hlavní využití těchto algoritmů je v případech tzv. clusteringu dat. Toho lze využít například v odlišování různých částí obrázků, nebo řazení nasbíraných obrázků do kategorií (zvířata, nábytek, budovy, …).

2.1.1 K-means clustering

Tento algoritmus je jedním z velmi oblíbených v oblasti unsupervised learningu. Umožňuje rozdělení dat v *n*-dimenzionálním prostoru do *k* skupin. Funguje na principu hledání středového bodu s nejmenší průměrnou vzdáleností k ostatním bodům v prostoru. Tímto postupem vzniká oblast bodů neboli cluster, který se vyznačuje podobnými vlastnostmi.



Obrázek 1 Ukázka algoritmu K-Means [13]

2.2 Supervised learning (učení s učitelem)

Tento přístup se liší od unsupervised learningu tím, že má přístup k datům, která jsou určitým způsobem ohodnoceny (často ručně). Snaží se tedy najít spojitost mezi daty samotnými a uděleným hodnocením. Toho se dá využít u problémů, u kterých je znám výsledek, ale není znám způsob, kterým se k výsledku dopracovat.

V praxi se vstupní data pro tento přístup dělí na dvě části. Na první části se algoritmus učí a snaží se co nejvíce přiblížit hledanému výsledku. Druhá část slouží pro kontrolu toho, jestli se algoritmus přibližuje námi hledanému spojení dat a jejich hodnocení. Tento přístup se používá proto, že je možné, aby se algoritmus „přeučil“. V tomto stavu je schopen správně klasifikovat (často s velmi vysokou úspěšností) pouze vstupní data, ale není již schopen klasifikovat jakákoliv jiná. O takovém stavu lze prohlásit, že si algoritmus „zapamatoval“, ke kterým datům má přidělit které hodnocení, ale nepřišel na spojitost, která toto ohodnocení určuje.

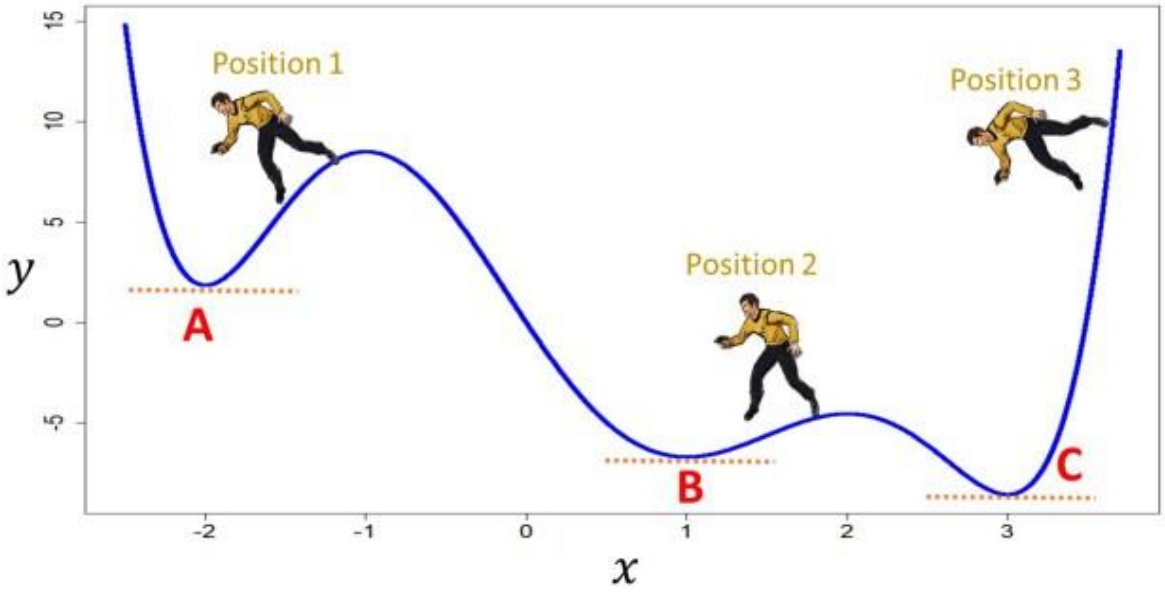
Příkladem takového problému je rozpoznávání ručně psaných znaků (číslic, písmen, …). U takového problému je výstup znám (je možné rozpoznat znak), ale neexistuje přesný postup, kterým se k výstupu dopracovat. To je jedním z mnoha problémů, ve kterých nalézá supervised learning uplatnění.

Toto široké uplatnění a existence množství nástrojů, které lze pro tento typ problémů použít (TensorFlow, pyTorch, IBM Watson, …), způsobilo, že supervised learning je aktuálně nejrozšířenějším typem umělé inteligence a jeho popularita nadále roste. Dalším příkladem takového problému je rozpoznání hlasového vstupu (např. virtuální asistenti), systém doporučování předmětů v e-shopu a mnoho dalších.

2.2.1 Gradient descent

Jelikož je v této oblasti nespočet různých algoritmů, bude zde zmíněn pouze jeden z nejpoužívanějších, a to gradient descent. Ve své nejjednodušší formě tento algoritmus pracuje na principu postupování po jakémsi „svahu“. V podstatě vyzkouší změnit hledaný parametr o určitý krok na obě strany a pozoruje změnu na výstupu.

Tímto způsobem se snaží naleznout lokální minimum, které nejvíc minimalizuje chybu klasifikace. Hlavním problémem tohoto přístupu je právě ono lokální minimum, jelikož algoritmus postupuje postupně. Je možné, že „uvízne“ v tomto lokálním minimu a nenalezne optimální globální minimum nebo alespoň optimálnější lokální.



Obrázek 2 Vizualizace algoritmu gradient descent [14]

2.3 Posilované učení (učení ze zkušenosti – reinforcement learning)

Tato oblast problému je pravděpodobně nejzajímavější a zároveň nejtěžší oblastí umělé inteligence a zároveň hlavním zaměřením této práce. Pro tento přístup je typické, že agent na základě pozorování vykonává akci. Tato akce je poté algoritmem vyhodnocena a je u ní určena odměna v rámci řešeného problému (domény).

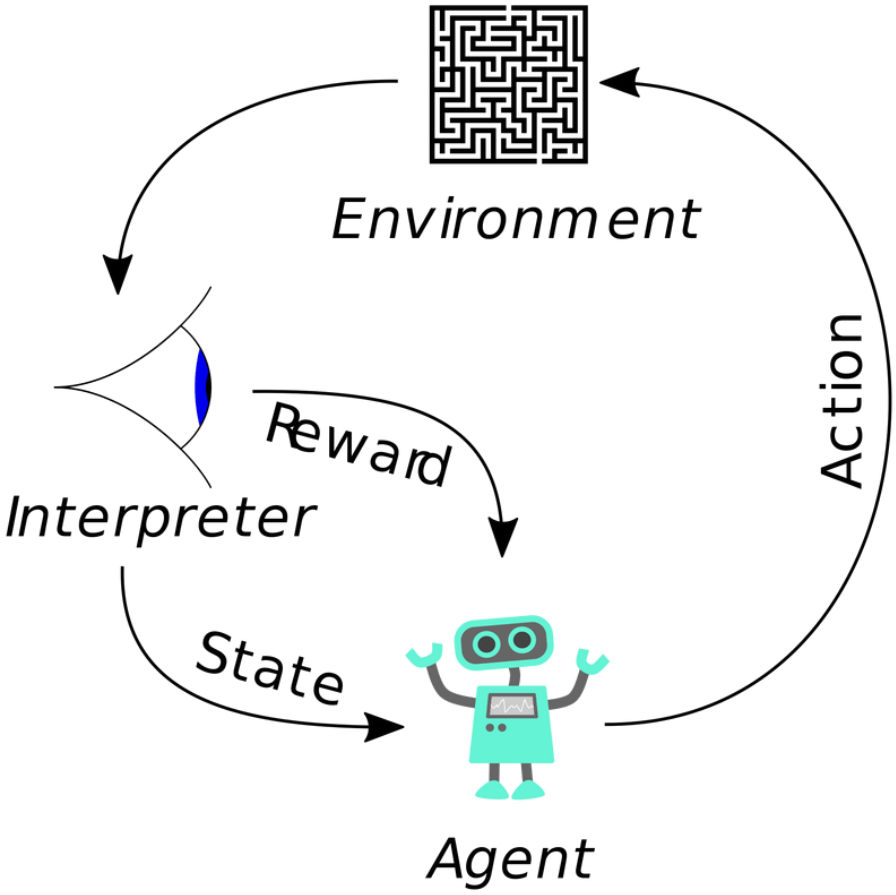
Tento přístup se nejčastěji používá pro modelování situací, ve kterých má agent (řízená entita) reagovat na vstupy (data z prostředí) a podle nich vyprodukovat výstup (akce), kterým ovlivňuje vstup (ovlivnit prostředí).

Příkladem takového úkolu je řízení robota. Robot zpracovává okolní prostředí senzory, které vyprodukují vstupní data pro agenta. Ten rozhodne, jak se má robot zachovat (pohyb pomocí motorů) a tím se posune dál. Cílem takového robota může být dostat se na konec bludiště. V tomto příkladu je možné vygenerovat vstupní data, pomocí výstupních ovlivnit prostředí a je možné vyhodnotit, jak dobře si robot vedl (vzdálenost k cíli). Není ale možné (většinou) určit, jaká by byla optimální strategie a tím přesně popsat, jak se má robot chovat, aby se dostal k cíli. Tyto vlastnosti problém činí problémem posilovaného učení.

Zajímavou částí tohoto problému je vyvažování tzv. „exploration“ a „exploitation“. Exploration je snaha agenta prozkoumávat doménu a hledat nová řešení. V rámci předchozího příkladu by se mohlo jednat o opuštění slibné cesty pro prozkoumání nové. Exploitation je snaha agenta držet se již známých řešení, o kterých ví, že nabízejí velké odměny.

Dalším problémem tohoto přístupu je, že některé testy agentů mohou být velmi nákladné. Například by se mohlo jednat o problém samo řídících automobilů. Pokud by agenti měli často v průběhu učení selhat (havarovat), k čemuž by v průběhu učení došlo, znamenalo by to

obrovskou finanční ztrátu. Z tohoto důvodu se velmi často pro tento přístup nejdřív sestaví co nejpřesnější simulace problému (pokud se problém řeší v reálném světě), na kterém se agent může naučit, jak se má chovat. Tento problém není přítomný pouze pro posilované učení, ale je v něm nejznatelnější, proto byla tato poznámka zařazena do této kategorie, ale může platit i pro ostatní typy učení.



Obrázek 3 Vizualizace posilovaného učení [15]

2.3.1 Monte Carlo

Jedním z přístupů k hledání optimálního chování je Monte Carlo. Tento přístup funguje pouze pro malé problémy s konečným množstvím řešení. V tomto přístupu zkoušíme náhodné chování na problému a vyhodnocujeme, jak si vedlo v rámci jednotlivých kroků simulace. Tento přístup lze aplikovat opakovaně v rámci tzv. epizod a díky tomu je možné přiblížit se optimálnímu řešení v doméně. Je možné toto upravit na tzv. TD metody, které fungují podobným způsobem, ale v rámci jednotlivých kroků, což jim často umožňuje dosáhnout lepších výsledků.

2.3.2 Evoluční přístupy

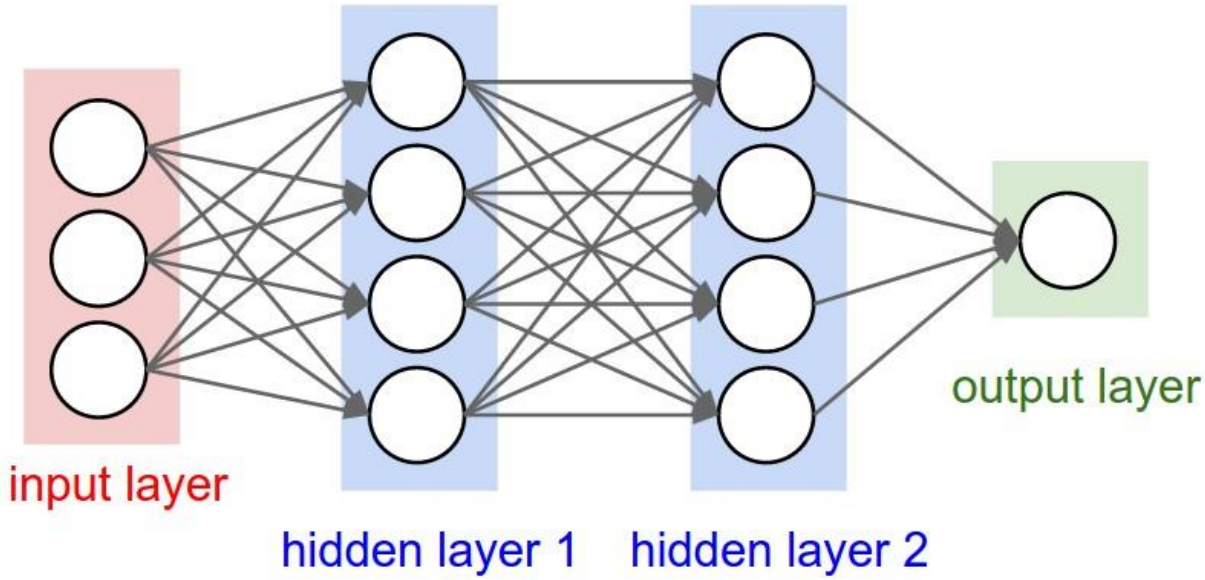
Jednou z možností, jak k tomuto učení přistupovat, je přes evoluci. Tento přístup bude dále popsán v kapitolách genetické algoritmy a NEAT.

3 NEURÁLNÍ SÍTĚ

Velmi dominantním modelem pro umělou inteligenci (mimo unsupervised learningu) jsou tzv. neurální sítě. Tento model, přesněji nazván umělá neurální síť (artificial neural network – ANN), se snaží volně napodobovat strukturu organického nervového systému (mozku). Je na místě upozornit, že se od svého organického protějšku výrazně liší a jedná se pouze o velmi volnou reprezentaci.

Tento model se skládá z neuronů, které jsou uspořádány v jednotlivých vrstvách. Tyto vrstvy se nazývají vstupní, výstupní a pro zbylé vrstvy se souhrnně používá název skryté. Neurony v těchto vrstvách jsou propojeny propojeními, které jsou definovány svými váhami. Neurony samotné jsou definovány vstupními a výstupními propojeními a aktivační funkcí. Výstupem takového neuronu je suma součinů všech výstupů vstupních neuronů a vah jednotlivých propojení. Tato suma je potom aplikována na aktivační funkci, která určuje výstup neuronu. Častým speciálním vstupem neuronu je také tzv. bias. Tento vstup je konstantní a používá se hlavně z důvodu nulového vstupu. V některých případech nulový vstup do neuronu nemá znamenat nulový výstup, a proto se zavádí konstantní vstup, který vstup posouvá a tento problém řeší.

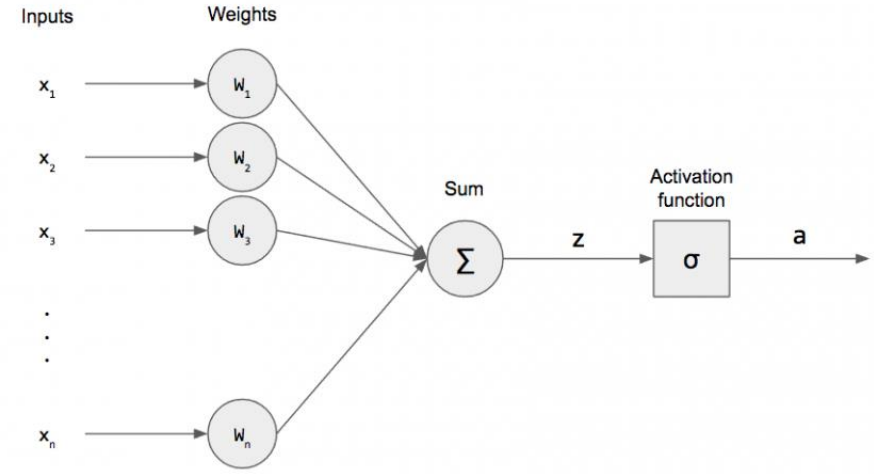
Aktivační funkcí neuronu může být téměř jakákoliv funkce, ale typicky se používá několik konkrétních. Nejčastěji se jedná o sigmoidu, RELU („rampa“), Gaussovu funkci, identitu, trigonometrické funkce a mnoho dalších s tím, že se seznam neustále rozšiřuje.



Obrázek 4 Ukázka schéma neurální sítě [16]

3.1 Perceptron

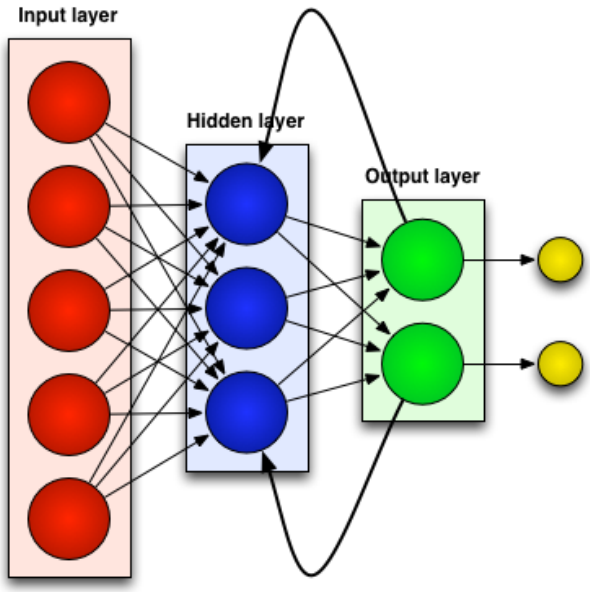
Perceptron byl jedním z prvních modelů umělé neurální sítě. Tento model se skládal pouze z jediného neuronu a jednalo se tak o lineární klasifikátor, který bral jako svůj vstup matici hodnot, které klasifikoval na základě svých parametrů (vah propojení, aktivační funkce).



Obrázek 5 Ukázka perceptronu – jednoho neuronu [17]

3.2 Rekurentní neurální sítě

Zvláštním případem neurální sítě je tzv. rekurentní neurální síť. V takové síti existuje neuron, jehož výstup je přímo nebo nepřímo napojen na svůj vstup. Výhoda takové neurální sítě spočívá v schopnosti uchovávat informace a nalézá uplatnění v sítích, u kterých minulý vstup souvisí s aktuálním. Příkladem takového problému je řízení robotů, rozpoznávání ručně psaných znaků nebo rozpoznávání řeči.



Obrázek 6 Ukázka schéma rekurentní neurální sítě [18]

4 GENETICKÉ ALGORITMY

Genetické algoritmy jsou skupina algoritmů, které jsou inspirovány Darwinovou evolucí. Hlavními koncepty těchto algoritmů je přežití nejsilnějšího a zachování genetické diverzity. Tyto algoritmy se skládají z několika kroků.

4.1 Charakterizace

Základem těchto algoritmů je vytváření modelů pro řešení problému, které mají množství parametrů, které lze měnit. Taková změna může být např. negace bitů, nebo změna reálných hodnot vstupní proměnné (např. váha propojení neurální sítě). Zajímavostí tohoto přístupu je i možnost zvětšit komplexitu modelu. Nutností pro tyto algoritmy je genetická reprezentace řešení problému (např. neurální síť) a funkce, která je schopná ohodnotit tato řešení (tzv. fitness funkce).

Jednou ze zajímavých aplikací takových algoritmů je anténa, kterou NASA vytvořila pro svoji misi ST5 viz obrázek 7.

Obrázek 7 Anténa vyvinutá pro misi NASA [19]

4.2 Inicializace

Prvním krokem genetického algoritmu je vytvoření velké populace potencionálních řešitelů problému. Tato populace může být náhodná, nebo být uzpůsobena tak, aby tvořila řešení v oblasti, o které bylo zjištěno, že obsahuje potencionální řešení. To ovšem může negativně ovlivnit hledání, které hledá v oblasti s lokálním maximem přesto, že se nemusí nutně jednat o globální maximum.

4.3 Selekce

Dalším krokem je tzv. selekce. V této části jsou zvoleni jedinci populace, kteří si vedli nejlépe (jejich fitness skóre je nejvyšší) a tato část populace určuje, ze kterých genomů vznikne generace další.

4.4 Mutace

V této části se z vybraných jedinců vytvoří další generace populace řešení. Tato generace může vzniknout jedním ze dvou způsobů. Prvním je asexuálně, tedy mutací parametrů rodiče. Druhou možností je sexuální reprodukcí, ve které je nové řešení kombinací dvou (nebo více) řešení předchozí generace.

5 NEAT (NEUROEVOLUTION OF AUGMENTING TOPOLOGIES)

Jedním z genetických algoritmů, který postupně nabývá na popularitě je NEAT [3] neboli Neuroevolution of augmenting topologies. Tento algoritmus a jeho rozšíření jsou hlavním zaměřením této práce a většina dalšího obsahu práce tento algoritmus rozvíjí.

5.1 Základní charakteristika

Jedním ze základních principů NEATu a většiny genetických algoritmů je snaha začít od co nejjednoduššího řešení, které se postupně rozšiřuje tak, aby se přibližovalo optimálnímu řešení (dle dané fitness funkce).

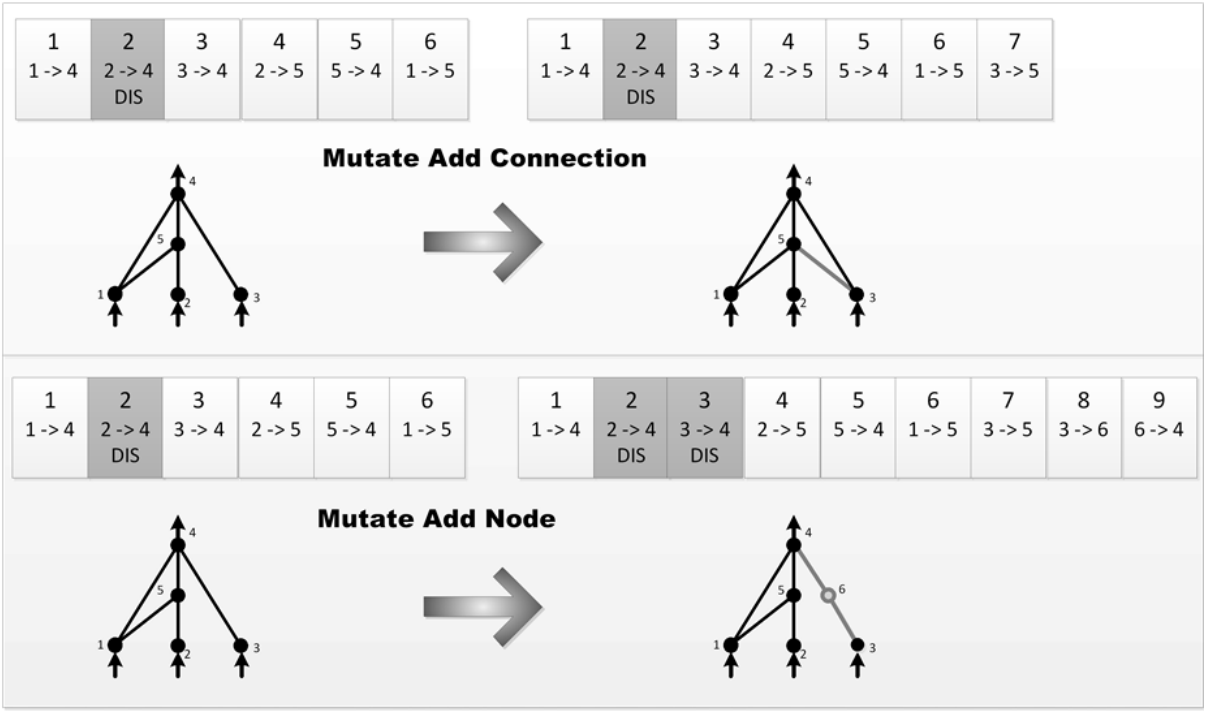
NEAT je algoritmem, který používá genetický algoritmus pro vývoj neurálních sítí. Hlavní výhodou modelu neurálních sítí je jeho modularita, možnost zvyšování komplexity a jeho schopnost reprezentovat složité struktury, což je dokázáno předchozím výzkumem.

Hlavním principem, který NEAT přináší ke klasickým genetickým algoritmům, je historické značkování a s ním související dělení genomů do ras. Tento přístup se ukázal jako velmi úspěšný, zejména v oblasti posilovaného učení.

5.2 Historické značkování

Jedním ze základních konceptů NEATu je schopnost určit vzdálenost dvou genomů. Algoritmus je toho schopen dosáhnout pomocí techniky tzv. historického značkování. Toto značkování přiděluje každé inovaci v topologii sítě (přidání neuronu, přidání propojení) unikátní inovační číslo, viz obrázek 8. Na konci každé generace se vytvoří seznam všech inovací a zkontroluje se, zda neproběhla stejná změna topologie vícekrát. Pokud se tak stane, algoritmus přidělí všem stejným inovacím stejné inovační číslo.

Tato technika umožňuje stanovit vzdálenost mezi dvěma genomy. Takovou vzdálenost lze stanovit pomocí množství počtu inovačních čísel v genomech, které se neshodují a zároveň podle rozdílu vah shodujících se propojení.



Obrázek 8 Ukázka historického značkování [20]

5.3 Dělení do ras

Díky schopnosti určit genetickou vzdálenost mezi dvěma genomy je možné určit genetickou podobnost genomů. Toho tento algoritmus využívá pro rozdělování genomů do předem stanového počtu ras. Tyto rasy vznikají pomocí algoritmu K-means clustering, který by šlo řadit do algoritmů unsupervised learningu.

Hlavním důvodem pro takové dělení jsou dva související body: zachování diverzity řešení a ochrana inovací. Toho je docíleno tím, že jednotlivé genomy rasy sdílejí své fitness skóre. To umožňuje ochránit inovace v rámci rasy a zároveň podporovat diverzitu. Při vývoji může nastat situace, že inovace, která je pro vyřešení problému nezbytná, může zpočátku snižovat celkové fitness skóre genomu. Díky tomu, že se v rámci rasy sdílí fitness skóre, je taková inovace zachována a může se dále rozvíjet.

Tento systém ras mírně upravuje část genetických algoritmů selekci, ve které se nevybírá skupina genomů s nejvyšším fitness skóre, ale místo toho se přiděluje každé rase místo pro genomy v další generaci. Genomy, které mají fungovat jako rodiče jsou vybrány kombinací selekce a náhodného výběru, ve kterém mají genomy s nejvyšším fitness skóre nejvyšší šanci, že budou vybrány do další generace.

5.4 Typy mutací

NEAT definuje několik typů mutací, které slouží k přibližování genomu k řešení problému. Tyto mutace se dělí na dvě skupiny, a to sexuální a asexuální.

5.4.1 Asexuální mutace

V rámci algoritmu může v genomu dojít k jednomu ze tří typů asexuálních mutací, tj. mutace, u které je vstupem jeden genom a výstupem je též jeden změněný genom.

Prvním typem je změna vah propojení, u které se vybere náhodné propojení, u kterého se náhodně změní váha v předem definovaném rozsahu. Toto propojení je většinou nejmenším zásahem do samotné neurální sítě, a proto má největší pravděpodobnost, že nastane.

Druhým typem je přidání propojení. Tato mutace vybere náhodně dva neurony sítě a vytvoří mezi nimi nové propojení s náhodnou vahou. Toto propojení je výraznějším zásahem do topologie sítě, a proto je pravděpodobnost toho, že nastane, výrazně nižší.

Posledním asexuálním typem mutace je přidání neuronu. Tato mutace vezme náhodné propojení v síti a přidá mezi dva neurony, které spojuje nový neuron, který je spojen s původními neurony. Tato mutace je velkým zásahem do sítě, a proto se tento zásah minimalizuje tím, že vstupní propojení do nového neuronu nabývá váhy původního spojení a spojení z nového neuronu nabývá váhy jedna. Tato mutace má z důvodu velkého zásahu do sítě nejmenší šanci na provedení.

Speciálním případem mutace je změna biasu. Ten je této implementaci definovaný jako vstupní neuron s konstantním výstupem a je spojený se všemi neurony sítě. Tudíž je jeho mutace stejná jako mutace váhy propojení.

5.4.2 Sexuální mutace

Sexuální mutace je taková mutace, ve které dva rodiče vytvoří potomka. Taková mutace je v NEATu velmi jednoduchá právě díky systému historického značkování. Díky tomuto systému můžeme zjistit, které části sítě dvou rodičů jsou shodné a poté rozhodnout, které neshodné do sítě zařadíme.

6 HYPERNEAT

Algoritmus HyperNEAT [4] je rozšířením NEATu, které mu umožňuje vytvářet větší (složitější) neurální sítě a zároveň umožňuje vstupům a výstupům mít geometrický význam.

6.1 Základní charakteristika

HyperNEAT (hypercube NEAT) umožňuje umisťovat neurony (vstupy, výstupy a neurony skryté vrstvy) do tzv. hypercubu. Hypercube je geometrická struktura, která popisuje *n*-dimenzionální pravidelné těleso. Toto těleso podle počtu dimenzí nazýváme úsečka (n = 1), čtverec (n = 2), krychle (n = 3), …

Toto umístění neuronů umožňuje využít informaci, kterou jsme v původním NEATu nemohli, a to umístění vstup v prostoru. Příkladem takového umístění je například robot, který jako vstup používá dvourozměrnou matici senzorů a jako výstup určuje, kde se nachází střed kruhu. Pro tento úkol potřebuje robot vědět, který bod matice se nachází vedle kterého, a je tedy nutné tyto body umístit do geometrického útvaru. To umožňuje HyperNEAT za pomocí nepřímého dekódování a konvolučních neurálních sítí. Hlavní nevýhodou tohoto přístupu je to, že neurony skryté vrstvy je nutné umístit ručně.

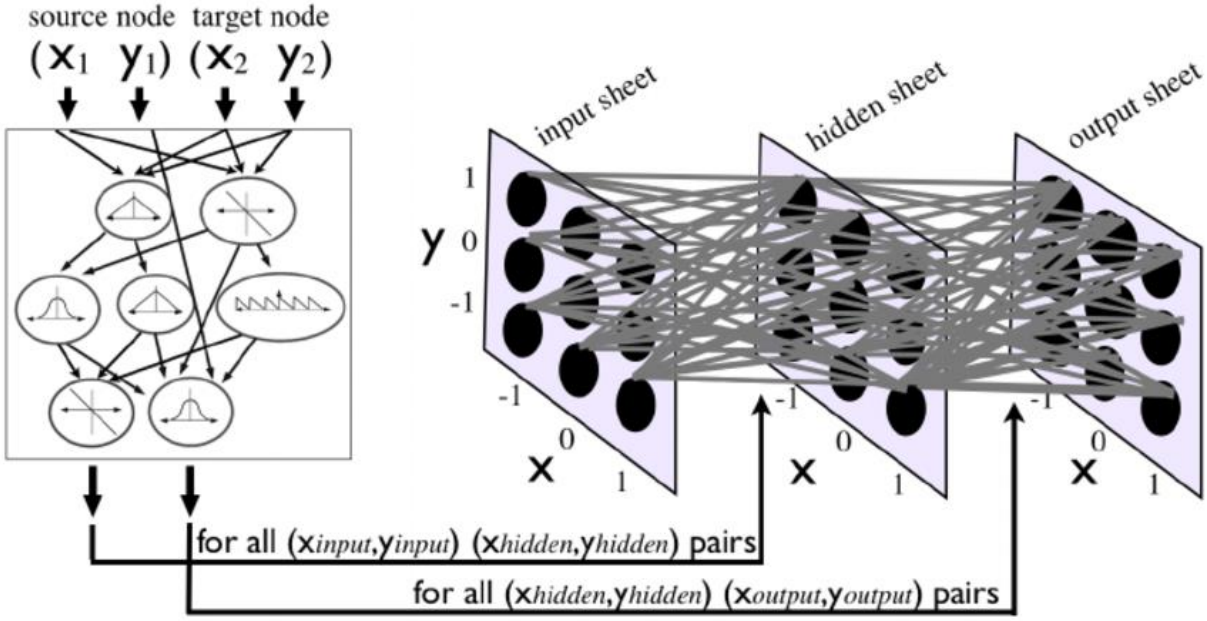
Zajímavostí na tomto přístupu je to, že u množství problémů, pokud jsou zachována stejná pravidla, je možné zvýšit množství vstupů a výstupů, a tedy zvětšit rozsah problému (například zvětšit snímanou oblast) a zároveň neztratit téměř žádnou schopnost agenta nalézt řešení.

6.2 CPPN

CPPN (compositional pattern producing networks) neboli konvoluční sítě jsou speciálním typem sítí, které se používají pro řešení problému v *n*-dimenzionálním prostoru. Tyto sítě se od klasických ANN liší tím, že využívají ve svých neuronech kombinaci mnoha aktivačních funkcí, zejména goniometrických, Gaussovy a sigmoidy. To těmto sítím umožňuje lépe vyjadřovat geometrické vzory nebo obrazce.

6.3 Nepřímé dekódování

Nepřímé dekódování je technika, ve které vyvinutý genom není přímým protějškem výsledné neurální sítě. V případě HyperNEATu je vyvíjená neurální síť (typu CPPN) použita k určení vah sítě ručně definované v hypercube. Toho algoritmus docílí tak, že vyvíjená síť má 2 x *n* vstupů a 2 výstupy, kde *n* je počet dimenzí hypercubu. Síť potom určuje váhy jednotlivých propojení tak, že na vstup vyvíjené sítě přivede pozice každých dvou neuronů v hypercube a určí váhu propojení podle hodnoty na prvním výstupním neuronu, viz obrázek 9. Druhý výstupní neuron je používán ke stanovení váhy propojení s biasem, který je určen jako speciální konstantní vstup na počátku souřadnic prostoru.



Obrázek 9 Vizualizace nepřímého dekódování [21]

7 ES-HYPERNEAT

ES-HyperNEAT [5] je rozšířením NEATu, které se snaží řešit hlavní problém HyperNEATu, kterým je nutnost ručně umisťovat neurony skryté vrstvy.

7.1 Základní charakteristika

Hlavní výhodou ES-HyperNEATu (evolvable-substrate HyperNEAT) oproti HyperNEATu je schopnost, nalézt pozici neuronů skryté vrstvy bez jejich ručního určení. Toho algoritmus dosahuje tak, že prostor postupně dělí a zjišťuje, jak velké množství informací se nachází v dané části hypercubu (neboli jaká je míra heterogenity). To umožňuje algoritmu vytvářet hustší síť v místech, kde je míra informací větší a tím vytvořit složitější model.

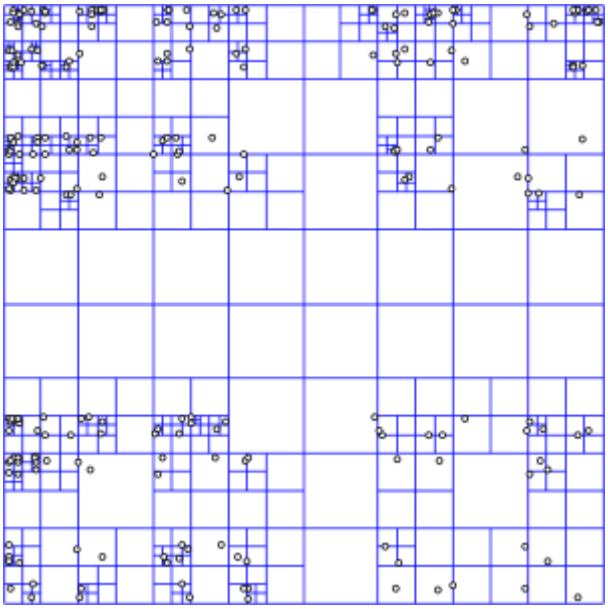
Tato vlastnost algoritmu vyžaduje najít algoritmus, který by umožňoval rozdělovat postupně prostor. Pro tento úkol byl zvolen algoritmus quadtree.

Druhou věcí, kterou je nutné si uvědomit je, že zhušťování sítě má smysl pouze do jisté míry. Každý problém obsahuje určitou hranici složitosti, za kterou už není žádná další informace, a proto je nutné si dávat pozor, do jak velké míry je třeba síť zhušťovat.

7.2 Quadtree

Tento algoritmus umožňuje postupně rozdělovat plochu na čtyři díly. Tento přístup je možné aplikovat opakovaně a vytvořit různé plochy, které jsou rozděleny do různé hloubky (hloubka je určení počtu aplikací algoritmu na daný kus). Tento algoritmus umožňuje vytvořit stromovou strukturu s různými stupni detailu a tím vyjádřit v různých částech různou míru informací viz obrázek 10.

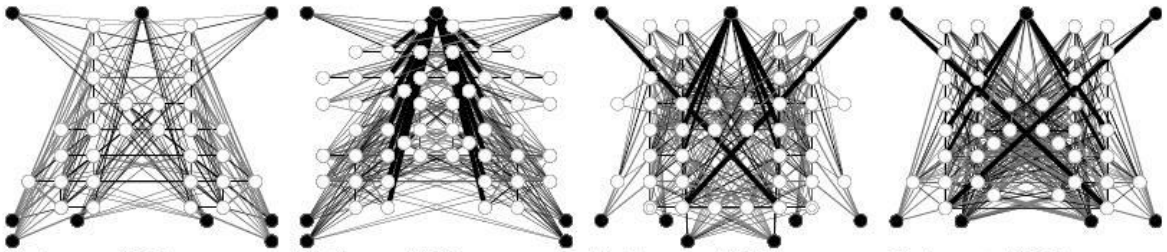
V této práci je použita implementace, která se nazývá hyperoctree. Tato implementace umožňuje dělit stejným způsobem *n*-dimenzionální prostor. Každým takovým dělením vznikne 2^n dětí. Tato implementace umožňuje pracovat i se složitějšími problémy, u kterých dvou dimenzionální vyjádření není dostatečné.



Obrázek 10 Ukázka dělení pomocí quadtree [23]

7.3 Určování pozice neuronů skryté vrstvy

Hlavní informací pro určení pozice neuronů skryté vrstvy je určení množství informací v dané oblasti. Ta je určena následujícím vzorcem $\sigma^2 = \frac{1}{k}\sum_1^k(\overline{w} - w_i)^2$, kde k je počet částí, $\overline{w}$ je průměrná váha propojení částí a w_i je váha propojení části. Tyto váhy jsou definovány jako váha propojení již známého neuronu s částmi quadtree (či hyperoctree). Pokud tato míra informací přesahuje předem stanovenou hranici, oblast se opět rozdělí až do maximální stanovené hloubky. Nejhlubší vygenerované oblasti jsou potom převedeny na neurony skryté vrstvy v hypercubu. S tímto modelem sítě v hypercubu se následně pracuje stejně jako u HyperNEATu viz 6.3.



Obrázek 11 Ukázka topologií vyvinutých pomocí ES-HyperNEAT [22]

8 CNAOS-NEAT

CNAOS-NEAT je rozšíření navržené v rámci této práce NEATu, které lze použít s NEATem, HyperNEATem a ES-HyperNEATem. Toto rozšíření se snaží řešit problém s deceptivitou domén v problémech posilovaného učení a zároveň se snaží hledaná řešení směrovat. Toho dociluje kombinací novelty a objective searche v kombinaci se systémem ras NEATu.

8.1 Vyhodnocování domén posilovaného učení

Častým problémem posilovaného učení je, jak domény vyhodnocovat. Vyhodnocování, které se zdá být přímým odrazem pokroku nemusí být nejlepší volba. U spousty problémů se stává, že cesta postupného přibližování k řešení nemusí být závislá na tom, jak dobře si agent v doméně vede.

8.1.1 Objective based search

Klasickým přístupem k tomuto vyhodnocování je tzv. objective based search, který vyhodnocuje, jak moc se agent blíží cíli domény. Příkladem takového vyhodnocování pro agenta v bludišti by byla vzdálenost k cíli. Čím blíže je agent cíli, tím větší fitness skóre by měl. Ovšem je možné, že se agent dostane velmi blízko cíli, ale přesto je velmi daleko od cesty, která by ho k cíli dovedla. Proto byla navržena alternativa k tomuto způsobu vyhodnocování a to tzv. novelty search.

8.1.2 Novelty search

Novelty search [6] byl navržen, jako alternativa ke klasickému objective based search. Tento přístup neodměňuje agenty na základě toho, jak moc se přiblížili k cíli domény, ale podle toho, jak moc je řešení odlišné od již nalezených. Díky tomu je tento přístup lépe přizpůsobený k odměňování agentů, kteří ukazují nové chování, a tedy prozkoumávají doménu a tím také prostor řešení. Zároveň díky této snaze prozkoumávat prostor řešení se snižuje riziko, že se vývoj zastaví v lokálním optimu, ze kterého nemůže pokračovat.

Tento přístup by se na první pohled mohl zdát, jako náhodné vyhledávání. Ovšem u většiny domén posilovaného učení je množství chování s jednoduchou strukturou omezený.

Například opět u navigace robota v bludišti, pokud se bude řídit jednoduchým chování, je jenom velmi omezený počet konečných stavů, které mohou nastat. Může se pouze zastavit o okolní zdi, nebo se pohybovat v kruhu. Díky novelty search se po prozkoumání těchto banálních řešení rychle penalizují, tj. snižuje se jejich fitness skóre. To algoritmus nutí vytvářet nová chování, která je možné vytvořit pouze tím, že vytvoří složitější chování.

Díky tomu je prostor řešení nalezených pomocí novelty search mnohem víc rozprostřený a pokrývá větší škálu chování. Jedním z hlavních problémů tohoto přístupu je možnost otevřených domén, tj. domény, které mají téměř neomezený počet řešení.

Tento přístup je v práci implementován vytvořením seznamu vektorů, které definují již nalezené chování. Tyto vektory mohou být například konečné pozice robotů v bludišti, nebo jiné informace, které reprezentují chování agenta v dané doméně. Novelty skóre je určeno jako průměr euklidovské vzdálenosti ke *k* nejbližších bodů, kde *k* je určeno uživatelem.

8.2 Charakteristika

Tato práce navrhuje vytvoření nového přístupu, který je kombinací obou přístupů k vyhodnocování domén, který byl pojmenován CNAOS-NEAT (combination of novelty and objective search – NEAT). Tento algoritmus se snaží řešit problémy otevřených domén a dále zlepšit vyhodnocování.

Hlavní myšlenkou tohoto algoritmu je, že řešení by algoritmus neměl hledat pouze tak, aby byla inovativní, ale také by se měla podporovat řešení, která směřují k cíli. Tím se algoritmus snaží předejít situacím, ve kterých se vývoj soustředí do prostorou řešení, která nevedou k cíli vývoje a zároveň zachovat snahu prozkoumávat nová řešení.

To algoritmus řeší vyhodnocováním ras pomocí součinu obou skór jak objective, tak novelty. Při navrhování algoritmu je tedy možné určit váhu obou metrik a tím algoritmus uzpůsobovat pro větší exploration nebo exploitation. Myšlenkou tohoto počinu je, že prostor kroků k řešení je korelován s jak objective, tak novelty skórem.

V rámci ras používá CNAOS-NEAT objective skóre, které určuje, jak velká je šance, že daný genom bude mít v následující generaci potomka. Tímto způsobem je rasa vedena k upřednostňování řešení, které se ukázala jako slibnější.

9 EXPERIMENTY

V práci bylo navrženo množství experimentů, které slouží k prokázaní určitých vlastností testovaných algoritmů a zároveň ukazují jejich potenciál. V této části budou dané experimenty popsány a bude vysvětleno, které algoritmy jsou pro ně vhodné.

9.1 Aproximace funkce

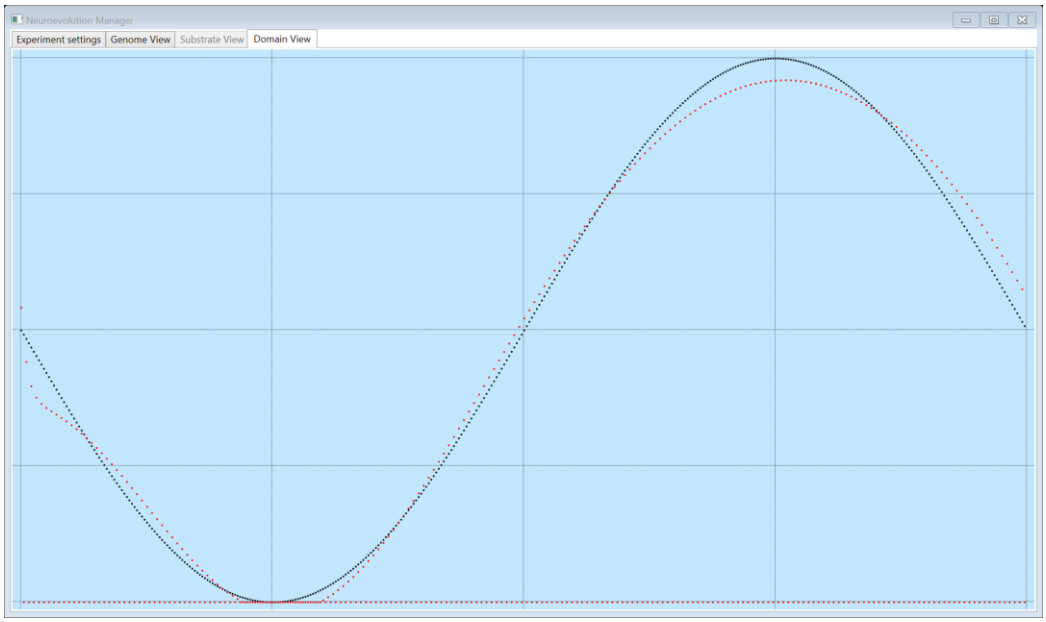
Tento experiment byl navržen tak, aby dokázal základní schopnost algoritmu se učit. Jeho cílem je vytvořit neurální síť, která je schopná zreprodukovat průběh funkce sinus.

Tento úkol by se mohl zdát jako úlohou supervised learningu, ale při zobecnění problému na nalezení modelu, který co nejpřesněji aproximuje hledaný průběh, zjistíme, že řešení problému vlastně není známé (průběh může zobrazovat jakoukoliv funkci, kterou je mu zadána) a proto se jedná o problém posilovaného učení.

Hlavní výhodou tohoto experimentu je jeho velká jednoduchost, a tedy i rychlost vyhodnocení. To umožňuje tento experiment použít pro testování funkčnosti nových algoritmů.

Vstupem tohoto problému jsou *x* souřadnice hledaného bodu, ke kterým má agent přiřadit *y* souřadnici. Fitness hodnocení tohoto experimentu je druhá mocnina sumy odchylek všech bodů od hledaných. Novelty hodnocení (v tomto případě přítomné jenom z důvodu testování) je vektor popisující nalezený průběh (posloupnost *y* souřadnic bodů).

Ideálním algoritmem pro tento experiment je NEAT používající fitness skóre.



Obrázek 12 Vizualizace experimentu aproximace funkce

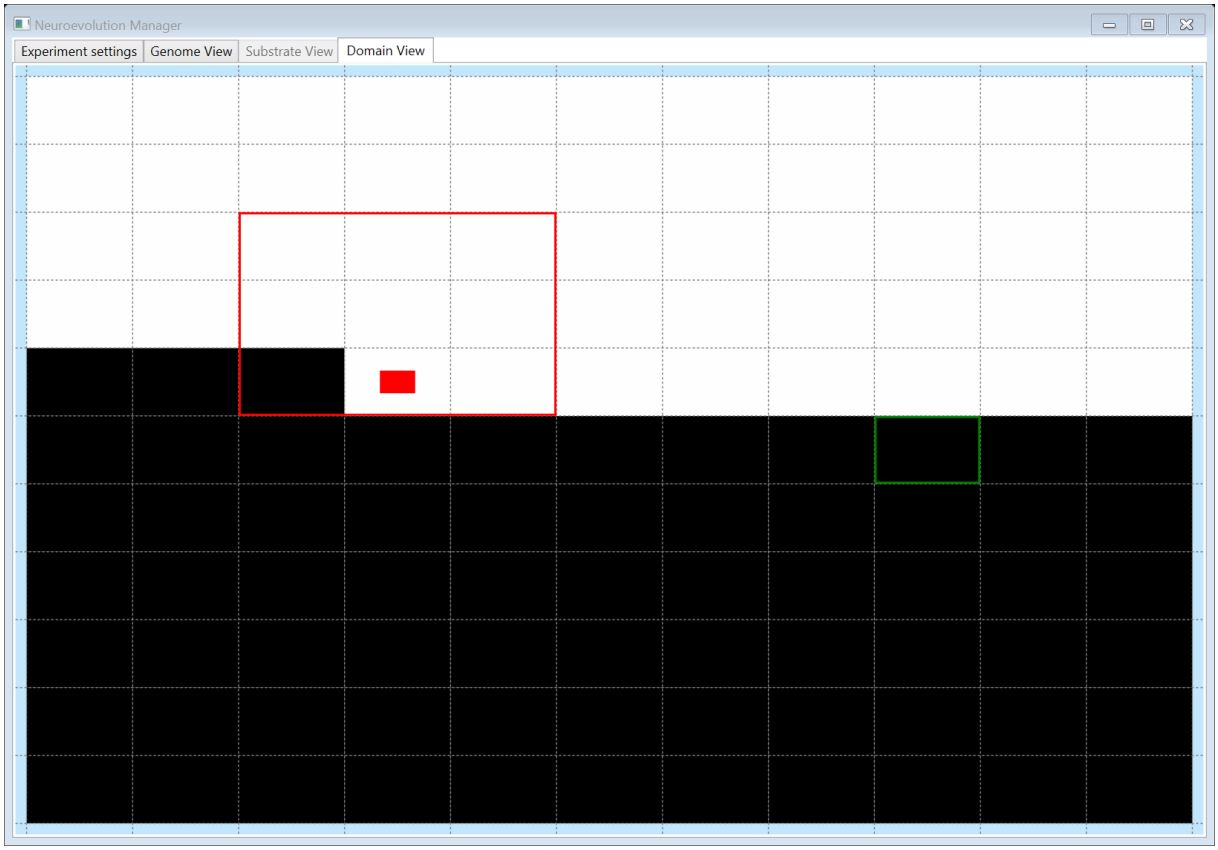
9.2 Boxes visual discrimination

Tento experiment byl navržen tak, aby ukázal výhodu HyperNEATu oproti klasickému NEATu na úlohách v prostoru. Cílem tohoto experimentu je, aby agent určil střed většího ze dvou čtverců v mřížce.

Tento problém spadá do kategorie supervised learningu, ale byl zařazen do práce, protože umožňuje jednoduché otestování HyperNEAT (a jeho rozšíření) na jednoduchém problému.

Tento experiment, stejně jako předchozí, má velkou výhodu, svoji rychlost, která mu umožňuje jednoduché testování algoritmů.

Vstupem tohoto problému je dvourozměrné pole, které nabývá hodnot 1 nebo 0. V tomto poli jsou přítomny dva čtverce, jeden s velikostí 1x1 a druhý 3x3. Cílem algoritmu je rozpoznat, kde se nalézá střed většího ze čtverců a nenechat se zmást menším. Výstupem agenta je pole o stejné velikosti, které určuje jeho přesvědčení nad tím, že se střed většího čtverce nalézá na tom místě. Fitness hodnocení tohoto experimentu je suma rozdílů výstupu agenta a správným řešením, tj. pole, ve kterém jsou všechny výstupy kromě toho, ve kterém se nalézá střed velkého čtverce na 0. Novelty hodnocení z důvodu charakteru experimentu nebylo implementováno.



Obrázek 13 Vizualizace experimentu boxes visual discrimination

9.3 Shooter

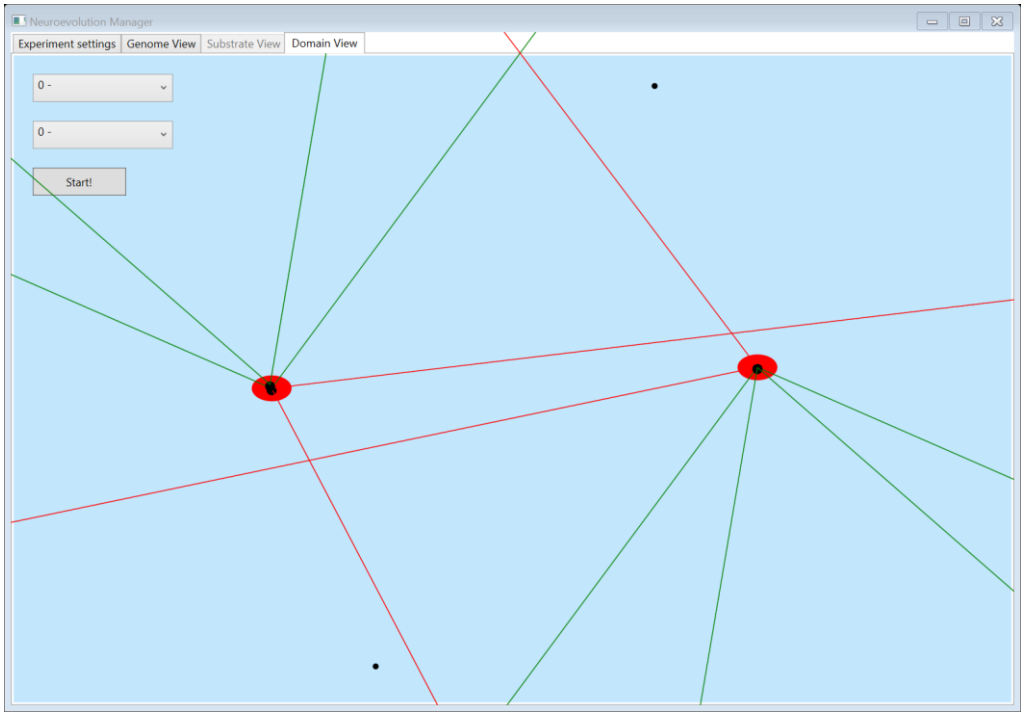
Shooter experiment byl navržen tak, aby vyzkoušel schopnost algoritmu ve více interaktivním prostředí, které je typičtější pro problémy posilovaného učení. Zároveň se jedná o experiment využívající kompetitivní evoluci.

Cílem tohoto experimentu je vytvořit agenta, který ovládá robota a je schopen sestřelit nepřítele a zároveň se vyhýbat jeho střelám. Robot, kterého agent ovládá, je vybaven třemi „oky“. Tato oka jsou rozprostřena v kruhu okolo středu robota. Agent může ovládat jejich polární souřadnice vzhledem k robotovi, jejich natočení, jejich FOV (field of view – zorné pole) úhel a zároveň z nich střílet. Tato oka jsou též vybavena senzory, které dokáží určit, zda je ve FOV přítomna nepřátelská střela nebo robot a jak daleko od robota jsou. Robot samotný má možnost se pohybovat a otáčet okolo své osy.

Vstupem pro tento experiment je, pro každé oko, vzdálenost k nepřátelské střele, robotovi, úhel, který svírá relativně k robotovi a doba, kterou musí agent vyčkat, než bude moci zase vystřelit. Dále má agent informaci o své i nepřítelově pozici, svém absolutním úhlu v rámci domény a úhlu, který svírá s nepřátelským agentem.

Výstupem agenta je rychlost, kterou se pohybuje daným směrem, změna úhlu a nastavení všech parametrů ok viz výše.

Fitness skóre agenta je určeno podle toho, jak rychle je schopen zasáhnout svého nepřítele, s penalizací za to, když ho zasáhne nepřítel. Novelty skóre je poté určeno podle stavu robota při každém zásahu (úhel, pozice, FOV úhel, …).



Obrázek 14 Vizualizace experimentu shooter

9.3.1 Kompetitivní evoluce

Kompetitivní evoluce je taková evoluce, ve které se nevyvíjí agent určitým stálým hodnocením (např. vzdálenost k cíli) ale relativně k evoluci samotné. To znamená, že agent se vyhodnocuje hraním proti jiným vyvinutým agentům. Tento přístup umožňuje překonat jeden z problémů, který podobné domény potkává, a to neschopnost dostat se přes jistou hranici.

Oblíbený přístup k vývoji podobných domén je, naprogramování statické inteligence, která slouží jako protivník vyvíjeného agenta. To umožňuje měřit schopnosti agenta na stálém chování a umožňuje mu se zlepšovat. Hlavním problémem tohoto přístupu ovšem je, že výsledný agent nemůže překonat stálého naprogramovaného agenta. Jelikož takový agent představuje stálé chování, potřebuje se agent naučit pouze to, jak vyhrát proti jeho taktice. Pokud tak učiní, ztratí důvod vymýšlet jinou taktiku, a může ji pouze mírně zlepšovat pro maximalizaci skóre.

Tomuto problému se navrhnutý přístup brání tak, že agent vždy hraje proti verzi ze starší generace. Algoritmus používá vždy nejúspěšnější genomy v každé rase z předchozí generace. Tímto způsobem je možné zajistit, aby agent soupeřil vždy s protivníkem na správné úrovni a takovým, který používá škálu taktik.

Jedním z problémů takového přístupu je, jak vyhodnotit úspěch experimentu. Samotné fitness skóre není objektivním měřítkem pro inteligenci agenta, jelikož je určeno relativně k agentům starší generace. Pro řešení tohoto problému lze například použít přístup stálého předem naprogramovaného agenta. Ten může sloužit jako orientační měřítko úspěchu vývoje.

9.3.2 Problém s vyhodnocováním experimentu

Jedním z problémů, který bylo u vyhodnocování této domény nutné vyřešit, byla nutnost balancovat odměnu a trest. Při vyhodnocování agentů, pouze na základě odměn za zásahy, agenti použili taktiku, která jim umožňovala co nejjednodušší zásah, bez ohledu na vlastní bezpečnost. Tato taktika spočívala v tom, že se agenti přiblížili co nejblíže k sobě a stříleli. Tato taktika byla z pohledu agentů logická, protože se v průměru maximalizovalo vyhodnocení agentů – minimalizovala se doba mezi úspěšnými zásahy. Podobný problém nastal i při vyhodnocování pouze na základě trestů. Agenti se opět pouze snažili vyhýbat se střelám nepřítele. To ukazuje důležitost balancování odměn a trestů v doménách, ve kterých agent soupeří s jiným.

9.4 Walker

Posledním implementovaným experimentem je tzv. walker experiment. V tomto experimentu agent řídí robota, který se snaží chodit dvourozměrnou doménou. Cílem robota je ujít co nejdelší vzdálenost co nejrychleji. Robot je sestaven ze dvou nohou, které může ovládat v pase a v koleně.

Vstupem agenta v tomto experimentu je úhel v torzu, rychlost torza, *y* pozice torza a úhel v pasu a kolenou obou nohou. Výstupem agenta je točivý moment v koleni a v pase pro obě nohy.

Tento experiment demonstruje schopnost algoritmu vyvíjet i komplexní chování a zároveň ukázal příklad domény, ve které CNAOS-NEAT poráží objective NEAT (v průměru o 550%) a novelty NEAT (v průměru o 56%).

Obrázek 15 Vizualizace experimentu walker

10 IMPLEMENTACE

V této kategorii bude popsán způsob implementace programu, který je výstupem praktické části práce. Budou zde popsány použité frameworky, struktura celé aplikace a GUI.

10.1 Použité frameworky a jazyk

Celá praktická část práce je implementována v jazyce C# a využívá frameworků .NET a .NET standart. To umožňuje programu běžet jak na Windowsovských zařízeních, tak na zařízeních používající systémy na bázi UNIX. To byla nutnost pro otestování programu na superpočítačích, které téměř výhradně používají operační systém Linux. Samotný program je optimalizován pro vysokou míru paralelizace a značná většina programu je paralelizována, pro libovolný počet výpočetních vláken.

Program je rozdělen do dvou částí, v jedné se nachází logika aplikace (.NET standart – UNIX) a v druhé GUI (.NET – Windows). Celá aplikace je psána bez použití externích knihoven, s výjimkou rozšíření Box2DX [7] (fyzikální engine pro simulace), OpenTK [8] (nízko úrovňová knihovna pro OpenGL), Vector-2D [9] (pomocná knihovna pro vektorové operace), LiveCharts [10] (vizualizace grafů ve WPF) a WFTools3D [11] (pomocná knihovna pro 3D simulace ve WPF).

10.2 Struktura aplikace

Hlavními cíli vytvořené aplikace byla možnost sledovat průběh evoluce v reálném čase, jednoduché rozhraní pro přidávání nových experimentů a jednoduchá rozšiřitelnost.

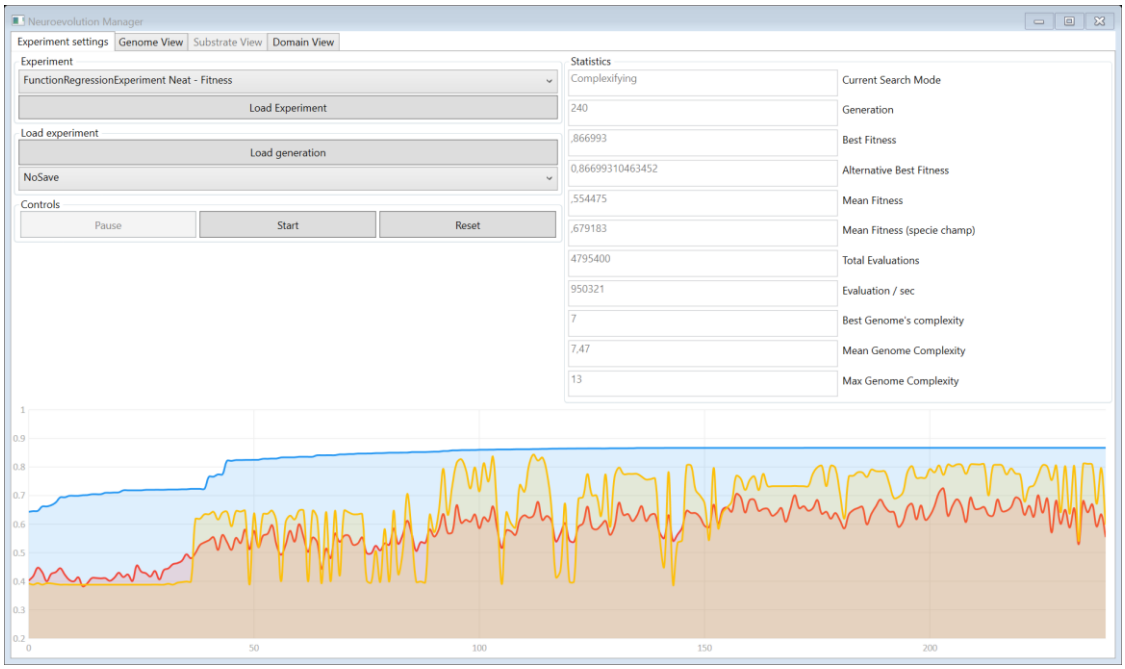
Těchto cílů bylo dosaženo navrhnutím odpovídajícího GUI (viz níže) a zvolení patřičné struktury programu, které tyto cíle umožňuje (rozdělení řešení na 8 projektů, struktura tříd, …).

10.3 GUI

Celá aplikace má dvě rozhraní, jedno pro Linux a jedno pro Windows. Linuxové je pouze CLI, které zobrazuje údaje o vývoji, proto bude v této části kladen důraz hlavně na Windows GUI, napsané pomocí WPF.

10.3.1 Hlavní stránka

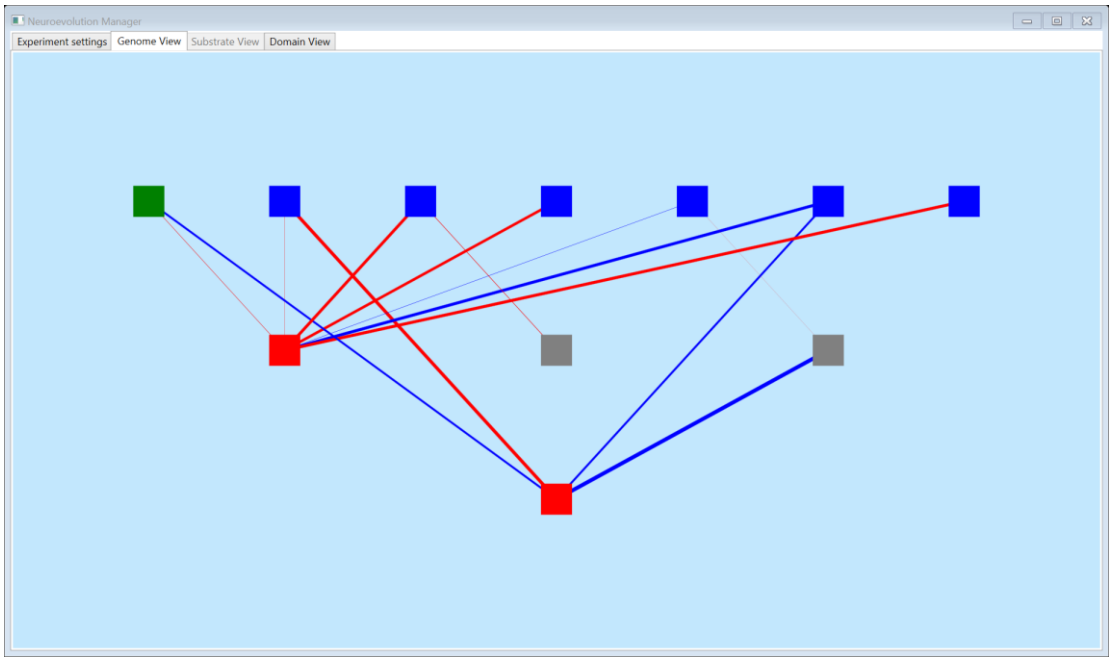
Hlavní stránka poskytuje rozhraní pro ovládání celého programu a sledování vývoje. Umožňuje nahrávat experimenty, nahrávat uložené populace genomů a řídit průběh simulace (zastavovat, spouštět). Zobrazuje statistiky současné generace vývoje a také graf, který ukazuje maximum, průměr a medián fitness skóre vývoje.



Obrázek 16 Ukázka hlavní stránky aplikace

10.3.2 Zobrazení genomu

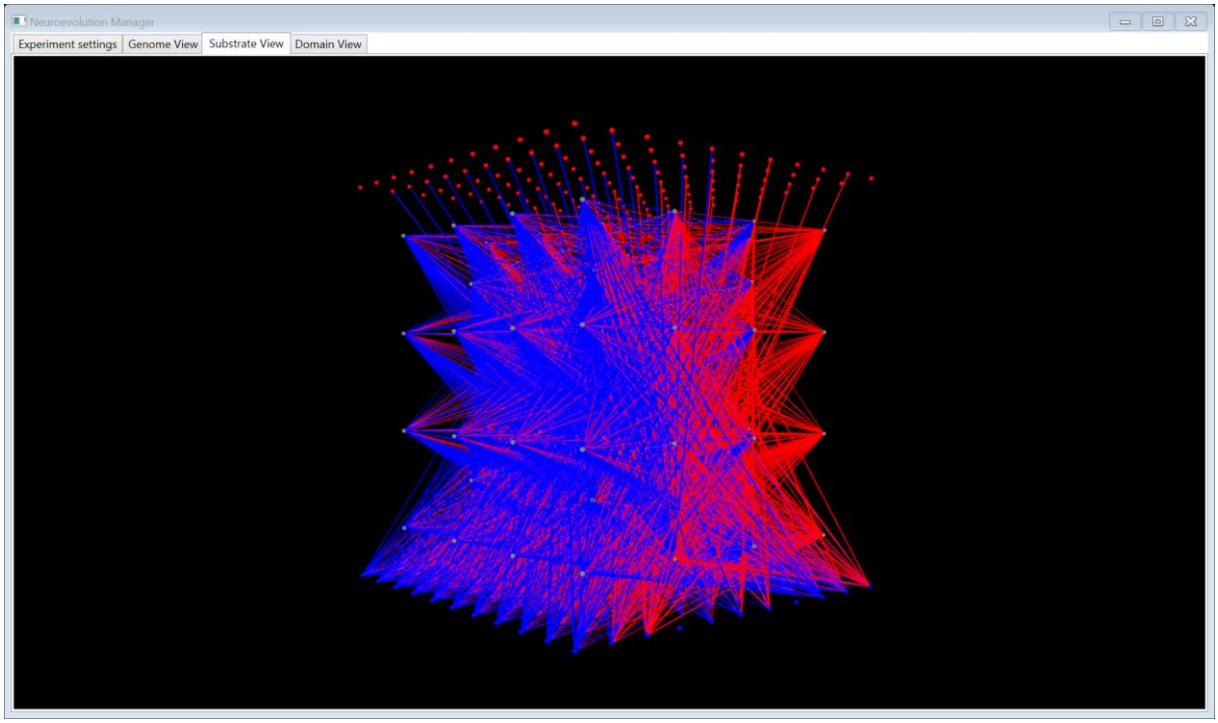
Tato stránka umožňuje zobrazovat strukturu vyvíjené neurální sítě. Zobrazuje vždy schéma neurální sítě, která má v aktuální generaci nejvyšší fitness skóre. Modrými čtverci jsou znázorněny vstupní neurony, červenými čtverci výstupní, šedé čtverce označují neurony skryté vrstvy a zelený čtverec představuje bias. Váhy spojující neurony jsou obarveny podle znaménka propojení – modrá pro kladnou vazbu a červená pro zápornou. Šířka čáry znázorňuje váhu propojení.



Obrázek 17 Ukázka vizualice genomu v aplikaci

10.3.3 Zobrazení substrátu (hypercube)

Tato stránka zobrazuje grafickou reprezentaci umístění neuronů v hypercube. Podporuje zobrazení až do tří dimenzí a používá stejné barevné schéma jako stránka s genomy, viz výše.



Obrázek 18 Ukázka vizualizace vyvinutého substrátu

10.3.4 Zobrazení domény

Tato stránka umožňuje interaktivní zobrazení domény. Podporuje její zobrazení (většinou s genomem s nejvyšším fitness skóre) a v některých případech i interakci s doménou (například možnost hýbat s robotem u experimentu Walker) viz obrázky 12–14.

11 ZÁVĚR

V práci byly popsány teoretické základy technik umělé inteligence se zaměřením na genetické algoritmy – konkrétně algoritmus NEAT a jeho rozšíření. Dále byla prostudována možnost aplikování algoritmu na problémy posilovaného učení a bylo vytvořeno několik experimentů, které ukazují schopnost programu učit se a nalézat řešení pro zadané problémy.

Součástí práce bylo vytvoření vlastního rozšíření pro algoritmus NEAT, které v některých doménách ukázalo potenciál být lepší, než alternativní řešení (v rámci NEATu). Toto rozšíření, ačkoliv by potřebovalo vyzkoušet na větším množství experimentů, dokázalo, že na poli umělé inteligence je pořád velký prostor pro inovace a tento algoritmus je jenom ukázkou jednoho ze směrů, kterým bychom se mohli vydat.

Dalším rozšířením práce by mohlo být přidání dalších algoritmů, nebo optimalizace ES-HyperNEATu [12]. Ale hlavním problémem tohoto přístupu je neschopnost vyjádřit dostatečně složité struktury pro velmi komplexní úkoly. NEAT a jeho rozšíření se v posledních letech ukázaly jako dobré algoritmy pro problémy především v posilovaném učení ale pouze u jednoduchých úkolů. Tento algoritmus nelze úspěšně aplikovat na problémy takového rozsahu jako například hraní Dota 2. Tento problém je nutno vyřešit, aby se mohl přístup s tak obrovským potenciálem jako NEAT dostat do popředí pole umělé inteligence.

12 POUŽITÁ LITERATURA

1. DeepMind. AlphaGO. *AlphaGO | DeepMind.* [Online] DeepMind. [Citace: 11. 3 2018.] https://deepmind.com.

2. OpenAI. Dota 2. *OpenAI Blog.* [Online] OpenAI, 9. 11 2017. [Citace: 11. 3 2018.] https://blog.openai.com/dota-2/.

3. Kenneth O. Stanley, Risto Miikkulainen. The University of Texas at Austin. [Online] 2002. [Citace: 11. 3 2018.] utexas.edu.

4. Kenneth O. Stanley, David D'Ambrosio, Jason Gauci. Brigha, Young University. [Online] 2009. [Citace: 11. 3 2018.] http://axon.cs.byu.edu/~dan/778/papers/NeuroEvolution/stanley3**.pdf.

5. Sebastian Risi, Kenneth O. Stanley. University of Central Florida. [Online] 2012. [Citace: 11. 3 2018.] http://eplex.cs.ucf.edu/papers/risi_alife12.pdf.

6. Lehman, Joel. Joel Lehman. [Online] 2007. [Citace: 12. 3 2018.] http://joellehman.com/lehman-dissertation.pdf.

7. lytico. box2Dx. [Online] 15. 10 2008. [Citace: 12. 3 2018.] https://code.google.com/archive/p/box2dx/.

8. varon. OpenTK. [Online] 2018. [Citace: 12. 3 2018.] https://github.com/opentk/opentk.

9. styxtwo. Vector-2D. [Online] 24. 1 2016. [Citace: 12. 3 2018.] https://github.com/styxtwo/Vector-2D.

10. Rodríguez, Alberto. LivCharts. [Online] 1. 3 2018. [Citace: 12. 3 2018.] https://lvcharts.net/.

11. Foerster, Wolfgang. Code project - WFTools3D. [Online] 25. 3 2016. [Citace: 12. 3 2018.] https://www.codeproject.com/Articles/1087090/WFTools-D.

12. Sebastian Risi, Kenneth O. Stanley. University of Central Florida. [Online] 2011. [Citace: 12. 3 2018.] http://eplex.cs.ucf.edu/papers/risi_gecco11.pdf.

13. NK, Mubaris. Mubaris' Blog. [Online] 1. 11 2017. [Citace: 12. 3 2018.] https://mubaris.com/2017/10/01/kmeans-clustering-in-python/.

14. Upadhyay, Rooparn. YOU CANalytics. [Online] 2017. [Citace: 12. 3 2018.] http://ucanalytics.com/blogs/gradient-descent-logistic-regression-simplified-step-step-visual-guide/.

15. Farhad. WebinDream. [Online] 19. 12 2017. [Citace: 12. 3 2018.] http://webindream.com/reinforcement-learning/.

16. Rosebrock, Adrian. pyimagesearch. [Online] 26. 10 2016. [Citace: 12. 3 2018.] https://www.pyimagesearch.com/2016/09/26/a-simple-neural-network-with-python-and-keras/.

17. Deshpande, Mohit. Pythonmachinelearning. [Online] 12. 9 2017. [Citace: 12. 3 2018.] https://pythonmachinelearning.pro/perceptrons-the-first-neural-networks/.

18. Roell, Jason. Towards Data Science. [Online] 26. 6 2017. [Citace: 12. 3 2018.] https://towardsdatascience.com/understanding-recurrent-neural-networks-the-prefered-neural-network-for-time-series-data-7d856c21b759.

19. Hornby, Gregory S., a další. Wikipedia. [Online] 9 2006. [Citace: 12. 3 2018.] https://en.wikipedia.org/wiki/Evolved_antenna.

20. Soroosh Sohangir, Shahram Rahimi, Bidyut Gupta. Scientific Research. [Online] 23. 5 2014. [Citace: 12. 3 2018.] http://file.scirp.org/Html/3-9301889_46575.htm.

21. Clune, Jeff. Research Gate. [Online] [Citace: 12. 3 2018.] https://www.researchgate.net/publication/291053132_Evolving_Gaits_for_Physical_Robots_with_the_HyperNEAT_Generative_Encoding_The_Benefits_of_Simulation/figures?lo=1.

22. Stanley, Kenneth O. EPLEX. [Online] [Citace: 12. 3 2018.] http://eplex.cs.ucf.edu/hyperNEATpage/.

23. Wikipedia. [Online] 5 2015. [Citace: 12. 3 2018.] https://en.wikipedia.org/w/index.php?title=Quadtree.

13 SEZNAM OBRÁZKŮ A TABULEK

14 PŘÍLOHA 1: FINÁLNÍ APLIKACE