Универзитет у Београду
Електротехнички факултет

ПРИНЦИПИ НЕУРО-ЕВОЛУТИВНИХ СИСТЕМА У УЧЕЊУ СА ПОДРШКОМ

Мастер рад

Ментор:
Доц. др Дражен Драшковић

Кандидат:
Владимир Сивчев 3172/2019

Београд, новембар 2020.

Садржај

1. Увод

Овај документ представља мастер рад на тему „Принципи неуро-еволутивних система у учењу са подршком“. Као и други системи засновани на техникама вештачких интелигенција, где исте за циљ имају да унесу одређено понашање у сам систем које би се могло окарактерисати интелигентним, тако је и циљ овог рада да се испита одређени подскуп алгоритама, модела и принципа вештачке интелигенције како би се мериле перформансе истих техника и донели одговарајући закључци.

Као концепт, вештачка интелигенција (енгл. *Artificial Intelligence* [1]) почиње да се развија средином 1950-их година. Термин је сковао научник *John McCarthy*, један од оснивача ове дисциплине. У питању су такође године где и само рачунарство креће са развојем, где истраживачи, научници и инжењери из ових области наилазе на проблеме где се конвенционално програмирање и начин размишљања за решавање проблема не може применити, односно, тиме добијена решења су за дате проблеме не тако ретко и по више критеријума неприхватљива. У оваквим ситуацијама, решење које би било прихватљиво се веома често, ако не и увек, састоји од карактеристичног понашања које указује на одређени ниво интелигенције, попут тога када би човек сам, уколико је у могућности, исти проблем решавао. Наведено представља главни појам истраживања ове области.

У литератури, вештачка интелигенција (ВИ) се може категорисати на више начина, кроз више типова и нивоа. У најопштијем смислу, најчешће се ВИ дели у две групе, једна заснована на функционалности ВИ, а друга заснована на њеној способности. По функционалности, разликују се 4 категорије ВИ:

- Реактивне машине (енгл. *Reactive Machines*) – представља најједноставнији тип ВИ; овакви ВИ системи не чувају меморију и претходна искуства за будуће сценарије и акције, већ се искључиво фокусирају на тренутну ситуацију и реагују на начин на који сматрају да је најбољи; *AlphaGo* [2] развијен од стране *DeepMind* [3] је пример оваквог система;
- Ограничена меморија (енгл. *Limited Memory*) – представљају машине које чувају претходна искуства или податке скупљене у неком краћем, ограниченом периоду; самовозећи аутомобили представљају најбољи пример оваквих система, с обзиром да у склопу функционалности чувају информације о недавним брзинама оближњих аутомобила, њихових растојања, ограничења брзине и друге информације;
- Теорија ума (енгл. *Theory of Mind*) – ВИ овог типа треба да разумеју људске емоције, човека, веровања, културе, као и да у друштвеном смислу на исти начин интерагују са човеком онако како би они сами међусобно; овакви ВИ системи се још увек не развијају, али истраживачи улажу много труда у побољшању оваквих система;
- Самосвест (енгл. *Self-Awareness*) – будућност ВИ; овакве машине би биле веома интелигентне, имале сопствену свест, сентименте и осећања, као и друге карактеристике које и сам човек има, а машина (до тог тренутка) не; самосвесни ВИ системи још увек не постоје у реалности и за сада су само хипотетички концепт.

По способности, ВИ се може поделити у 3 категорије:

- Слаб ВИ (енгл. *Narrow, Weak AI*) – овакав тип ВИ је способан да извршава специфично намењени задатак са неким видом интелигенције; овакви системи немају могућност да врше било какав други задатак ван њихове намене, области и граница, јер су обучавани искључиво за дати задатак (отуда и име – слаб); уколико зађу ван својих граница, греше на непредвидиве начине; до сада, једини постојећи ВИ у свету јесте ВИ ове категорије;
- Јак ВИ (енгл. *Strong, General AI*) – уопштени тип интелигенције који би био у стању да извршава било какав интелектуални задатак са ефикасношћу попут људске; идеја је да се направи неки систем који би био паметнији, а такође и способан да сам по себи размишља као човек; до сада не постоји никакав систем који би могао да се сврста у ову категорију, а такви системи би по природи првенствено морали да прођу *Turing*-ов тест [4];
- Супериорни ВИ (енгл. *Super, Superior AI*) – овакве машине превазилазе интелигенцију човека и њега уопште, и могу да изврше било какав задатак боље него човек у сваком смислу; овакав ВИ би био производ јаког ВИ, а исти је само хипотетички концепт.

У току развоја саме области, настајали су разни модели, алгоритми и приступи како би се остварили системи са одређеним нивоом интелигенције. Циљ овог рада је да представи и тестира само један подскуп широког спектра техника које припадају овој дисциплини, а у питању је спој два концепта, а то су **неуралне мреже** [5] и **генетички алгоритам** [6].

Сам концепт неуралне мреже (НМ) је познат већ око 70 година. Многи ВИ системи који постоје су засновани на њима, или у неком свом делу садрже неки вид неуралне мреже. Инспирисан је од стране биолошког нервног система, а представља поједностављен и имитиран математички модел који се показује као веома ефективан. Важна способност коју НМ поседују је да имају могућност такозваног машинског учења (енгл. *Machine Learning* [7]). Системи, модели и алгоритми са овом могућношћу заправо имају способност да уче и да се унапређују из искуства, а да истовремено не буду експлицитно за исто програмирани. Чест појам који се користи уместо учења јесте обучавање или тренирање модела, односно у овом случају, обучавање НМ. У овом раду, од интереса ће бити метода која се заснива на Дарвиновој теорији еволуције, а у литератури је познатија под именом генетички алгоритам.

Генетички алгоритам (ГА) је само један у низу од еволутивних алгоритама који се баве оптимизацијом функција. Инспирисан је од стране еволутивних процеса који се одвијају у природи, а који као резултат манифестују промене које углавном указују на одређена побољшања и унапређења. Поред еволутивних алгоритама, постоје и други алгоритми чија је намена иста, као што су градијентни метод, симулирано каљење, симплекс алгоритам, динамичко програмирање и други, међутим, сам концепт коришћења ГА у обучавању НМ се показује као не само интересантан, већ и веома ефикасан концепт, а у литератури познатији под називом „неуро-еволуција“ [8].

Неуро-еволуција (НЕ) се појављује у више облика. Наиме, као што је већ речено, сам ГА који је одговоран за тренирање НМ је само један у низу од могућих алгоритама који се могу изабрати у исту сврху. Чак и у тренутку када се он одабере, ГА поседује разнолике параметре и могућности измена да и сам представља веома пространо појам, употребљив на много начин. Главна мотивација и инспирација која стоји иза ове идеје јесте принцип еволуције биолошког нервног система, где НЕ покушава да имитира и примењује апстракције природне еволуције за конструкцију апстракција биолошких неуралних мрежа. Циљ дакле јесте да се еволуирају НМ које поседују неку врсту интелигентног понашања.

Оваква метода машинског учења (МУ) показује ефикасан приступ у решавању проблема који се баве учењем са подршком (енгл. *Reinforcement Learning* [9]). У оваквом учењу, дефинисано је постојање света, као и лица, често званог агента, који врши одговарајуће акције у свету и тиме пролази кроз одговарајућа стања. По свакој акцији, агент добија одговарајућу награду (енгл. *reward*), која може бити позитивна, негативна или неутрална. Циљ агента је да максимизује награду, односно да се што боље понаша у свету у ком актује. Као што је сам концепт НМ, ГА, па и НЕ инспирисан од стране процеса у природи и човеку, тако се и учење са подршком (УП) може повезати са психологијом човека. У оквиру научне области психологије, постоји објашњење које изјашњава да ће човек подржати и појачати (енгл. *reinforce*) акције за које је позитивно награђен, а одбацити и клањати се акција за које је негативно награђен (где је појам награде у овом смислу апстрактнији него иначе). Ово такође не важи само за човека, већ и за животиње уопште, а сам концепт се и користи за тренирање животиња.

У поглављу 2 овог документа биће дата анализа НМ, ГА и НЕ у УП, као и поменута друга постојећа решења и друге методе које се могу у сличну или исту сврху користити. У поглављу 3 биће описани подсистеми и концепти који су неопходни за реализацију анализираних појмова из поглавља 2. У поглављу 4 биће описан принцип обучавања НМ помоћу ГА, односно детаљно ће бити објашњена техника НЕ. У поглављу 5 биће презентовани проблеми и симулације који су УП-типа, на којима се тестира метода која се радом истражује, као и дати резултати који су кроз исте симулације постигнути. У поглављу 6 биће објашњене напредније технике које се могу користити у овој области, а такође ће бити представљени резултати истих. У поглављу 7 биће дат закључак до сада направљене инфраструктуре за НЕ и биће приказане могућности за надоградњу и унапређење.

2. АНАЛИЗА ОСНОВНИХ КОНЦЕПАТА

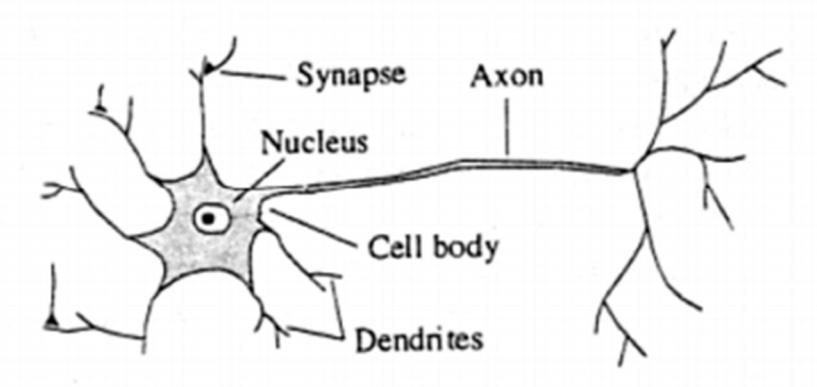
Као што је већ речено, циљ овог рада је да истражи и представи технике неуро-еволуције у учењу са подршком. Пре него што би се сама техника објаснила, неопходно је анализирати њене делове, а поготово је потребно да се ти делови добро дефинишу. Стога, у овом поглављу, првенствено ће бити детаљно анализиране неуралне мреже и јасно дефинисане операције које се у оквиру њих одвијају, а затим ће се анализирати генетички алгоритам, где ће такође бити дефинисане операције које се у истом догађају. Затим ће се ова два појма спојити у анализи саме неуро-еволуције, где ће бити представљено како се овај концепт уграђује у системе где се одвија учење са подршком.

2.1. АНАЛИЗА НЕУРАЛНИХ МРЕЖА

Већ је поменуто да НМ представљају математички модел инспирисан биолошким нервним системом. Наиме, у свакој мрежи постоје неурони који имају могућност процесирања информација које прихватају, као и слања излазних информација добијених истим процесирањем даље, ка дубљим (енгл. *deep*) неуронима који имају сличне улоге. Самим тим што неурони међусобно размењују информације, говори да међу њима мора постојати неки вид веза (енгл. *connection*). Након повезивања и мањег броја неурона, често именовани „процесирајуће јединице“ или „процесирајући елементи“, добијамо мрежу неурона, одакле је и име сковано.

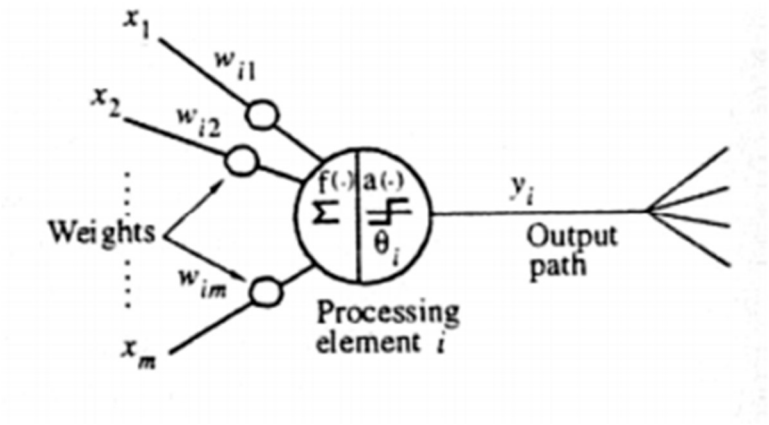
2.1.1. ПРИНЦИП И НАЧИН РАДА ПОЈЕДИНАЧНОГ НЕУРОНА

На слици 1 је представљен биолошки неурон са својим основним компонентама. Неурон представља јединицу која је способна да обрађује информације. Са слике 1, то представља *Cell body*, тело које у себи садржи једро (енгл. *Nucleus*). Међутим, пре него што ће процесирати информације, потребно је да их прими. Као што је поменуто, неопходно је да постоје везе одакле ће саме информације, односно сигнале тело примити, а то се дешава преко синапси и дендрија које представљају интерконекцију неурона у биолошком нервном систему. Након што информације достигну до тела и обраде се, излазни сигнал из датог неурона се шаље до наредних неурона, помоћу дела налик на танку нит, која се назива аксон.



Слика 1 – Биолошки неурон

Од интереса су такође сензорски неурони, чија је улога да одговарајуће стимулансе које примају преко својих рецептора конвертују у акциони потенцијал, који представља информацију у виду сигнала који се пропагира даље, до наредних неурона, који су претходно описани, све дубље. Ова свеукупна обрада као резултат може имати више ефеката. Један пример би био да се излаз одговарајућег неурона шаље даље до мишићног ткива, које диктира да ли исто треба да се скупи или опружи. Из шире перспективе, може се рећи да је донета одлука о контракцији овог мишића у зависности од улазних стимуланса који су прикупљени. Описани концепт, заједно са принципом рада је главна инспирација и мотивација за креирање вештачког неурона, који је приказан на слици 2.



Слика 2 – Вештачки неурон

Као и биолошки неурон, и вештачки садржи тело које се често назива процесирајући елемент. Неурон прима информације са улазних веза, које су окарактерисане појачањем w_i. У општем случају, неурон може имати произвољан број улаза. На сваком улазу прима податак x_i у виду броја (попут сигнала и стимулуса у биолошком неурону), који се затим множи са кореспондним појачањем. Након множења, резултати који улазе у сам неурон се сабирају. Ова операција се математички може дефинисати као скаларни производ два вектора. Добијена вредност се затим одузима од вредности која донекле подсећа на „нагиб“ или „падину“, склоност да се дати неурон активира (енгл. *bias*).

$$PE(n) = \sum_{i=1}^{k} w_{ni} \cdot x_i - b_n$$

Ако претпоставимо да неурон n има k улаза, за произвољни i-ти улаз у опсегу од [1, k] можемо рећи да се информација x_i множи са појачањем w_{ni}. Појачања овде представљају један од главних параметара који носе део знања оваквог модела. Тешко је интерпретирати који је то део знања, што постаје све теже како саме мреже расту, чак и тамо где целокупно знање и „интелигенција“ представљају најобичнију и једну одлуку о кретању, да ли скренути лево, десно или наставити право. Само у специјалним случајевима где се мреже састоје од свега неколико неурона, одређена интерпретација се и може извести.

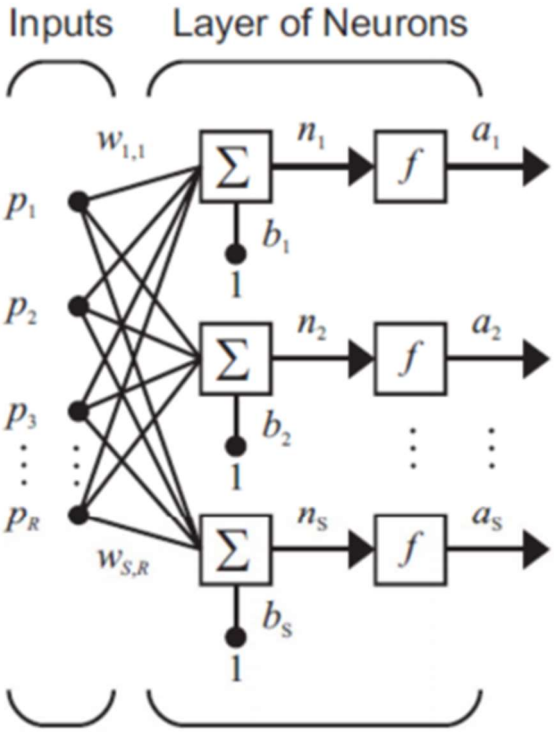
Наредни параметар који такође носи део знања је поменута склоност, под ознаком b_n. Ово се може објаснити по принципу прага проводности. На пример, уколико постоје одређене, веома мале варијације у улазу које не играју никакву улогу у коначној одлуци, овакве варијације треба игнорисати. Склоност као параметар регулише овакве и друге сличне појаве, јер да би одређени улаз или група улаза била прихваћена као битна у доношењу одлуке јер садржи битне информације, потребно је да пређу праг битности, а склоност неурона управо ово дефинише.

Када процесирајући елемент обради информације и добије излаз, потребно је пропустити ову вредност кроз одређену нелинеарност. У општем случају, ово и може да буде линеарна функција, али то уз себе повлачи одређене последице. У сваком случају, оваква функција се у литератури назива активациона функција. Постоји више врста активационих функција. Ове функције практично дефинишу начин на који се неурон активира (на који начин окида) и шаље информације даље ка наредним неуронима у мрежи. То рецимо може бити *hardlim* функција, која за позитивне вредности окида са константном вредношћу, а у супротном не окида уопште. Слична њој је *sign* функција, која за негативне вредности окида са негативном константном вредношћу. У овом раду се разматрају 7 активационих функција [10], које не представљају једине које се могу користити:

- *LTU* (*Linear Threshold Unit*) – линеарна функција у најосновнијем облику;
- *Sigmoid* – логистичка функција која сажима вредности у опсег (0, 1);
- *Tanh* – хиперболичка функција која сажима вредности у опсег [-1, 1];
- *ELU* (*Exponential Linear Unit*) – функција која се у позитивном домену понаша као *LTU*, а у негативном као експоненцијална;
- *RELU* (*Rectified ELU*) – функција као *ELU*, осим што у негативном домену није експоненцијална већ константна 0;
- *LRELU* (*Leaky RELU*) – функција као *RELU*, само што у негативном домену није константна 0 већ линеарна али са малим константним градијентом;
- *Softmax* – функција која оперише над излазима групе неурона, која дате излазе сажима у пробабилистичку дистрибуцију вредности чија је сума једнака 1.

2.1.2. ЈЕДНОСЛОЈНЕ НЕУРАЛНЕ МРЕЖЕ

Када се говори о појму неуралне мреже, сматра се постојање групације неурона који су међусобно повезани на одговарајући начин. Мада, и један неурон засебно може да представља једну веома једноставну „мрежу“ (*Perceptron*, *Adaline*). Како је у таквој ситуацији и сама мрежа једноставна, не може се ни очекивати да буде способна да решава теже проблеме. Отуда и потреба за јачим, већим мрежама. Први корак у овом смеру је да се направи слој (енгл. *layer*) до сада описаних неурона који би имали потенцијал да решавају за нијансу сложеније проблеме од обичних једноћелијских мрежа. На слици 3 се може видети представа описаног слоја.



Слика 3 – Слој вештачких неурона

Код обичне једноћелијске мреже, број улаза у једини неурон је могао да буде произвољан, а како неурон има само један излаз који се воде ка наредним неуронима, али других неурона заправо нема, мрежа отуда ни не може да има више од једног излаза (супротно ни нема смисла). За разлику од једноћелијске мреже, мрежа са једним слојем може да има и произвољан број неурона. Неурони у овом случају нису повезани међусобно, те се излази једног не доводе ни до једног другог неурона. Отуда, мрежа има онолико излаза колико и самих неурона у слоју. Што се улаза тиче, као и код једноћелијских мрежа, број може да буде произвољан. Сада, сваки улаз се води на сваки неурон, те се формира читава структура конекција са својим појачањима. Код једноћелијских мрежа, то је био низ конекција у зависности од тога колико улаза има, а сада се ради о читавом низу једноћелијских структура, па се добија матрица конекција са својим појачањима.

Сваки неурон у слоју, као код једноћелијских мрежа, садржи и параметар склоности. У овом тренутку, гледајући појединачно сваки неурон у слоју, он функционише идентично као и једноћелијска мрежа, односно, рачуна се скаларни производ улазног вектора са кореспондним појачањима, а затим се од дате вредности одузима склоност. Сваки неурон у слоју функционише на овај начин, али са својим скупом појачања (дакле нема дељења појачања, односно веза). Сваки неурон такође има своју активациону функцију кроз коју се процесирана информација прослеђује, мада најчешће у литератури, сви неурони у слоју имају идентичну активациону функцију (уколико је то *Softmax* функција, она ни не може да се дефинише на нивоу индивидуалних неурона, већ се и дефинише на нивоу читавог слоја!). Потребно је математички дефинисати до сада описане појмове:

- $\boldsymbol{p} = [p_1, p_2, p_3, \dots, p_R]$ – улазни вектор који представља улазне податке, дужине R; иако биолошки гледано постоје неурони, претходно описани као сензорски неурони, могуће је и сваки улаз представити као један неурон који кореспондни прикупљени податак пропагира до свих неурона у слоју; у раду се сматра да је улазни вектор већ припремљен од стране сензорских елемената и других неопходних претпроцесирања (попут нормализације), па тиме улазни неурони не би требало да врше никакво процесирање, већ на најједноставнији начин податке директно пропагирају до неурона у слоју; отуда се у наставку рада улазни неурони, када се ради у оквиру контекста неуралних мрежа, исти неће ни помињати;
- $\boldsymbol{W} = \begin{bmatrix} w_{1,1} & \cdots & w_{1,R} \\ \vdots & \ddots & \vdots \\ w_{S,1} & \cdots & W_{S,R} \end{bmatrix}$ – матрица конекција која представља појачања улазних веза, димензије SxR; наиме, S је број неурона у слоју, док је R, као што је већ наведено, број улазних података; сваки ред ове матрице представља управо низ појачања једноћелијске мреже, која је претходно анализирана, а како у слоју постоји S оваквих неурона, природно се ова $1D$ структура унапређује у $2D$ структуру (односно матрицу), где i-ти ред представља низ појача за i-ти неурон у слоју;
- $\boldsymbol{b} = [b_1, b_2, b_3, \dots, b_S]$ – вектор склоности који представља појединачне склоности сваког неурона у слоју, дужине S;
- $\boldsymbol{f}$ – активациона функција слоја; као што је речено, сваки неурон може да има своју сопствену функцију активације, али се у пракси, па и у овом раду користи иста функција за читав слој; функција може да има једну независну променљиву, где се за сваки елемент у том случају појединачно рачуна (енгл. *element-wise*), док у другом случају, може бити дефинисана на нивоу читавог слоја, где узима све податке и пресликава их на одређени начин, али тако да број елемената остане конзистентан (попут *Softmax*);
- $\boldsymbol{a} = [a_1, a_2, a_3, \dots, a_S]$ – излазни вектор који представља излазне вредности неуралне мреже, дужине S; ове информације се могу објаснити као пример где се биолошким неуронима контролише контракција мишића; на пример, један излаз диктира контролисање једног мишића, други другог, док остали излази на пример имају потпуно другачију улогу; ове вредноти се морају интерпретирати на одговарајући начин, у зависности од тога који проблем сама мрежа решава.

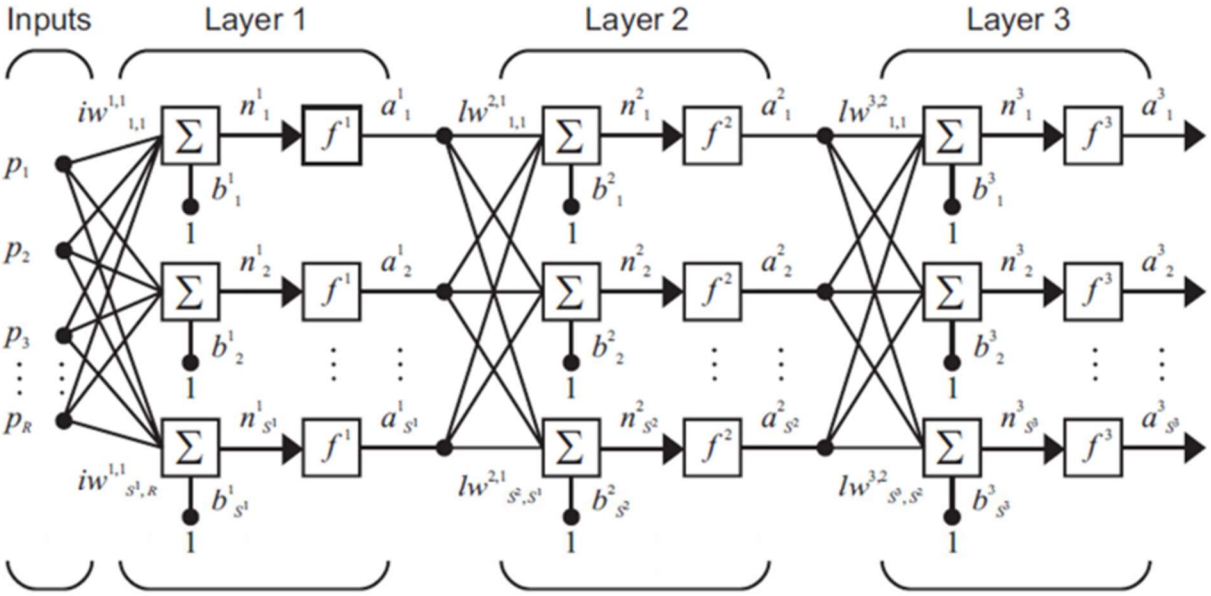
Код једноћелијских мрежа, обавља се скаларни производ како би се добила, скаларна вредност, односно једна вредност која се даље пропушта кроз склоност и активациону функцију. Интересантно је да математички гледано, уз евентуалне транспозиције, може се искористити матрични производ како би се добио, сада вектор вредности дужине броја неурона у слоју S уместо једне, скаларне вредности, где свака вредност управо одговара скаларном производу кореспондентног неурона. У осталом, производ матрица није ништа друго него скуп скаларних производа из перспективе елемената излазне матрице. С обзиром да је недељена димензија вектора $\boldsymbol{p}$ једнака 1, и излазна матрица производа мора за једну димензију имати 1, што се може интерпретирати као вектор, уз евентуалну транспозицију. Након тога, добијени вектор је компатибилан са вектором склоности $\boldsymbol{b}$ и може се исти одузети, а затим се тај резултат пропушта кроз активациону функцију $\boldsymbol{f}$, односно: $\boldsymbol{a} = \boldsymbol{f}(\boldsymbol{W}\boldsymbol{p}^T - \boldsymbol{b})$.

Знак транспозиције је овде примењен на улазни вектор $\boldsymbol{p}$, међутим, често се овај знак изоставља. Наиме, једини разлог зашто је он и постављен овде је да од вектора који се може посматрати као *1xN* матрица транспонује у *Nx1* вектор, односно матрицу која се може множити са матрицом $\boldsymbol{W}$ димензија *MxN*. Међутим, ово производи вектор *Mx1*, који би се даље одузимао са вектором склоности $\boldsymbol{b}$ који је димензија *1xM*; математички исправно би било транспоновати дати производ, међутим, транспозиције не уносе никакво право процесирање које је овде од важности, већ само представљају конвенцију писања, па се тиме могу занемарити, а вектори се увек могу сматрати као матрице било *1xK* или *Kx1* матрице, уколико су сами вектори величине *K*. Стога, излазни вектор слоја $\boldsymbol{a}$ у зависности од улазног вектора $\boldsymbol{x}$ може се записати као $\boldsymbol{a}(\boldsymbol{x}) = \boldsymbol{f}(\boldsymbol{W}\boldsymbol{x} - \boldsymbol{b})$.

2.1.3. ВИШЕСЛОЈНЕ НЕУРАЛНЕ МРЕЖЕ

За разлику од једноћелијских мрежа, које имају свега један излаз, једнослојне мреже имају онолико излаза колико и сам слој има неурона. И једноћелијска мрежа се може сматрати као једнослојна, са једним неуроном. У сваком случају, карактеристично је да дати слој уједно представља и такозвани излазни слој (енгл. *Output layer*). Разлог за ово је једноставно тај да се излази овог слоја третирају као излази читаве мреже (ни не постоје други који би могли). Међутим наредни корак ка креирању сложеније (али и даље једноставне) неуралне мреже јесте тај да се сада убаци више слојева тако да се они каскадно повезују.

Сада, добијамо више слојева који су међусобно повезани на такав начин да се излази одређеног слоја везују за сваки наредни неурон у наредном слоју, где дати наредни слој исте везе посматра као своје улазе. Први слој је једини који нема претходника, већ он прихвата вектор улазних података, док последњи слој не предаје податке ником, па се ти подаци заправо користе као излазни подаци, који се даље интерпретирају на одговарајући начин. Број неурона у сваком слоју је независан од било којих других параметара, па и од броја неурона у околним слојевима. На слици 4 може се видети представа вишеслојне неуралне мреже.



Слика 4 – Вишеслојна вештачка неурална мрежа

Сваки слој понаособ се понаша идентично као једна засебна једнослојна мрежа (математички модел је исти). Разлика је у току података, те сваки слој има своје улазне и излазне податке (као и свој скуп параметара – појачања и склоност, односно поново нема дељења параметара). Активационе функције могу бити другачије од слоја до слоја. Често се у пракси користи иста активациона функција за све слојеве осим за последњи. Последњи слој управо представља излазни слој ове мреже јер се његови излази сматрају излазима читаве мреже, а има их онолико колико неурона има последњи слој. Често се у класификационим проблемима користи *Softmax* функција као излазна како би нормализовала податке у пробабилистичку дистрибуцију за сваку класу, а интерпретација се обавља на начин тако да податак припада оној класи кореспондној неурону са највећом вероватноћом представљеном излазним податком истог неурона. Остали слојеви се називају скривени (енгл. *hidden*) слојеви, а *Softmax* функција често није погодна за њих, па се друге користе.

Као што је већ речено, може се такође дефинисати улазни слој неурона, међутим такав слој не врши никакво процесирање улазних података, већ би само сваки податак прослеђио до сваког неурона првог скривеног слоја. Стога се он, као и раније, може једноставно занемарити.

2.1.4. Знање неуралне мреже и параметри

До сада су математички дефинисани начини функционисања неуралних мрежа, од најједноставнијих једноћелијских мрежа до вишеслојних. Сваки неурон садржи скуп параметара, а то су појачања улазних веза и склоност за активацију. Шире гледано, свака мрежа има одређени квантитет ових параметара. У њима је сачувано знање мреже, а што је мрежа већа, број ових параметара расте, па и њен капацитет за решавање тежих задатака, односно садржи потенцијал за учење тежих ствари, као и да садржи „веће“ знање. Ово не значи да су увек веће мреже нужно и боље.

Параметри мреже су они који такође имају поменуту особину способности учења. Када се ови параметри промене, мрежа тиме добија ново знање, које може бити добро, лоше или неутрално. Заједно представљају једну функцију која је све више комплекснија што је сама мрежа већа. Ову функцију треба формирати тако да што мање греши, односно да што боље доноси одлуке, класификује податке, рачуна вредности или ради оно што јој је и сама намена (да добро моделира податке, енгл. *fit*).

Знање мреже такође не зависи од само појединачних конекција. Могуће је да се одређене конекције једноставно обришу (тако што се поставе на 0). Мрежа би и даље функционисала, највероватније нешто слабије (ако и уопште), што зависи од броја конекција које се бришу, као и њихове значајности. Често се мреже могу кратко ретренирати након овога како би вратиле штету назад (енгл. *brain damage*).

Неке напредне методе понекад и намерно бришу везе између неурона, па чак и читаве неуроне, што је еквивалентно брисањем свих веза које иду ка том неурону, и из тог неурона.

2.1.5. СЛОЈЕВИ НЕУРАЛНИХ МРЕЖА

Иако је у моделу вишеслојне НМ описано да се последњи слој користи као излазни, а да се остали називају скривени, у овом раду се ипак не прави разлика између ових слојева (мада она свакако ни не постоји). Неуронска мрежа се у овом раду дефинише као вектор слојева где сваки слој прима податке од претходног слоја (улазни вектор ако је у питању први слој), од кога на одређени начин прави излазни вектор података који прихвата наредни слој, и тако до последњегм излазног слоја, који даље не прослеђује податке, већ се ти подаци третирају као излази целокупне мреже.

Слој сам по себи треба да дефинише операције које врши над улазним подацима да би направио излазне податке (у оквиру тог истог слоја). Гледано из перспективе рада, до сада је описан само један тип слоја, а то је такозвани потпуно повезани слој (енгл. *Fully Connected Layer*, или чешће скраћено *FC layer*). Сваки потпуно повезани слој је управо дефинисан на начин као и обична једнослојна неуронска мрежа, па уколико је потребно направити вишеслојну мрежу, то је могуће тако што се више потпуно повезаних слојева додају у вектор слојева саме неуронске мреже, један након другог.

Потпуно повезани слој је само пример и представља најједноставнији слој. Имплементацијом је омогућено да корисник сам прави произвољне слојеве ради истраживања и даље анализе могућности у овој области, о чему ће више речи бити у наредним поглављима. Такође, могуће је правити хиперслојеве, односно слојеве који у себи садрже подслојеве, а такође и ти слојеви могу да садрже даље подслојеве (теоријски ово се може понављати до неограничене дубине), чиме се формира читава хијерархија слојева. У суштини, један хиперслој се може посматрати као засебна подмрежа у оквиру неке веће мреже, што се може користити у неуронским мрежама са сложенијим архитектурама.

2.2. АНАЛИЗА ГЕНЕТИЧКОГ АЛГОРИТМА

У области софтверског инжењерства, веома често се наилази на проблеме где је неопходно извршити одређену претрагу у простору који је не тако ретко изузетно обиман. Претрага у таквим ситуацијама тражи хипотезе проблема, односно потенцијална „решења“ проблема. То наравно може бити право, најбоље (глобално) решење које се може пронаћи, мада често је и „довољно добро“ решење задовољавајуће, иако није најбоље, односно једна довољно добра хипотеза постаје практично прихватљиво решење.

За претрагу се такође мора дефинисати такозвана оптимизациона функција. Наиме, она је та која заправо пореди хипотезе и говори која је боља. Две најосновније технике за претрагу простора јесу систематично (потпуно) претраживање и случајно претраживање. Међутим, само када је простор претраге мали, ове две технике се могу употребити, али таквих проблема је изузетно мало. Чим се простор претраге и мало повећа, ове методе постају неефикасне, па је потребно наћи оптималнији начин претраге. С обзиром да је систематско претраживање једини начин да се са сигурношћу пронађе најбоље решење у најопштијем случају, сада претрага постаје хеуристичка, делимична, таква да се у сваком кораку тражи боље решење од текућег, разним опзимизационим инжењерским техникама.

Оптимизациона функција сада постаје предмет оптимизације (отуда и име). Суштински, у сваком кораку, пронађена хипотеза се пореди са до сада најбољом пронађеном, а управо оптимизациона функција има способност да ово и уради. Разни инжењерски оптимизациони алгоритми су развијени који се управо баве оваквим проблемима. На најгрубљи начин, могу се поделити у две независне групе по две категорије:

	Детерминистички	**Стохастички**
Локални	- НМ симплекс - Данциг симплекс - Градијентни спуст (енгл. *Gradient Descent*)	- Симулирано каљење - Стохастички градијентни спуст (енгл. *Stochastic Gradient Descent*) - Еволутивни алгоритам - Диференцијална еволуција - *Particle Swarm Optimization* - *Ant Colony Optimization*
Глобални	-Систематско претраживање	-Случајно претраживање -Генетички алгоритам

Табела 1 – Груба подела неких од алгоритама претраге

Табела 1 никако не представља научно/експериментално подстакнуту поделу. Први вид поделе алгоритама је на „глобалне“ и „локалне“. Генерално, у оптимизационим проблемима, честа је појава такозваних „локалних минимума“, односно локално оптималних решења. Локални алгоритми по покретању имају могућност само проналажења оваквих минимума, док глобални алгоритми по покретању имају могућност проналаска глобалног, најбољег решења. Многи алгоритми заправо локално оптимална решења и траже. Иако су најбоља решења представљена глобалним минимумима, то не значи да локалне алгоритме не треба узимати у обзир. Наиме, разлози су вишеструки зашто су ови алгоритми добри.

До сада је већ поменуто да је у пракси, често довољно пронаћи неко задовољавајуће решење које не мора бити глобално. У ову сврху, локални алгоритми се итекако користе због своје ефикасности у датим проблемима. При томе, и сам глобални минимум се може сматрати локалним у својој околини. Дакле, иако је алгоритам локалан, не значи да не може да пронађе глобални минимум. Ово највише зависи одакле се претрага започне. Уколико је почетак добро одабран, ове методе показују брзу конвергенцију ка глобалном (локалном у тој околини) минимуму, али је неопходно добро предзнање проблема. За неке од ових алгоритама се и може дискутовати да су на граници између локалних и глобалних алгоритама.

Други вид поделе јесте тај да алгоритми могу бити детерминистички и стохастички. Детерминистички алгоритми по покретању увек врше идентичну претрагу,

самим тим и исто решење проналазе, док стохастички уносе неки вид случајности, па се по покретању претрага може драстично разликовати. Случајно претраживање је управо један пример оваквог алгоритма.

Из табеле 1 се може видети да систематско претраживање представља технику која детерминистички увек даје глобално решење, али је већ дискутовано да је ова метода изузетно неефикасна и за мало комплексније претраге. Слично се може изложити за случајно претраживање. Преостали алгоритам који има способност проналажења глобалног минимума јесте генетички алгоритам. Међутим, ГА никако не представља једини алгоритам који има ову особину, и додатно уколико се не употреби на одговарајући начин, његово понашање престаје да има особине способности тражења глобалног минимума и деградира у претрагу локалних решења.

Генетички алгоритам такође представља само један од низа алгоритама из фамилије еволутивних алгоритама. Наиме, еволутивни алгоритам (ЕА) представља најопштији алгоритам у овој фамилији. Дакле, ГА у својој конкуренцији, поред ЕА, има и диференцијалну еволуцију, *PSO*, *ACO* и друге.

2.2.1. Еволуција као оптимизациони процес

Главна идеја и мотивација која стоји иза ГА јесу еволутивни процеси у природи. Интересантно је да еволуција заправо представља један опзимизациони процес. ГА се заснива на Дарвиновој теорији еволуције [11], теорија која је објединила биологију. Даље, у ове процесе, потребно је укључити интелигенцију. Данас се сматра да су интелигенција и еволуција нераздвојни концепти. У оквиру еволуције, постоје три нивоа учења:

- Филогенетичко – учење кроз наследство;
- Онтогенетичко – учење кроз искуство током живота;
- Социогенетичко – учење кроз интеракцију у групи.

Локални оптимизациони алгоритми се могу схватити као апстракција онтогенетичког учења. На основу неких правила, једно решење се унапређује и траже се боља. За разлику од њих, ГА се може схватити као апстракција филогенетичког учења, кроз искуства претходних генерација које је сачувано у генима.

Основна идеја алгоритма је опстанак из природе. У алгоритму не постоји само једно решење које се унапређује, већ читава група потенцијалних решења, односно хипотеза. Ова група решења се назива **популацијом**. Свако решење у оквиру популације може се окарактерисати као **индивидуа**, јединка или биће које је део еволуционог процеса. Свака индивидуа је сачињена од **ДНК** (ова два појма су суштински синонимна), који је сачињен од **гена**. Ген представља једну оптимизациону променљиву у запису оптимизационог проблема. Такође представља најмању јединицу за рекомбинацију, односо за претрагу у простору.

Поред тога, може се дефинисати функција **способности** индивидуе (енгл. *fitness*). Ова функција говори колико је одређена индивидуа адаптирана у еволуцији, колико је напредна у односу на остале, тј. колико је до сада унапређена и добро се понаша, односно представља добро решење. Из перспективе оптимизације, функција способности такође представља уједно и оптимизациону функцију. Стога, она нам говори колико је до сада пронађена хипотеза, односно решење добро. Треба напоменути да у одређеним

проблемима, функција способности представља хеуристичку функцију која процењује апстрактну праву функцију оптималности, јер у тим проблемима, права функција оптималности потенцијално ни не постоји. Међутим, чак и када постоји, често је довољна апроксимација како би еволуција била успешна.

Процес еволуције тече кроз **генерације**. Наиме, популација у одређеном тренутку еволуције назива се генерација. У току еволутивног процеса, једна генерација актује, а након што она достигне свој крај, на одређени начин се од ове генерације прави нова, где се стара више не разматра. Нова генерација наравно постаје популација еволутивног процеса. Идеја је да се кроз генерације знање преноси и побољшава, и наравно конвергира ка најбољим (пандан високо еволуиране индивидуе/популације).

У овом процесу, потребно је да постоји полазна генерација, често названа као прва или чак нулта генерација. Она заправо представља полазни скуп решења, који се може формирати на произвољан начин, најчешће на случајно. У општем случају, ГА не залази у начин формирања полазне генерације. Свака наредна генерација се прави од најбољих решења претходне генерације. На тај начин, следећа генерација би требало да има решења која су боља од претходне. Ипак, ова претпоставка зависи од много фактора, мада се у пракси показује да чак није ни обавезна, а понекад је чак и непожељна, јер у тим ситуацијама алгоритам, иако има бржу конвергенцију, може да конвергира ка локалном минимуму.

2.2.2. Критеријуми еволуције и операције у ГА

У природи, еволуција се заснива на неколико принципа: размножавање, мутација, такмичење, опстанак, селекција. Да би систем могао да буде еволутиван, потребно је да постоје 3 испуњена критеријума:

- Наследност (енгл. *heredity*) – потребно је да постоји неки механизам по ком се информације претходника преносе на следбенике;
- Варијација (енгл. *variation*) – потребно је да постоји неки вид варијације (разноликости, па и промене) како систем не би стагнирао;
- Избор (енгл. *selection*) – потребно је да постоји неки вид избора у систему који диктира пренос информација на следбенике (диктира наследност).

Ови критеријуми су у ГА јасно дефинисани као поједине операције које имитирају принципе еволуције у природи. Наиме, првенствено се дефинише операција **укрштања** (енгл. *crossover*). Овом операцијом, индивидуе које се укрштају праве нову индивидуу (или понављањем више) која представља комбинацију укрштајућих. Укрштајуће индивидуе се често називају **родитељима** (енгл. *parents*), док се новонастала индивидуа назива **потомак** (енгл. *offspring*). Потомство (следбеници) управо кроз укрштање наслеђује информације од родитеља (претходника) тако што се гени родитеља који управо ове информације носе постављају у ДНК потомака. Овим је испуњен први критеријум еволуције (наследност).

Већ је поменуто да се у оквиру ГА не прати само једно решење, већ читава група, односно популација решења (прецизније хипотеза). Популација сама по себи испуњава критеријум варијације, под условом да су ова решења разнолика. Ово се често веома лако постиже тако што се полазна популација направи да буде што је разноврснија могућа, а

случајно изабрана решења управо овоме доприносе. Такође, повећање саме популације такође доприноси варијацији, али наравно са собом повлачи друге мане.

Наредна операција која се дефинише је операција **мутације**. Ова операција је можда и централна у овом алгоритму, али никако сама по себи довољна. Мутација се у основном облику ГА појављује у тренутку настанка потомства. Одређени (или неодређени) гени наслеђени од родитеља се излажу **променама** које су најчешће стохастиче природе, односно готово увек у себи садрже барем неки вид случајности. Ове промене уносе ново знање, у систем, новине које су потенцијално до сада невиђене, а које наравно могу бити и позитивне и негативне. Мутација је често окарактерисана вероватноћом да се догоди на сваком гену након укрштања, и по својој природи учествује у доприносу на варијацији у еволутивном систему.

И мутација и укрштање зависе од записа гена у ДНК, често званог **генотип** (енгл. *genotype*). Наиме, сам ГА не може ни не треба да специфицира ове операције. Оне су увек везане за ДНК и при пројектовању ДНК, и ове операције се такође пројектују. Могуће их је такође додатно параметризовати, тако да им се понашање додатно контролише. У овом тренутку, добро је поменути да сам генотип не представља директно једно решење проблема, иако је до сада било речи о томе да једна индивидуа управо то и представља (у општем случају ово разматрање је тачно, али постоје специфичне ситуације где генотип и може директно представљати једно решење). Поред генотипа, постоји и концепт **фенотипа** (енгл. *phenotype*). Фенотип је управо решење о коме је до сада било речи. Фенотип представља исход самог генотипа, оно што се од информација које су сачуване у генотипу (у генима) конструише, што управо представља наше решење (односно хипотезу). У општем случају, ове две ствари су одвојене, па генотим представља индиректно једно решење проблема. У неким ситуацијама ова два типа се подударају, па се губи потреба раздвајања ова два концепта, те се у тим ситуацијама може рећи да генотип и представља директно једно решење проблема.

Последња операција која се дефинише у ГА јесте операција **избора**. Само име говори да се ова операција бави трећим критеријумом еволуције, односно критеријумом избора (селекције). Наравно, овом операцијом се критеријум и испуњава. Ово чак није и једини разлог за постојање ове операције у ГА. Она се такође и природно намеће. Раније је наведено да се индивидуе укрштају како би се потомство направило (операција укрштања), међутим, питање које се овде намеће је да ли се зна и које су то индивидуе које учествују у укрштању? Операција избора управо одговара на ово питање.

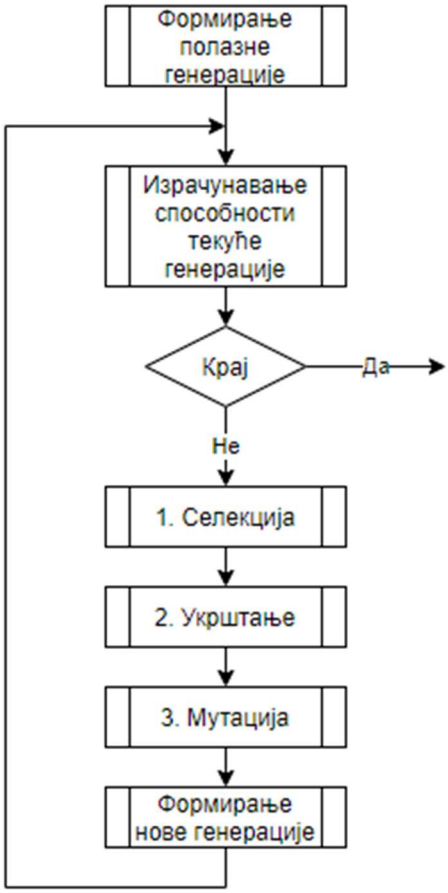
У општем случају, број индивидуа, односно родитеља који се укрштају не мора бити 2, као што је познато из природе. Овај број може бити произвољан, укључујући и 1. Родитељи наравно морају да потичу из текуће генерације, а потомство представља наредну генерацију. Одавде се може претпоставити да читава родитељска генерација учествује у креирању потомства, што није немогуће, а у одређеној ситуацији, овакав начин не би ни испуњавао трећи критеријум еволуције, критеријум избора. Реч је о томе да се читава популација укршта како би створила једног потомка, па се овај процес затим понавља док се не створи читава нова генерација.

Операција избора управо на одређени начин бира индивидуе које ће учествовати у укрштању и стварању наредне генерације. Сама операција не зависи од типа представе индивидуе, односно ДНК. Такође, овом операцијом се може и ограничити да одређене индивидуе ни не учествују у размножавању уопште! Постоји неколико начина како се

операција избора може дефинисати. Један начин је да се одабере део популације и да само тај део учествује у стварању нове. Други начин је да се за сваког потомка бирају родитељи (на неки унапред дефинисан начин) који ће учествовати у укрштању (поново, може их бити произвољно много, понекад чак и иста индивидуа може биди више пута изабрана), чиме се, уколико то унапред није уграђено, даје шанса свим индивидуама да барем једном учествују у неком укрштању и свој генетски материјал пренесу у наредну генерацију. Међутим, и даље постоји шанса у таквој ситуацији да одређене индивидуе неће бити одабране, јер често у оваквим имплементацијама у позадини стоји неки вид случано генерисаних бројева (стохастичности), што не представља негативну особину, већ напротив, лоша решења углавном и треба одбацити.

2.2.3. Ток еволуције у ГА

Као што и сам назив ГА сугерише, ради се о алгоритам у ком се еволуција догађа. Дакле, алгоритам није ништа друго него поступак којим се одређен проблем решава, односно поступак којим се производи резултат који представља циљ рада самог алгоритма. Тако и ГА има своје кораке који чине сам поступак који се у њему одвија. Треба напоменути да је ово само једна варијанта ГА (формални и основни ГА), а постоје и разне друге варијанте. У суштини, било какав алгоритам који поштује критеријуме еволуције се може сврстати у фамилију еволутивних алгоритама, а такви алгоритми који уједно садрже концепт гена аутоматски постају део те фамилије где се налазе варијанте ГА. На слици 5 је приказан ГА у основном облику:



Слика 5 – Основни облик ГА

Као што је већ речено, првенствено се формира полазна генерација која ће представљати популацију на почетку алгоритма. Након тога се улази у петљу која се потенцијално одмах може и завршити. Наиме, прво је потребно израчунати способности текуће генерације, како би се проценила решења која постоје у њој. Већ је помињано да ово може бити хеуристичко. У таквим ситуацијама, додатно је отежано препознавање најбољег глобалног решења, уколико је оно и пронађено. У сваком случају, ако се пронађе задовољавајуће решење након ове евалуације, алгоритам може да стане. Често се и предефинише максимални број генерација како би алгоритам у неком моменту сигурно стао са најбољим до тада нађеним решењем, уколико наравно пре тога није постојао неки други испуњен услов за излазак из алгоритма.

Уколико се ипак наставља са еволуцијом, у овом тренутку се раде операције које су до сада објашњене. Прво се врши селекција родитеља који ће учествовати у укрштању (било на нивоу читаве популације или индивидуално на нивоу потомка). Затим се изабрани родитељи укрштају и стварају се потомци. Величина наредне популације је иста као и величина текуће, па се тиме кореспондне операције позивају онолико пута колико је то потребно и довољно. Након што се потомство створи, врши се операција мутације. Још једно битно својство мутације је да је оно кључно за претрагу других делова простора који до сада нису истраживани. Ово је последица тога да је мутација операција која уноси често апсолутну случајност у одређеној количини. Без ње, систем конвергира ка локално добрим решењима. Међутим, већ је помињано да мутација сама по себи није корисна (сама би деградирала у неки вид или чак директан алгоритам случајног претраживања), али као део ширег еволутивног (генетичког) система представља веома корисну функционалност.

Сада, након мутирања потомства, читава претходна генерација се замењује новом, која се сада оцењује, односно рачунају се способности свих њених индивидуа и поново тражи решење које је потенцијално задовољавајуће. Уколико се такво решење или решења не пронађу, поступак се на исти начин наставља.

2.3. Анализа неуро-еволуције

До сада два описана концепта, концепт НМ и концепт ГА, потпуно одвојено функционишу, односно имају свој домен у којима се користе. Међутим, до сада су НМ само представљене како се користе, како функционишу, под претпоставком да су већ подучене и да већ имају потребно знање. Ако бисмо почели од почетка, и имплементирали математички модел саме неуралне мреже, природно би се постављало питање, о почетним вредностима појачања, склоности и других евентуалних параметара. Прва интуитивна ствар је да се ове вредности случајно генеришу. Међутим, веома су мале шансе да се баш генерише одмах мрежа која потребно знање и поседује. Могуће је даље наставити генерисање других случајних вредности, а у овом тренутку, ако се из шире перспективе погледа, ово заправо представља случајну претрагу.

Проналажење правих вредности параметара неуралних мрежа је управо један оптимизациони проблем, проблем претраге, и представља обучавање, односно тренирање мреже. Данас, популаран начин тренирања НМ је помоћу метода градијентног спуста, обично стохастичке варијанте (познатије као *SGD*). Ова метода, заједно са својим варијантама и побољшањима не само да представља популарну методу, већ и методу која се у пракси показује као веома ефикасна за тренирање неуралних

мрежа, и представља главну конкуренцију било којим новим методама које се истраживају.

2.3.1. Тренирање НМ помоћу ГА

Још један од начина како се саме неуралне мреже могу тренирати, или прецизније група неуралних мрежа може тренирати, јесте помоћу ГА. Иако тежина имплементације ове идеје може врло брзо да ескалира, сама идеја је поприлично једноставна. Неуралну мрежу треба посматрати из перспективе „учећих параметара“ (енгл. *learnable parameters*). Управо до сада помињани параметри (појачања и склоности) представљају параметре који имају способност да науче, да приме знање, кроз алгоритам који их обучава. Скуп ових параметара представља једну хипотезу, једно потенцијално решење за неуралну мрежу. Наравно, одмах се намеће питање да ли је ово решење прихватљиво, али, то је управо сврха претраге, коју сада можемо обављати помоћу ГА.

Сваки учећи параметар НМ представља један ген у ГА. У најосновнијој НЕ, ово је дефиниција која стоји иза саме идеје. Креира се почетна популација неуралних мрежа, које су обавезно исте структуре. Како би се унела варијација у систем, параметрима (генима) се поставе случајне вредности. Наиме, нигде није назначено о каквим се вредностима овде ради. Теоријски, могуће је да овде стоје било какве вредности. У пракси, па и у овом раду, то су најчешће неке мале вредности око нуле. Обично се користи опсег [-1, +1]. Међутим, овај критеријум је слабе природе. Нигде се не ограничава да поједине вредности изађу ван овог опсега, али најчеће такве вредности не иду много даље од њега. Исто тако, често се и улазни подаци у саму мрежу на неки начин нормализују како би се свели у овакав или сличан опсег, како би били „компатибилни“ са параметрима унутар мреже (компатибилност је такође слабо дефинисан појам).

Након креирања прве популације неуралних мрежа, свака мрежа се сада евалуира на начин зависан од саме употребе. Уколико постоји задовољавајућа неуралн мрежа, или више њих, оне се могу одабрати за даље коришћење, мада ово је ретка појава, поготово у ситуацијама где су мреже веома велике и дубоке. Затим, по генетичком алгоритму, уколико се наставља са еволуцијом, улази се у процес селекције, укрштања и мутације и прави се наредна генерација. Процес селекције, као што је већ речено, није зависан од саме представе гена и проблема генерално, па се донекле и може стандардизовати.

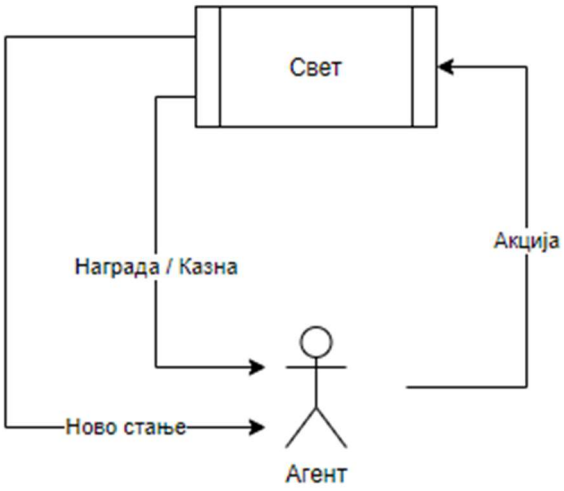
Процес укрштања се може дефинисати на више начина. Јасно је дефинисано да сваки параметар представља један ген, оно што је дакле битно урадити је да се сваки параметар поравна са кореспондним параметром других индивидуа које учествују у датом укрштању. Ово избегава да се, на пример, мешају гени који носе различите информације, на пример појачање и склоност. Чак и мешање гена различитих појачања је угланом непожељан ефекат.

Мутација на насумичан начин са одређеном вероватноћом мења сваки ген, односно вредност сваког параметра. Ово је једини моменат где саме вредности параметара могу да изађу ван поменутог опсега. Наравно, могуће је и ограничити да се дате вредности строго налазе у том опсегу, али у пракси се показује да то није неопходно, и да чак може бити непожељно. Генерално, идеја малих вредности је да се имитирају сигнали који се преносе у биолошком нервном систему, који се такође могу окарактерисати као мали импулси.

Као што је и раније помињано, мутација управо уноси новине у сам систем. Без ње, само би се комбинације почетне популације неуралних мрежа испитивале, док се уз мутацију на овај начин претражују и други делови простора претраживања и оптимизације. Након мутације, створена је нова генерација неуралних мрежа, и може се наставити са њиховом евалуацијом, па и даљом еволуцијом.

2.3.2. УЧЕЊЕ СА ПОДРШКОМ

Принцип НЕ се веома добро понаша у УП системима. Наиме, природно постоји веза између ова два концепта. Као што је већ описано, УП се може повезати и са психологијом човека. Првенствено, треба детаљније описати сам принцип УП. На слици 6 се може видети најопштија блок шема принципа УП.



Слика 6 – Принцип учења са подршком

Када посматрамо неки УП систем, у њему постоји свет, у ком се налази неки агент (актор), који има могућност извршавања различитих акција у оквиру света (иако је на слици агент издвојен, он се заправо налази у оквиру света). Уколико агент не би вршио никакву акцију, систем би се увек налазио у једном **стању**. Међутим, како агент направи неку акцију систем прелази у наредно стање које свет (такође се често назива и околином) диктира. Агент у овом тренутку такође прима битну информацију у виду награде или казне (која може бити значајног или небитног карактера), на основу које (уколико је и сам способан) може да учи. Ова информација практично представља подршку његове акције (отуда и име, учење са подршком). Што је та акција била боља, награда је већа. Уколико је акција безначајна, и сама ова информација (која се иначе назива сигнал подршке) је безначајног карактера. Уколико је акција агента била лоша по њему, и сигнал подршке престаје да буде попут награде, већ постаје казна, која је све већа што је сама акција била гора. Наравно, стање самог система, односно света је боље по агенту, што су сигнали подршке бољи и обрнуто.

Често у оваквим системима није могуће одмах дати сигнал подршке јер није познато да ли посматрана акција агента јесте добра или није. Често се ово сазнаје тек

након неког времена, када је то и потенцијално касно (тек тада се свет мења у корист или штету агента). Међутим, ако постоји одговарајући метод који је у стању да учи из оваквих ситуација, могуће је направити агента који ће знати које акције и када треба да ради, као и то да унапред предвиди шта је у текућем тренутку добра акција.

Интересантно је да функција способности ГА представља управо савршен механизам за представљање сигнала подршке у УП. Што је дата индивидуа у оквиру ГА боља, то значи да је вредност функције способности те индивидуе веће. Ова вредност се може употребити као сигнал подршке у УП. Потребно је само добро дефинисати саму функцију способности ради формирања добгор сигнала подршке, али често у УП не постоје и чак и нису ни потребни веома прецизни модели сигнала подршке, већ и најједноставније хеуристичке апроксимације се у пракси показују као веома ефикасне. Већ је поменуто да је ово такође и особина функција способности у ГА, да се могу или чак и морају користити хеуристичке апроксимације у појединим ситуацијама. Дакле, природно се испоставља да су ова два концепта компатибилна управо због поменутих карактеристика.

Са друге стране, представљено је да се ГА може користити за тренирање групе неуралних мрежа. Стога, ГА практично представља спону између УП и НМ, а заједно чине концепт неуро-еволуције у учењу са подршком, што је и тема овог рада. Агенти који актују у свету садрже неуралну мрежу која диктира њихово понашање. Улази у неуралну мрежу представљају све информације прикупљене од сензорских компоненти које представљају битне или небитне одлике у доношењу одлука коју акцију агент треба да предузме у датом тренутку. Излази управо представљају неки вид кода који након предефинисане интерпретације даје управо информацију о акцији коју треба предузети. Унутрашњи делови мреже представљају процесирајуће елементе који поседују одговарајуће знање за доношење дате одлуке, која треба да буде квалитетна. Генетичким алгоритмом ове мреже пролазе кроз еволуцију и унапређују своје понашање, где функција способности представља сигнал подршке који управо има претходно описану улогу.

Треба напоменути да се на овај начин не оптимизују неуралне мреже директно, већ као што је речено, ГА оптимизује функцију способности јер она заправо представља оптимизациону функцију у оквиру претраге. Самим тим, у оваквим системима, оптимизује се сигнал подршке, па се често догађа ситуација да се оптимизује нешто што није унапред предвиђено и саме неуралне мреже почну да уче нус ефекте који могу бити непожељни, мада некад интересантне природе, али понекад и могу да науче нешто позитивно (али непредвиђено). Коректном изменом функције способности ови ефекти се могу кориговати.

Мутација у оваквим системима управо дефинише однос истраживања нових акција и експлоатације до сада научених ствари (енгл. *exploration/exploitation tradeoff*). Наиме, уколико се мутација укине, само се најбоље понашање до тог тренутка експлоатише, односно агенти неће покушавати нешто ново (локална оптимизација). Уколико је мутација присутна у превеликој количини, агенти ће непрестано радити нове ствари, без коришћења стеченог знања која могу да употребе, а пандан томе је случајна претрага, па експлоатисања нема. Када је мутација добро подешена, агенти ће повремено покушавати нове ствари, чиме потенцијално излазе из јаме локалног минимума и усмеравају се на пут ка глобалном.

3. ИНФРАСТРУКТУРА НА НИЖЕМ НИВОУ

Иако до сада описани концепти представљају основицу за реализацију УП система, већ при самом почетку имплементације намеће се потреба за постојање подсистема који је по природи још нижи, чије функционалности НЕ и њени делови користе. Као пример, може се узети математички модел НМ. Овде се помињу вектори, матрице, скаларни производи, множење матрице, генерално појмови алгебре.

Поред тога, како се не би све изнова имплементирало за сваки проблем посебно, добро је да се наведени принципи имплементирају у виду библиотеке која би се даље у датим проблемима користила. На овај начин се добија реупотребљив ресурс и код који доприноси на продуктивности и олакшава имплементацију крајњих система. Ово не представља проблем за ниже концепте, штавише ово је и чест начин који се у пракси користи, а такође се у библиотеку могу уградити и концепти саме НЕ, како се и ови не би морали поновно имплементирати.

Међутим, овде ситуација постаје делимично комплекснија. Генерално, ГА је такав алгоритам да садржи веома много параметара (не учећих) који се подешавају пре него што се сам алгоритам покрене. Ови параметри се називају **хиперпараметри**, а ГА није једини који исте и има. Наиме, све што није учећи параметар, а представља неки вид параметра који се подешава (најчешће) пре покретања представља хиперпараметар. На пример, структура саме НМ је хиперпараметар. Број неурона у скривеном слоју је хиперпараметар. Број генерација у еволуцији је хиперпараметар. Даљом анализом може се врло брзо уочити да ГА има велик број хиперпараметара, а и поред тога, као што је већ описано, ГА може да има много различитих варијанти, као и то да је најчешће његова имплементација зависна од конкретног проблема, па се тешко може апстраховти у неки општи ГА који би се користио кроз било какав проблем који би њега користио (из библиотеке). А како ГА представља главни спој између НМ и УП, ово може да представља проблем. Међутим, ГА у оквиру овог рада ће се већински тицати само НЕ, па се донекле и може издвојити варијанта која би се користила само у НЕ за УП, или сличне примене.

Још један разлог за постојање библиотеке јесте оптимизационе природе, али сада у смислу перформанси. Наиме, у свету постоје различите технологије које се користе у различитим применама, зависно од потребе и конкретних проблема. Њих прате језици у којима се дате технологије програмирају. Када су перформансе потребне, често се прибегава језицима ниског нивоа, како би се приближило хардверу што је више могуће. У овом раду, изабран је језик *C++* у ту сврху, којим би се креирала инфраструктура на коју би концепти НЕ у овом раду били засновани. Као што је већ поменуто, и НЕ ће бити део ове библиотеке, али је битно напоменути да се у оквиру библиотеке налазе два слоја, слој са најнижим концептима који не познају појмове неуралних мрежа, генетичког алгоритма, неуро-еволуције и других, али чине најосновније операције које ови појмови управо користе, а сами појмови се дакле налазе у другом слоју, а који користе оно што први слој пружа.

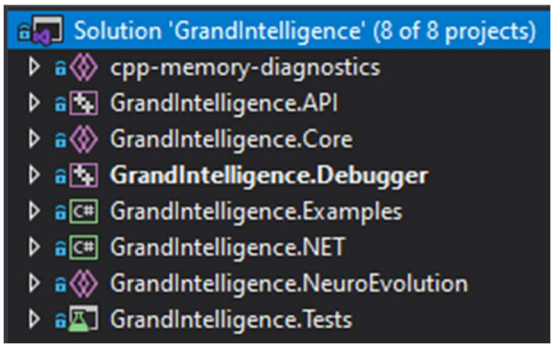
Помоћу језика *C++* могуће је направити динамички линковану библиотеку (енгл. *dynamic link library – dll*), која се може учитавати не само у другим програмима писаним на истом језику, већ може и представљати такозвану неуправљану (енгл. *unmanaged*) библиотеку која се користи у управљаним (енгл. *managed*) технологијама и језицима,

попут *.NET* технологије и језика *C#*, који, иако намењени за системе где је примена другачија, где не постоји потреба за толиким спуштањем до хардвера, па самим тим не постоје ни потребе за перформансама тог нивоа, и даље могу да користе овакве библиотеке у тренуцима када су перформансе ипак потребне. Кроз симулације у поглављу 5 управо ће се *.NET* технологије користити како би тестирале ову библиотеку где ће се такође и перформансе мерити.

Још један разлог за постојање библиотеке, нарочито кроз два слоја, јесте сама имплементациона природа алгоритама и појмова који су потребни. Данас, хардверске технологије напредују ка паралелизацији операција које се извршавају ради убрзавања читавог процеса како би се перформансе подигле што је то више могуће. Ово наравно захтева да се делови алгоритмама који диктирају дате операције могу паралелизовати, односно да сами алгоритми тако функционишу да буду компатибилни са паралелизацијом на графичким картицама које управо овакву особину поседују. Алгебарске операције за НМ које су поменуте природно овакву особину имају, а такође и делови ГА имају потенцијала за паралелизацију, па тиме читава НЕ може да добије додатно убрзање у извршавању уколико се ова парадигма укључи. У овом раду ће од интереса бити *Nvidia* графичке картице које се могу програмирати помоћу *CUDA Toolkit* [12], где се могу програмирати паралелни алгоритми. У оквиру библиотеке, намеће се потреба за постојањем моста (енгл. *bridge*) који омогућује да се бира место извршавања алгоритама. Као могући избори, у овом раду су имплементирани алгоритми и на *CPU*, као и на *NvidiaGPU*, али се природно софтверским узорком моста оставља потенцијал за унапређење, где би се и други типови уређаја касније додавали, на којима би се даље алгоритми извршавали.

3.1. АРХИТЕКТУРА СИСТЕМА БИБЛИОТЕКЕ

По почетку имплементације саме библиотеке, врло брзо се намеће потреба за различитим услужним (енгл. *utility*) функционалностима које доприносе како квалитету саме библиотеке, тако и могућности и продуктивности у њеном развоју. Стога, потребно је паралелно развијати и друге услужне подпројекте који управо имају ову сврху. На слици 7 предсатвљени су сви пројекти који су укључени у изградњи библиотеке.



Слика 7 – Пројекти укључени у изградњи библиотеке

Као што је већ речено, коначни исход изградње овог решења (енгл. *Solution*) треба да буде једна *.dll* датотека која садржи описане имплементације која би се даље

укључивала у друе пројекте који би њу и користили. Иако ово делује као да је један пројекат довољан, на слици 7 се може видети 8 учитаних пројеката.

3.1.1. ГЛАВНИ ПРОЈЕКТИ

Треба почети од главних пројеката који се тичу 2 поменута слоја, један са имплементацијама најнижих појмова, и други који садржи имплементације НЕ, а који користи први слој. С обзиром да други слој користи први, а како је први слој самосталан (енгл. *standalone*), први слој се такође може имплементирати као библиотека коју би сада други слој користио. Међутим, овај начин се из више разлога испоставља као непрактичан. Један од разлога јесте експлозија *.dll* датотека. У овој ситуацији заправо би постојала само још једна додатна таква датотека, али уколико се систем у будућности буде унапређивао, овај, као и други проблеми који би у оваквој архитектури настали, могу се заједно решити такозваном *C++ Shared Project* парадигмом. Идеја је да се кôд овог пројекта дели (отуда и име, дељени пројекат) кроз више других *C++* пројеката. Међутим, такви пројекти не могу да буду самостални (нису *standalone*). Потребно је да их други, циљни (крајњи) пројекат увезе (односно искористи), па се онда целокупан код дељеног пројекта заједно преводи и гради са кодом циљног пројекта.

Као споредни ефекат, један дељени пројекат може да користи кôд другог дељеног пројекта. Ово се ипак не сматра као да један дељени пројекат укључује други, јер заправо дељени пројекти само садрже кôд који другима пружају, и само за њега знају, ништа друго (па ни кôд других пројеката) ни не познају. Ово иако делује контрадикторно, начин на који се разрешава јесте тај да уколико се у неком циљном пројекту укључи само други дељени пројекат који користи први дељени, изградња пројекта неће успети. Уколико неки циљни пројекат жели да користи дати дељени пројекат, мора да укључи и друге дељене од којих коришћени зависи. Ово заправо и има смисла. У системима где постоји више слојева, где је сваки слој засебно написан, а где виши користи нижи, циљни пројекат може да бира, уколико му је само потребан нижи слој, може само њега да укључи. Уколико му је потребан виши слој, такође му је индиректно у том случају потребан и нижи, јер виши зависи од нижег, па тиме мора да укључи оба, иако можда директно не користи нижи, мада га ништа не спречава да у тој ситуацији користи и нижи.

Управо оваква архитектура се користи у овом раду. Пројекат под именом *„Core“* (језгро) представља први, најнижи слој библиотеке где се налазе имплементације најосновнијих градивних елемената овог пројекта. Затим, пројекат под именом *„NeuroEvolution“* представља други слој библиотеке где су имплементирани концепти НЕ, а који на претходно описан начин користи *Core* пројекат. Даље, пројекат именован са *„API“* (Апликативни Програмски Интерфејс) заправо представља пројекат који гради *.dll* датотеку, а у ком се искључиво налази АПИ библиотеке, без икаквог кôда логике. Наравно, ово је пројекат који укључује претходна два, па отуда и његова улога да само излаже функционалности два слоја кроз наведени АПИ. Такође, ова 3 пројекта се дакле заједно граде као један, отуда се генерише само једна *.dll* датотека.

Оваква архитектура такође има још једну веома лепу погодност. Наиме, у овом случају, изграђена библиотека укључује и виши и нижи слој. Речено је да уколико је виши слој потребан, бесмислено је покушати изградњу само са њим, јер се он ослања на нижи. Међутим, уколиок је рецимо само нижи слој потребан, могуће не направити нов циљни пројекат који би укључио само нижи слој, и веома брзо само његов АПИ изложио,

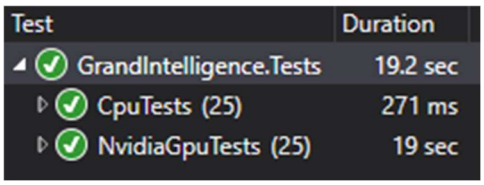
уколико је то рецимо потребно. У оквиру овог рада, није постојала наведена потреба. Такође, два слоја, односно два дељена пројекта не морају бити само укључена у циљне пројекте који граде *.dll* библиотеку, већ по истој особини, могу бити укључене у било какве друге циљне пројекте, у моментима када потреба за таквим настане. Још једна предност је да је у даљем развоју, довољно само у дељеним пројектима кôд изменити, када се рецимо неки делови унапређују или поправљају, а дата промена се затим види кроз све остале пројекте.

Следећи битан пројекат јесте *„NET“* пројекат. Намена овог пројекта је једноставна, као и њена имплементација. Наиме, *.NET* технологије омогућују креирање такозваних *managed .dll* библиотека које се могу користити у *managed* системима који су претходно кратко поменути. Такође, иста технологија поседује могућност позива *unmanaged* кода који управо долази из *unmanaged .dll* библиотека, а претходно описана 3 пројекта управо овакву библиотеку и генеришу. Спајањем ова два концепта, може се креирати *managed .dll* библиотека која у себи садржи позиве ка *unmanaged* библиотеци, што се често назива *wrapper*-ом (омотачем). Постојање омотача није неопходно, али често прелаз *managed* према *unmanaged* представља један процедурални АПИ који је често мање интуитиван и тежи за коришћење, а обично је и добро његово разумевање потребно. Улога омотача је управо да овакав процедурални АПИ „умота“ у објектно оријентисани АПИ који је интуитиван и једноставан за коришћење, а такође се и губи потреба за разумевање детаља који нису од суштинске логичке важности. Као додатна погодност, овакве библиотеке се веома лако укључују и користе у другим *managed .NET* системима и технологијама, попут *Unity3D Technologies* [13], где се рецимо једна примена библиотеке може користити у креирању паметних агената (енгл. *bot*) који су сами способни да играју развијану игру, који могу да представљају противника људским играчима ради забаве и разоноде.

3.1.2. Услужни пројекти

До сада описана 4 пројекта представљају главне пројекте, који су такође и делови главне архитектуре система библиотеке. Ово представља само половину пројеката. Остатак пројеката, иако представљају другу половину, не носе оволику тежину, већ представљају услужне пројекте чија је улога да олакшају развој библиотеке, као и да обезбеде одређени квалитет (енгл. *Quality Assurance*). Пројекат под именом *„Tests“* представља пројекат који је написан у *.NET Unit Testing* технологији, која се бави јединичним тестовима саме *NET* библиотеке. Међутим, како *NET* библиотека представља само омотач око праве библиотеке која је писана у *C++* језику, ови тестови заправо њу индиректно тестирају. Генерално, библиотека је тако пројектована да се само по њеној иницијализацији наводи на ком уређају од подржаних (за сада *CPU* и *NvidiaGPU*) треба да се извршава, а у току рада, о томе крајњи корисник више не треба да води рачуна, већ о томе сама библиотека брине. Стога, корисник може да пише један код, а да га извршава на било ком од подржаних уређаја, налик на принцип *Cross-Platform* технологија. Стога, тестови који су писани су писани тако да не зависе од уређаја на ком се извршавају, а вишеструко се покрећу, и то за сваки подржани уређај, како би се библиотека на свим уређајима тестирала. Између осталог, то је и главна намена библиотеке, да апстрахује све операције како би корисник писао један, што је једноставнији код могућ, а њена улога је да води рачуна о томе где и како се ове операције, алгоритми и сви појмови који се истраживају у овом раду извршавају и одвијају. У овом тренутку, постоји 25 тестова који

пролазе кроз оба слоја, од једноставних до комплекснијих делова, а како за сада постоје два уређаја која су подржана, ово се двоструко на сваком покреће, па се може рећи да постоји 50 (јер се практично тестирају други делови кода, у зависности од самог изабраног уређаја).



Слика 8 – Исход тестова

Интересантно је да је улога неких тестова да провере исправност алгоритама који су стохастичког карактера (у себи садрже генерисање псеудослучајних бројева). Овакви алгоритми се показују као изазовни за тестирање. Један корак ка прављењу ових тестова јесте да се генератор случајних бројева иницијализује константним семеном (енгл. *seed*), мада и то и даље није довољно, јер је секвенца генерисаних бројева и даље непозната и донекле се може назвати случајном (чак и да није само псеудослучајна). Један начин је наравно да се погледа дата секвенца, односно каква ће бити генерисана у тесту. У том моменту, алгоритам постаје познато детерминистички, па се може пројектовати тест који га проверава. Међутим, овакав приступ се испоставља као лош јер у том случају тест губи своју основну поенту. Наиме, поента теста је несамо да провери исправност кода у овом тренутку, него и у будућности, када се рецимо уведе одређена измена у систем (било да се поправља или се унапређује). Тест и у таквим случајевима треба да буде конзистентан. Уколико се, међутим, претходно описани тест на такав начин пројектује, он је валидан само до тренутка када се делови који се тестирају не мењају, или мењају само на поједине начине. Уколико се, рецимо, убаци додатна генерација псеудослучајног броја, читава секвенца се тиме мења, па сам тест више није валидан. Потребно је дакле пројектовати такве тестове који имају одређене критеријуме који се морају испунити да би тест прошао. Рецимо, када се мутација тестира, може се пројектовати тако да се у тесту обавезује да у оквиру генома барем један ген мутира, и да барем један ген не мутира. Постављање константног семена обезбеђује да се при сваком прокретању теста иста секвенца генерише, па је тест детерминистички, односно, увек ће се исте мутације дешавати. Сам критеријум може бити јачи или слабији. Описани критеријум за мутацију је слабијег карактера. Рецимо, може се пројектовати такав тест мутације да захтева да одређени проценат гена у геному мутирају, а исти или различит проценат не мутира (док је за остатак гена небитно шта се деси). Ово је мало јачи критеријум, али се мора водити рачуна да критеријум не постане толико јак да практично деградира у уску повезаност са генерисањем дате секвенце бројева као што је претходно описано. Потребно је дакле направити баланс између ове две крајности. Наравно, најбоље је логовати ове операције у датотеке, а затим директно погледати понашање операција како би се установило да ли оне делују као прихватљиве, или потенцијално садрже неки проблем, али ово престаје да буде аутоматизовани тест. Стога, тестови овог типа на почетку имају једну подешљиву променљиву где се може означити да ли дати тест треба да генерише наведени лог, уколико корисник жели додатно да провери рад алгоритма, након чега ову вредност може да негира како би у будућности само аутоматизовано проверио да ли је критеријум теста испуњен.

Наредни пројекат од услужних у низу је *„cpp-memory-diagnostics“*. Овај пројекат је заправо први пројекат који је насатао од свих. Штавише, он ни не припада директно решењу овог рада, већ је накнадно укључен у решење поред других пројеката, након што је развијен. У питању је још један дељени *C++* пројекат. Наиме, с обзиром да се ради о *unmanaged* домену, програмер је сам одговоран за управљање меморијским ресурсима, а веома честа појава је такозвано цурење меморије (енгл. *memory leak*). Улога овог пројекта јесте да прати коришћење меморије и да обезбеди начин да корисник сазна колико је у сваком тренутку меморије алоцирано, у различитим јединицама мере. Након што се библиотека деиницијализује на крају, и сви коришћени ресурси ослободе, ове вредности треба да падну на нулу. Библиотека садржи неколико позива који се усмеравају ка овом пројекту, који дијагностику враћа кориснику (ово само важи за *Debug* и *TestRelease* режиме рада, док *Release* режим ову функционалност не поседује). Стога, тестови који су претходно описани, након сваког извршавања додатно проверавају да ли постоји негде алоциран део меморије који није деалоциран, односно управо део меморије који је „процурио“. Штавише, ово се може употребљавати и у току рада самог теста, у средини алгоритама, где је услов да не постоји повећање у количини алоциране меморије, односно да је количина коришћене меморије предвиђено константна.

Дијагностика меморије се показала као веома корисан услужни алат, али у принципу уноси додатне временске трошкове (енгл. *overhead*) који у крајњој примени представљају беспотребан трошак јер је у тим ситуацијама дијагностика меморије непотребна. Иако је дакле дијагностика меморије укључена у саму библиотеку и тестове, ови делови кода су санкционисани у *Release* режиму изградње. Међутим, поред *Debug* режима рада, настала је потреба и за *TestRelease* режимом. Наиме, одређени проблеми су се јавили у *Release* режиму када се врше одговарајуће оптимизације кода, па се понапање библиотеке разликовало између *Debug* и *Release* режима. Стога, потребно је ипак библиотеку тестирати на оба режима, али ипак *Release* режим не би требало да садржи дијагностику меморије. Стога је направљен *TestRelease* режим који је идентичан као *Release*, али укључује дијагностику меморије за разлику од *Release* режима и погодан је за тестирање. Наравно, и *Release* режим може да се тестира, али он не укључује дијагностику меморије, али с обзиром да је он по опзимизацијама идентичан као *TestRelease*, довољна је провера на *TestRelease* режиму како би се установило да је и сам *Release* режим у реду.

Преостала два услужна пројекта су *„Debugger“* и *„Examples“*. Наиме, за ефикасан и могућ развој библиотеке, тестови нису довољни с обзиром да се покрећу из *managed* дела. Ту није могуће тражити проблеме (енгл. *debug*), па је направљен један додатни циљни *C++* пројекат, који укључује *„Core“* и *„NeuroEvolution“*, а садржи само делове кода који користе два дељена пројекта одакле би се сама *debugging* сесија покренула и проблем истражио. У овом моменту, пројектована архитектура са дељеним пројектима се поново показала као корисна. Поред описаног *„Debugger“* пројекта, преостаје још *„Examples“* пројекат, који у себи садржи само примере коришћења библиотеке, писан у *.NET Framework C# Console Application* технологији, а уједно и представља неки вид крајњег теста где се показује како би се у неком пројекту библиотека укључила и користила.

3.2. ФУНКЦИОНАЛНОСТ БИБЛИОТЕКЕ

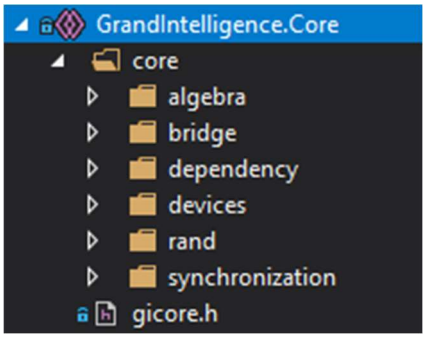
Већ је до сада описано да се библиотека, не рачунајући на *managed* слој који само представља омотач, састоји из два слоја. Први, нижи слој садржи апстракције и имплементације најосновнијих градивних елемената који се помињу у НЕ, а који се могу користити и у друге сврхе, док други, виши слој садржи апстракције и имплементације концепата саме НЕ који се ослањају на нижи слој.

3.2.1. ПРВИ СЛОЈ

Више пута су помињани појмови „најосновнији градивни елементи“ и „концепти нижег слоја“, али нису до сада били дефинисани. Мада, помињани су математички алгебарски појмови тако да се може наслутити о каквим се елементима и ради. Међутим, постоји један генералнији концепт који обједињује све помињане и штавише, за овај рад, представља и довољан концепт за имплементацију првог слоја. То је концепт тензора (енгл. *tensor* [14]).

У математици, тензор је алгебарски објекат који описује (мултилинеарне) релације између скупова алгебарских објеката повезаних са векторским простором. Објекти који тензори могу да мапирају укључују векторе, скаларе, па чак и друге тензоре. Од интереса у овом пројекту је примена тензора, и то из перспективе НЕ. Наиме, тензор се посматра као n-димензиони низ који се састоји од елемената који су у овом случају реални бројеви. Дакле, једнодимензиони низови (вектори), дводимензиони низови (матрице), па чак и скалари, се могу представити тензорима. Један тензор је практично поред самих елемената дефинисан и обликом (енгл. *Shape*), који описује начин повезаности индивидуалних елемената у склопу самог тензора. Облик представља целобројни вектор, где елементи овог вектора указују на дужину i-те димензије тензора, а дужина овог вектора представља хипердимензију тензора.

Иако довољан концепт, тензор ипак није једина ствар која је неопходна за потпуну функционалност првог слоја. На пример, помињано је да библиотека подржава више уређаја на којима се може извршавати. Самим тим, тензори који се праве кроз библиотеку морају бити управљани тако да се креирају на уређају на ком се и сама библиотека „извршава“. Такође, тензори засебно представљају само један пасиван концепт који има способност да чува (меморише) податке, али не и да нешто са њима ради. Потребно је такође дефинисати активне операције које оперишу над тензорима. На слици 9 приказана је структура фолдера у оквиру пројекта *„Core“*.



Слика 9 – Први слој библиотеке

Треба почети од фолдера *„dependency“*. У оквиру овог фолдера, дефинисана је иницијализација и деиницијализација библиотеке где се такође чува и глобални контекст. На неки начин, може се повезати са *Dependency Injection Pattern*-ом. Следећи битан фолдер је *„bridge“*. У оквиру овог фолдера дефинисане су све неопходне апстракције у узорку моста. То су наредне апстракције:

- *Device* – апстракција уређаја у најширем смислу; модел сваког уређаја се у принципу у овој библиотеци састоји из 2 апстракције, а то су алокатор и рачунар;
- *Allocator* – апстракција алокатора; алокатор је систем који на одговарајућем уређају врши управљање меморијом;
- *Computer* – апстракција рачунара; рачунар (у оквиру уређаја) је систем који врши управљање операцијама које се унутар уређаја одвијају;
- *Memory* – апстракција алоцираног дела меморије на неком уређају; меморија се дефинише као низ сукцесивних локација (што укључује и само једну локацију) које могу да чувају информације, а у овом тренутку то могу бити само реални бројеви (за потребе библиотеке други типови података нису ни потребни, мада је остављен простор за унапређење ове одлике, уколико то буде потребно);
- *Operation* – апстракција операције; операција се може дефинисати као активни објекат који се може призвати (енгл. *invoke*, *invokable object*), који се такође може параметризовати како би се његово извршавање споља контролисало;
- *Async* – апстракција асинхроности; наиме, многе операције, па и процеси су у библиотеци асинхроне природе, стога *async* је управо апстракција која омогућује униформан рад са оваквим процесима;
- *OperationGenerator* – за разлику од претходних појмова, генератор операција представља конкретну апстракцију (независну од уређаја), а улога генератора операција јесте да чува методу која зна како се одговарајуће операције генеришу; на пример, могу се генерисати неколико операција множења матрица, а свака се параметризује посебним матрицама (тензорима), па се ове две операције могу у паралели покренути (с обзиром да су операције саме по себи асинхроне); генератор операција треба да зна како да генерише тражене операције како би се корисник опслужио.

Осим генератора операција, као што је већ речено, претходно само представљају апстракције које сада сваки уређај који библиотека подржава треба да имплементира. Кориснику библиотеке се наравно ове апстракције излажу за коришћење али у позадини стоје инстанце конкретног уређаја са којим је сама библиотека иницијализована. Наиме, конкретне имплементације претходних апстракција за *CPU* и *NvidiaGPU* се налазе у фолдеру *„devices“*. У овом фолдеру се поред имплементација налази и иницијализациона *Spec* класа чија је улога да чува подаке који се пре иницијализације библиотеке унесу, а подаци диктирају иницијализацију конкретног уређаја који се бира за коришћење (између осталог у њој се и дефинише који уређај се тражи за коришћење), а инстанца ове класе се кроз претходно поменуту иницијализацију у оквиру *dependency* прослеђује, одакле се иницијализација започиње.

Наредни фолдер, *„rand“*, садржи имплементације генерисања псеудослучајних бројева који се кроз библиотеку користе. Наиме, сваки уређај мора дефинисати неки начин генерисања псеудослучајних бројева у великим количинама. На *CPU* уређајима, ово не представља ништа друго него секвенцијално генерисање бројева редом, док на *NvidiaGPU*, овај посао се може паралелизовати. Интересантно је да је на *NvidiaGPU* могуће библиотеком генерисати преко 16 хиљада псеудослучајних бројева у истом моменту, а уколико је још потребно, наравно овај посао се може итерирати (на процесору се по кораку генерише један број, док се на графичкој картици генерише 16 хиљада по кораку!). Штавише, број генерисаних бројева се може и повећати чак и за више редова, али што је број генерисаних бројева по кораку већи, то је и контекст који графичка картица одржава већи, и самим тим заузима додатну меморију на графичкој картици. Уведен је параметар који практично представља *tradeoff* између ове две појаве, а чија уобичајена (енгл. *default*) вредност је подешена да заузима релативно мало меморије, а да се по кораку генерише довољан број псеудослучајних бројева како би се добило на убрзању у односу на *CPU*. Овај параметар, као и на пример семена генератора псеудослучајних бројева, се могу такође подешавати у оквиту *Spec* класе кроз иницијализацију (уколико се семена не наведу при иницијализацији, библиотека користи сопствени механизам како да постави почетно семе). Поред генератора за сваки уређај, постоји и системски генератор псеудослучајних бројева који се користи у системске и услужне сврхе, а такође се може користити и када је потребно генерисати мали број псеудослучајних бројева, јер у овим ситуацијама генерисање истих на *CPU* представља ефикаснију операцију, па се независно од уређаја овакви бројеви увек могу генерисати, уколико је то и неопходно.

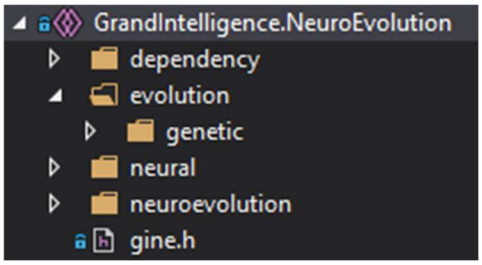
Последњи фолдер коначно користећи до сада описане апстракције имплементира помињану алгебарску структуру тензор, а то је наравно фолдер *„algebra“*. Ту се налазе имплементације тензора и облика. Такође се са слике 9 може видети и фолдер *„synchronization“*, који садржи само услужне синхронизационе алате, као што су семафор (енгл. *semaphore*) и догађај (енгл. *event*). Ови алати помажу са синхронизацијом нити које се покрећу у оквиру библиотеке.

У наставку је дат листинг операција које су дефинисане у овом слоју. Већина операција оперишу над било каквим тензорима (па чак и меморијом где облик меморије није дефинисан), али поједине захтевају тензоре одговарајуће хипердимензије.

- *add*
- *sub*
- *mul*
- *div*
- *assign*
- *neg*
- *scalar*
- *dot*
- *matmul*
- *transpose*
- *randomize*

3.2.2. Други слој

Након имплементације првог слоја, могуће је користећи те имплементације у другом слоју направити апстракције концепата у НЕ, који чак више и не морају да воде рачуна о томе на ком се уређају сама библиотека извршава. Ипак, концепт операције је флексибилан, па се могу правити произвољне нове операције, а у том тренутку међутим треба за сваки уређај дефинисати како се операција извршава, али једино дефиниције операција морају да воде рачуна о уређају на ком се извршавају, све остало је независно. Наравно, након што се ове операције направе, крајњи корисник неће имати потребу да о томе уопште води рачуна (уколико већ и сам не жели да проба нове концепте да имплементира и допринесе библиотеци, или из радозналости да истражи, на пример друге типове мутација или укрштања и слично). На слици 10 се може видети структура фолдера другог слоја, односно пројекта *„NeuroEvolution“*.



Слика 10 – Други слој библиотеке

Као и први слој, и други садржи фолдер *„dependency“* који се бави иницијализацијом. Ове две иницијализације се иначе тим редом позивају. Након тога, наредна 3 фолдера прате концепте НЕ редом, прво ГА и НМ, па тек онда та два спојена у НЕ. Наиме, ГА се заправо налази у оквиру фолдера *„evolution“*, што је остављено због потенцијала да се у неком тренутку уведе други тип еволуције (негенетичка еволутиција), односно неки други алгоритам из фамилије еволутивних алгоритама.

У оквиру *„genetic“* фолдера, постављене су апстракције и неке имплементације концепата генетичке еволуције, а то су већ до сада помињани концепти, као што су ДНК, популација, селекција (избор), репродукција (укрштање) и мутација. Такође, и сам ГА као алгоритам је имплементиран у виду Дарвиновог система. Овде се подразумева имплементација погодна за НЕ, што углавном значи да су гени реални бројеви тензора или групе тензора, али ово је делимично остављено имплементацији конкретних ДНК које изводе из апстрактне дефинисане у овом делу.

Имплементација НМ у фолдеру *„neural“* за разлику од ГА је мало комплекснија. Помињано је већ да се јенда НМ дефинише као вектор слојева од најмање једног елемента (једног слоја – минимум да би мрежа уопште могла да се направи), тако да су слојеви међусобно редом компатибилни. Компатибилност се дефинише тензором који примају и који дају. Наиме, тензори су управо погодне стуктуре за процесирање кроз НМ. Битно је да се два суседна слоја слажу по томе какгов облика треба да буде тензор кога међусобно размењују. У ту сврху, направљена је градитељска класа (енгл. *builder*), која заједно са неколико других услужних класа чини „архитекту“ неуралних мрежа (енгл. *neural network architect*), чија је улога да помогне крајњем кориснику у прављењу мрежа како се корисник не би оптерећивао детаљима који су мање важности.

Такође је могуће правити хиперслојеве, слојеви који у себи садрже друге подслојеве. Ово се међутим у оквиру рада не користи, али представља једну додатну могућност за прављење различитих архитектура неуралних мрежа. До сада је описан само потпуно повезан слој. Он је само један од слојева који су у овом раду имплементирани, а такође је остављена флексибилност за креирање нових, произвољних. Наиме, оно што је интересантно је да се параметри слојева такође могу представити тензорима. Отуда и толика потреба за првим слојем, да се тензор апстрахује и да се у другом слоју не разматра о томе како се тензори чувају, на ком уређају се чувају, како се оперише са њима и слично. Иако то није помињано, и ГА користи тензоре колико и НМ, на пример, када се геном мутира, геном такође може бити тензор који сада сваки један елемент са одређеном вероватноћом мења у неку другу вредност, на неки дефинисан начин.

Такође, у оквиру НМ, имплементиране су и помињане функције активација. Наравно, овде се могу и додати нове, произвољне функције активација, али то не значи да корисник ван библиотеке не може ово да уради. Наиме, функције активација су такође имплементиране као операције, а већ је помињано да су операције флексибилне и да се могу нове произвољне правити, па тиме и корисник може на тај начин правити нове функције активација.

Последњи фолдер, *„neuroevolution“*, представља срж овог слоја. У њему је дефинисана апстракција основног мозга (енгл. *basic brain*), што представља спој ГА и НМ који се даље може користити у НЕ. Ова апстракција познаје начин како мутира, како се укршта, а такође и зна како из генотипа да направи фенотип, а то је неуралнa мрежа (заправо, ово је ситуација где се генотип и фенотип не разликују, па се ради једноставности фенотип одмах може креирати, а генотипске операције се могу изводити над њим). Ова апстракција наравно наслеђује од ДНК апстракције из ГА, па може учествовати у оквиру ГА, док такође садржи НМ коју представља (фенотип), а која се може употребљавати за доношење одлуке у проблему који се решава, односно у проблему или свету у ком НМ актује.

4. Обучавање у неуро-еволуцији

До сада, концепти НЕ су само описно представљени. Теоријски, то је довољно да се направи један НЕ систем. Међутим, то не значи да би се такав систем добро понашао. Штавише, НМ у оваквом систему вероватно не би ни могле да се обучавају на ефективан начин. Потребно је добро дефинисати све операције које се помињу у НЕ, и такође добро одабрати хиперпараметре како би се оптимизација у оквиру еволуције добро одвијала, односно, како би саме НМ заиста училе.

4.1. Пројектовање неуралних мрежа

Прва битна ствар у оквиру НЕ је дефинисати сам проблем, па тиме и НМ која је способна да актује у таквом проблему. Ово подразумева пројектовање одговарајуће структуре НМ. На пример, потребно је донети одлуку о броју слојева, о типовима слојева, о њиховим величинама и активационим функција, о начину интерпретације излазних података, о начину паковања улазних података и њихово достављање самој мрежи. Овим процесом се прави прототипска мрежа која се практично еволуира. То такође представља фенотип у ГА, мада, већ је речено да у овој (основној) варијанти НЕ не постоји суштинска разлика између генотипа и фенотипа, па се тиме уједно и пројектује генотип, а ово представља полазну тачку у даљем дефинисању система како би се постигло ефективно обучавање у НЕ.

4.1.1. Кодовање улазних података

Иако је речено да сваки слој у оквиру неуралне мреже може бити произвољан и произвољно велик, поједине одлуке о пројектовању мреже могу довести до пада или повећања перформанси у смислу учења. Треба ипак овај поступак схватити као избор хиперпараметара, а добар избор може драстично да побољша способност учења (енгл. *learning capability*) саме НМ. Први слој, иако до сада назван скривеним као и други слојеви сем последњег, ипак поседује мало рестриктивнија ограничења у томе како би требала да се пројектују. Заправо, може се рећи да је први слој улазни који прихвата сензорске податке.

Управо то представља ограничење о ком се води реч. Ако се у проблему рецимо појављује 5 битних података, који се представљају као *1D* тензор (вектор), улазни слој би требало да буде такав да управо овакав тензор може да прихвати. Рецимо, то може бити потпуно повезани слој са произвољно много неурона. У овој ситуацији ограничење се толико и не примећује, међутим, неки други слојеви се овде не би уопште могли ни употребити. Уколико је представа улазних података у виду матрице, у том случају потпуно повезани слој се не може употребити.

Постоји и други начин пројектовања улазног слоја, који се сада окреће ка томе да се улазни подаци прилагоде самом улазном слоју. Наиме, из претходно описане ситуације, када се матрица прослеђује мрежи, ова матрица се може развући у *1D* вектор, рецимо ред по ред, или колону по колону, и тиме се прилагоди првом слоју мреже који је рецимо потпуно повезан слој. Међутим, оваква техника развлачења, па и друге технике које прилагођавају улазне податке мрежи могу (али и не морају) потенцијално негативно да утичу на понашање читавог система и на само обучавање НМ.

Поред начина прослеђивања података мрежи, и вредности података су оне које су од велике важности. Наиме, уколико су подаци произвољних магнитуда, ово може веома непогодно да утиче на ток обучавања. Честа техника је да се подаци нормализују у одговарајуће опсеге, међутим та нормализација захтева познавање проблема који се решава, односно начин на који се сами подаци склапају и скупљају, и најчешће НМ, па и улазни слојеви, не залазе у ове детаље, али је нешто о чему итекако треба посветити пажњу.

4.1.2. Интерпретација излазних података

У пројектовању НМ, треба посветити пажњу и излазном слоју. Као и код улазног слоја, и излазни слој садржи поједина „ограничења“. Ова ограничења поново нису директна, већ се намећу као неопходна како би се омогућило ефективно обучавање. Треба поћи од проблема и сагледати шта је то што се у њему дешава, односно, шта је то што НМ треба да управља и/или детектује.

Честа је пракса да се излазни слој пројектује као потпуно повезани. Разлог за ово је тај да потпуно повезани слој има онолико излаза колико и самих неурона у датом слоју, а он управо представља једноставно подешљив хиперпараметар. Ово је битно јер број излаза представља главну концепцију у интерпретирању излазних вредности. На пример, један излаз може да говори у којој вероватноћи мрежа мисли да у том тренутку треба скренути лево. Други излаз може говорити у којој вероватноћи мрежа мисли да је потребно остати у истом правцу. Трећа, аналогно, у којој вероватноћи треба скренути десно. Битна ствар је да се овде налазе три излаза која се интерпретирају на одговарајућ начин, па пројектовањем потпуно повезаног слоја са 3 неурона као излазни слој представља погодан начин да се управо оваква 3 податка добију. Потребно је наравно и направити неки вид логике који би сада од ова три податка донео конкретну једну одлуку, рецимо гледавши која је од ове три вероватноће највећа, па такву одлуку треба и донети.

Треба заправо паралелно размишљати о начину интерпретације излаза и њиховој представи кроз излазни слој. Наиме, за исти проблем, углавном се може дефинисати више, ако не и бесконачно различитих начина кодовања излаза и интерпретација њихових вредности. Међутим, нису све погодне за ефективно обучавање НМ, штавише, обично једна или евентуално неколико дефиниција се могу ефективно употребити.

4.1.3. Број слојева и њихове величине

Још један битан део у пројектовању НМ јесте избор скривених слојева. Наиме, ово је најлабавији део, али и даље битан да се добро пројектује како би се омогућило ефективно обучавање. Први хиперпараметар јесте број скривених слојева. Не постоји генерална правилност у овој одлуци, али се треба придржавати неких мера. Превелике мреже имају способност учења тежих проблема, док мање ову особину немају. Честа метрика у мерењу колико је проблем „тежак“ јесте гледање колико улаза сама мрежа има, односно колико података она прима. Исто тако, број излаза игра сличну улогу. Сама природа оваквих података се исто може укључити у допринос на тежини (рецимо, уколико сви подаци представљају нешто сличне природе, на пример неке дистанце, процесирање таквих би требало да је лакше него да су сви подаци потпуно другачије природе – слично се може извести и за излазе, рецимо, уколико су излазни сигнали исте

природе, односно управљају стварима које су исте природе). Један можда и најзначајнији указатељ на тежину проблема јесте поређење са човековим извршавањем оваквог проблема. Уколико је човеку дати проблем лак, може се претпоставити да је и за НМ (вероватно) лак и обрнуто.

Једна помисао је наравно да се у том случају увек направи мрежа која је много велика (када се ради о броју слојева, у том случају се каже дубока). Ово је потенцијално решење које се често показује као и не тако добро, јер ако је мрежа превише велика за дати проблем, честа ситуација је да се она претерано прилагођава над подацима и/или околином над којом се обучава, што је познатије као *overfit*-овање.

Такође, потребно је и изабрати величине ових слојева (конкретно код потпуно повезаних). Величина слоја практично диктира број параметара у самом слоју, и што је слој већи, и сама мрежа је већа и иста дискусија важи за овај тип хиперпараметра (заправо, ово и јесте део претходне дискусије о величини мрежа). Међутим, уколико су величине слојева мале и/или их има мало, мрежа можда ни нема довољан капацитет да научи оно за шта је намењена (енгл. *learning capacity*), тако да би ипак требало конструисати нешто већу, како би само обучавање уопште и могло да се обавља.

4.2. ДЕФИНИСАЊЕ ЕВОЛУЦИЈЕ

За сам ГА, као и за НМ, потребно је одабрати добре хиперпараметре како би обучавање било ефективно. По својој природи, ГА има велик број хиперпараметара и сваком треба посветити пажњу како би се пројектовао смислен еволутивни систем. То наравно важи и за НЕ системе.

4.2.1. ПОПУЛАЦИЈА И БРОЈ ГЕНЕРАЦИЈА

Често се у ГА првенствено размишља о величини популације, као и кроз колико генерација ће се сам алгоритам одвијати. Помињано је да се доприноси варијацији еволуције што је популација већа. Међутим, тиме се такође и итерација ГА продужава јер сада постоји више индивидуа које треба евалуирати, и које ће учествовати у операцијама стварања наредне генерације. Често се због тога прибегава мањим популацијама, такве да поседују довољно варијације. Мање популације понекад омогућују да се еволуција брже одвија (брже конвергира), јер је и сам пролазак кроз генерације бржи. Наравно, ово никако не значи да треба имати што мање популације. Често се користи величина популације до 1000 индивидуа, а у овом раду, користе се популације од максимално неколико стотина индивидуа.

Број генерација је такође један хиперпараметар, мада је он од мање важности. Наиме, обично се са еволуција стаје оног тренутка када се најбоље решење пронађе. Еволуција заправо може да тече бесконачно. Број генерација се најчешће само уводи из разлога да се ограничи ток алгоритма како се не би десила ситуација да никад не престане. Ово је заправо честа ситуација јер се у многим системима ни не може лако дефинисати или уочити најбоље решење, а исто је још више отежано уколико се користи хеуристичка (апроксимативна) процена решења (некада је само ово и могуће). Насупрот, у неким системима може постојати произвољно много различитих, али једнако најбољих решења, па је тешко покрити логиком свако решење ради изласка из алгоритма. Зато се често ни не прави додатна логика за излазак, али се управо ограничава број генерација.

Споредно, често се и кориснику омогући неки вид праћења система, рецимо путем логова у реалном времену, како би корисник пратио стање и ток еволуције, и у моменту када је задовољан сам и заустави еволуцију.

4.2.2. Избор

Избор у ГА се може вршити на произвољан начин. Међутим, како би се обучавање ефективно изводило, добро одабрани алгоритми избора овоме могу да допринесу. Између осталог, избор сам по себи представља један критеријум еволуције! Као што је већ речено, избор не зависи од конкретног генотипа који се еволуира, стога се може уопштено и дефинисати. Наиме, постоји неколико видова селекција.

Први пример селекције јесте насумична селекција. Бирају се насумично родитељи који ће учествовати у стварању потомства (било читаве наредне генерације или само једног/неколико потомака). Овај метод избора углавном нема проблем губитка варијације (као што неки алгоритми селекције имају), али због своје природе често може бирати решења која би потенцијално требало избацити из еволуције, као и то да можда уопште не одабере она која су добра. Овај тип избора се уопште не разматра у раду, али га је могуће веома једноставно имплементирати.

Други пример избора представља бирање једног дела популације који учествују у репродукцији читаве наредне генерације. Ове индивидуе се не бирају случајно, јер би тада ова метода деградирала у претходну, већ се бирају као најбоље у генерацији (неки вид такмичења). Управо функција способности се користи за ову сврху, и практично овај алгоритам деградира у тражење K максимума из низа (децимација). Након тога, две варијанте су могуће: или да се укрштање врши са $P=K$ родитеља, па да се свих N (величина популације) потомака конструише укрштањем свих одабраних родитеља, или да се укрштање врши са $P<K$ родитеља, одакле се за сваког потомка сада поново на неки начин (рецимо случајан) бирају P родитеља из K-скупа који ће учествовати у том укрштању.

За било какав алгоритам избора, може се увести ограничење да се иста индивидуа не може више од једном (или више од неког броја пута) изабрати. Међутим, ово ограничење се такође и може занемарити. Ове одлуке практично раздвајају алгоритме који се користе, а избор ових алгоритама донекле личи на избор хиперпараметара других делова система. Као и код хиперпараметара, у зависности од проблема, неки алгоритми (хиперпараметри) се боље понашају, док су други не толико добри.

Такође се при формирању наредне генерације може одлучити да се најбоља решења задрже (елитизам). На први поглед, може се чинити да је добро оставити барем најбоље решење из претходне генерације, међутим статистичи се често показује супротно, јер, иако се тада доприноси на бржој конвергенцији, често се заправо долази до конвергенције ка локалном минимуму из код је алгоритму потребно много времена да изађе. Елитизам се може употребити када је потребна бржа конвергенција, а иначе, у општем случају, није погодан.

Пропорционална селекција представља вид селекције где се свакој индивидуи у популацији придружи вероватноћа по којој моће бити одабран. Ова вероватноћа зависи способности саме индивидуе (односно, од вредности функције способности). Што је ова вредност већа, то ће сама вероватноћа избора дате индивидуе бити већа. Формално, ово

се дефинише као нормализација вредности функција способности у односу на суму способности читаве популације.

$$p_i = \frac{f_i}{\sum_{k=1}^{N} f_k}$$

Поново, може се одабрати део популације на овај начин који ће учествовати у стварању читаве наредне генерације, или се овај метод може користити за избор само неколико родитеља који учествују у стварању једног потомка, па се овај поступак може поновити *N* пута. Такође, још једном, одабрани родитељ се може избацити из скупа за избор у наредном кораку како не би био више пута биран, мада ово често представља скупу операцију (не само због избацивања, већ се и поново мора рачунати сума способности преосталих индивидуа), па се најчешће дозвољава поновни избор (између осталог, и у природи, један родитељ може да има више од једног потомка). Нус ефекат је додуше да се у тој ситуацији родитељ може одабрати више пута у истом укрштању, што и не мора да представља лошу карактеристику, али уколико је пожељно, могуће је пратити локални скуп одабраних родитеља како се исти не би више пута одабрао.

Стратегија пропорционалног избора може бити неефикасна када је популација велика. У овом случају, турнир селекција представља добро решење. Наиме, у турнир селекцији се индивидуе такмиче не на нивоу читаве популације, већ на нивоу подскупа. На случајан начин се одабере део популације (*K* индивидуа). Често се користи бинарни турнир, где је ова вредност једнака 2. У овом подскупу се нађе индивидуа са најбољом способношћу, и она се бира за укрштање. Ова индивидуа наравно може да се избаци из популације како се не би поново користила, или се, као и раније, овај део може занемарити. Понављањем се добија *P* продитеља који учествују у укрштању (и поново, или стварају читаву наредну генерацију, или само једног/неколико потомака). Ова метода је једноставнија за имплементацију у односу на пропорционалну и има мање рачунарске захтеве.

У овом раду, с обзиром да се не ради о великим популацијама, користи се пропорционална селекција, мада је турнир селекција један од кандидата који би могли да представљају конкуренцију у мерењу перформанси креираних система у тражењу бољих решења, а самим тим би се и унапредило обучавање у НЕ.

4.2.3. Ток еволуције (ГА)

Конкретан ток ГА је већ описан и приказан на слици 5. У овом раду се користи алгоритам у овом облику, али он никако не представља једини могући ток еволуције. Штавише, одговарајућом променом обучавање у НЕ се може унапредити (уколико је то и потребно). Већ је делимично дотакнута тема о чувању претходних најбољих решења из претходне генерације. Наиме, идеје које се овде рађају је да се, на пример, чувају две генерације и ту међусобно индивидуе такмиче, укрштају, мутирају и праве нове генерације.

Такође, може се креирати и континуални систем где би повремено настајале нове индивидуе, а претходне исто повремено одумирале по неком критеријуму. Међутим, за добар број проблема континуални системи и нису толико примењиви, мада су идеални за симулације које немају дефинисан крај (мада се могу употребити и за оне које имају). У поглављу 7 биће наведене могућности унапређења, где ће о овоме бити више речи.

4.3. ДЕФИНИСАЊЕ ОПЕРАЦИЈА У НЕ

До сада дефинисани појмови су генерално појмови који не морају стриктно бити везани за НЕ, али у овом контексту представљају појмове који се могу третирати као алати помоћу којих се могу градити бољи или гори НЕ системи где се обучавање одвија исто тако. Међутим, постоје и појмови који се стриктно тичу НЕ и само се у оквиру ње могу дефинисати (односно, немогуће их је дефинисати на тај начин ван НЕ контекста). Реч је заправо о операцијама укрштања и мутације у ГА, јер као што је већ поменуто, ове операције су зависне од генотипа, од самог проблема, па се у овом раду у оквиру НЕ само и могу дефинисати.

4.3.1. УКРШТАЊЕ

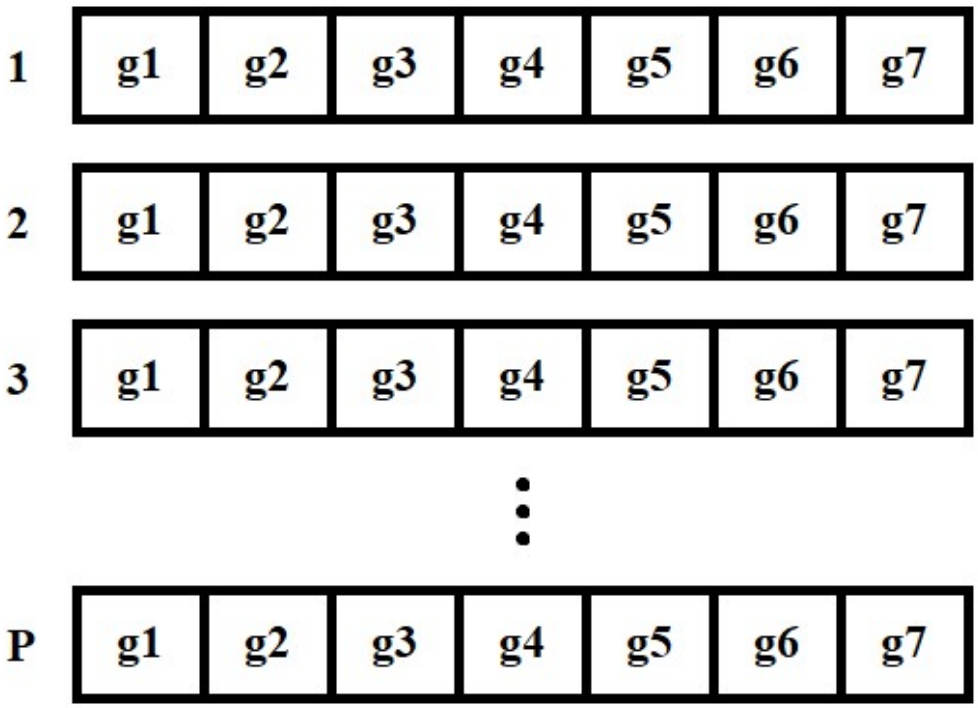
Након селекције родитеља, као што је до сада описано, они учествују у операцији укрштања како би створили, најчешће једног потомка (мада се из једног укрштања може теоријски креирати произвољно много потомака). Како би се постигло ефективно обучавање, потребно је да се добро дефинише ова операција, мада, с обзиром да је и ову операцију могуће разноврсно дефинисати, неке се показују као боље у одређеним проблемима, а неке као мање добре (у смислу колико утичу на ефективност обучавања).

Операција укрштања представља главну операцију која води еволуцију до бољих решења, односно води ка конвергенцији, мада често само локалног решења (уколико се мутација не укључи). Али, то је управо и сврха ове операције, претрага бољих решења комбинацијом претходних. Стога је битна варијација у еволуцији. Када варијације не би било, то би значило да су сва решења у текућој генерацији практично иста или слична, па се и нова решења комбиновањем ових не могу много одвајати од постојећих (иако ово зависи од конкретног проблема).

Потребно је прво сагледати генотип како би се сама операција укрштања могла дефинисати. Већ је неколико пута речено да се у НЕ генотип и фенотип не разликују, а како је фенотип заправо једна неуралне мрежа, предмет ове анализе је управо једна неуралне мрежа. Мрежа се дакле састоји из слојева. Сваки слој, као што је већ помињано, се заправо састоји од одговарајућих параметара (рецимо, тежине конекција, односно појачања и вредности склоности у потпуно повезаним слојевима).

Анализом (која је већ делимично дотакнута) се може установити да се слојеви заправо састоје од одговарајућег броја тензора у којима су смештени параметри датог слоја (који носе одговарајуће „знање“ у појединачном тренутку). Рецимо, потпуно повезани слој се може рашчланити на два тензора. Први тензор представља матрицу конекција (или појачања) $\boldsymbol{W}$ која је описана у поглављу 2.1.2. Други тензор представља вектор склоности $\boldsymbol{b}$ који је такође описан у истом поглављу. Обавезно је потребно уредити ове тензоре, односно поставити их у идентичан поредак, тако да се може рећи да је матрица конекција рецимо први тензор, а вектор склоности други. На овај начин, као што је укратко помињано, добија се поравнање гена. Један ген овде не представља читав тензор (а они су овде поравнати на управо описан начин), већ сваки елемент тензора, што је практично реалан број, а представља вредност параметра самог слоја. Како су тензори уређене структуре, довољно је да се они сами поравнају, јер се тиме директно поравнавају и њихови елементи, јер им се индекси подударају (исти су с обзиром да су мреже исте). Сваки тензор практично можемо развући на вектор, а онда

их редом спајати и на тај начин добити дугачак геном који се на исти начин може једноставно поравнати са другим геномом (имплементационо, развлачење нема потребе да се ради, већ се једноставно итерира кроз све елементе, редом кроз тензоре). С обзиром да се ради о популацији НМ које су исте структуре, обезбеђено је да се увек сви параметри (гени) на овај начин могу поравнати.



Слика 11 – Поравнање гена *P* родитеља

Сада се могу дефинисати оператори укрштања. Наиме, потребно је знати број родитеља, мада се у овом раду пројектују укрштања која раде над произвољно много индивидуа, тако да се сам алгоритам селекције не оптерећује са потребом да има увид у то који алгоритам укрштања је одабран, па тиме може да врши произвољну селекцију (која се такође може и параметризовати, како би се контролисао број родитеља).

Први метод укрштања је *RUC* метод (*Random Uniform Crossover*). Овом методом, за сваки поравнат ген (параметар – са слике 11 то су *g1* до *g7*) од *P* родитеља, са једнаком вероватноћом (дакле *1/P*) бира се један ген (практично из случајног родитеља). Овај ген се на кореспондно место копира у геном потомка. Након проласка кроз све гене, укрштање се завршава и потомак је створен.

Други метод укрштања је *RFC* метод (*Random Full Crossover*). Наиме, овај метод је веома сличан претходном, стим да је разлика у томе да се не гледа сваки ген индивидуално, већ се за групу гена бира један родитељ (на исти начин као код *RUC* укрштања) од ког ће се читава група наследити. Ову групу заправо представља један

тензор. Дакле, читав тензор се копира у исто поравнати тензор потомка (отуда и *Full* у имену), за разлику од *RUC* методе, где се гени посматрају индивидуално.

Ове две методе, иако сличне, заправо поседују различите, па и супротне карактеристике. У окиру *RUC* методе, с обзиром да се сви гени скоро насумично бирају, систем садржи богату варијацију. Међутим, ово често може да доведе до претеране варијације, па обучавање не успева да конвергира уопште. Углавном је погодан у НЕ где су НМ мале, јер са великим НМ, готово да је немогуће задржати тензор који поседује одређено знање (врло брзо се ово знање растури).

Код *RFC* метода, ситуација је обрнута. Читава група гена се бира (дефинисана појединачним тензорима) од једног насумичног родитеља, па се она копира у исто поравнати тензор потомка. Ово на први поглед може да упућује на то да се варијација система врло брзо губи, међутим, ово се само дешава у НЕ системима са малим НМ (тамо је *RUC* метода погоднија). Идеја ове методе је да се тензори који су примили одређено знање не растуре пребрзо.

Трећа метода укрштања је *SEC* метода (*Sequential Even Crossover*). За разлику од претходне две методе, ова метода је детерминистичка. Наиме, детерминистичка укрштања не би требало користити у тренуцима где се кроз једно укрштање прави више потомака, јер би се тиме правила иста решења и варијација би се смањивала. Стохастичка укрштања овај проблем немају, мада се и са њима ретко прави више од једног потомка.

Идеја *SEC* методе је корак даље од *RFC* у односу на *RUC*. Наиме, идеја је да се више узастопних тензора бирају од истог родитеља и копирају у исто поравнате тензоре потомка. Разлог овоме је да потенцијално тензори заједно садрже неко знање које треба очувати, а да појединачно вредности њихових параметара не носе никакве корисне информације. У овој методи се не бира група насумично изабраних тензора, већ се секвенцијално узима првих приближно *Q/P* тензора (*Q* је број тензора у овим НМ – све су исте па и имају исти број тензора) који се копирају из првог родитеља, затим других *Q/*P тензора из другог родитеља, и тако до краја (последњем се даје да копира и остатак тензора уколико *Q* није дељиво са *P*).

Конкретно, код потпуно повезаног слоја, може се рећи да матрица конекција и вектор склоности заједно функционишу. Појединачно је тешко да носе неко корисно знање. Међутим, за друге типове слојева ово и не мора бити тачно, па се отуда ове три технике могу и комбиновати.

Операције укрштања дефинишу централни вид учења (односно обучавања) НМ у НЕ, и практично представљају филогенетички ниво учења. Међутим, као што је већ напоменуто, овакво обучавање би доводило до локално добрих решења, односно, до локалних минимума. Како би се увео потенцијал да се претражују друга решења, односно да се уведе потенцијал за усмеравање ка глобалном оптимуму, или да се учење не би заустављало након што се из околине више ништа ново не може искусити, потребно је увести мутацију као извор додатне случајности.

4.3.2. Мутација

На неколико места је већ помињана потреба за мутацијом, њена улога, као и карактеристике и ефекти који се њом постижу. Мутација се дешава одмах након што се потомак створи укрштањем. Сама по себи, операција је једноставна, и може се реализовати на начин описан у наставку.

С обзиром да се ради о параметрима који су реални бројеви и (углавном) се налазе у опсегу [-1, 1], може се једноставно генерисати нов реалан број у том опсегу, и у потпуности заменити са старим. Наравно, ово се дешава под одређеном вероватноћом. Па тако, на пример, када се сваки ген (параметар) понаособ посматра, првенствено се генерише број у опсегу [0, 1). Уколико је генерисани број испод вероватноће мутације (рецимо 0.1, што је 10%), ген мутира, па се генерише нов реалан број у опсегу [-1, 1], и добијени број се смешта као нова информација, односно нов генетички материјал у посматрани ген (стара се брише). У супротном, уколико је првогенерисани број већи или једнак вероватноћи, ген не мутира, па се стара вредност задржава (практично се ништа не дешава).

У овом раду, користи се веома слична варијанта претходно описане мутације која се назива *RNM* (*Random Normal Mutation*). Наиме, једина разлика је у томе да се, у тренутку када ген мутира, не бира потпуно нова вредност за тај ген, јер ово донекле може да у потпуности промени део знања који је потенцијално битан, већ се уместо тога генерише нов реалан број са Гаусовом (нормалном – отуда и *Normal* у имену) расподелом са средином око 0 и стандардном девијацијом 0.1. Овим се генеришу веома мале вредности око нуле (позитивне и негативне), а добијена вредност се надодаје на стару вредност која је смештена у посматрани ген. Овим, стара информација у гену је само за неку малу вредност промењена (може се направити аналогија са градијентном методом, да се за мали, али случајан градијент променила). Тиме, стара вредност се не мења пуно, али довољно да се ново знање у систем унесе.

Наравно, с обзиром да је у питању случајна вредност, нови генетички материјал потенцијално нарушава знање, али у тој ситуацији, два су могућа исхода: или је тај материјал генерално лош и еволуцијом ће ишчезнути, или је можда само тренутно материјал лош али представља новине које ће водити ка бољим решењима или чак ка глобалном решењу! Наравно, уколико неки делови НЕ система нису добро пројектовани, постоји и трећи исход, а то је да еволуцијом заиста лош генетички материјал неће ишчезнути, али то није проблем мутације, већ као што је и сама хипотеза започета, проблем је до делова НЕ система који нису добро пројектовани!

Од велике важности је овде такође и избор вероватноће мутације. Ако је вероватноћа превелика, генетички материјал ће се у претераној мери мењати, и иако то доприноси варијацији, практично се цела поента укрштања губи јер се већина, или чак сав генетички материјал мења, а како операција укрштања управо и представља механизам тражења бољих решења, систем престаје са конвергенцијом и алгоритам деградира у (донекле) случајну претрагу.

5. СИМУЛАЦИЈЕ И РЕЗУЛТАТИ

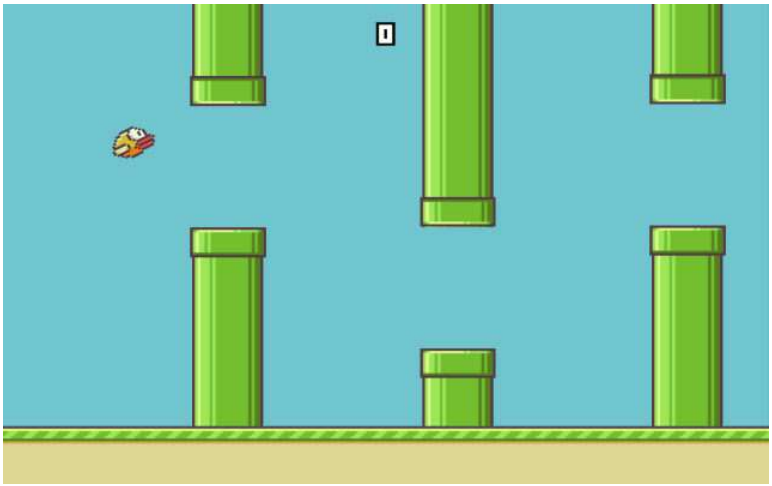
У претходном поглављу су у потпуности дефинисане операције, алгоритми, методе и начини којим се обучавање у НЕ одвија. Сада је могуће овакав систем тестирати и мерити његове перформансе на конкретним проблемима. У овом раду, посматраће се два проблема у УП, а представљају игре у којима је потребно обучити агента да прихватљиво актује у свом свету и да поседује одговарајуће знање.

5.1. *FLAPPY BIRD*

Вијетнамски програмер и уметник за видео игре *Dong Nguyen* је године 2013. развио игру која се назива Лепршава птица (енгл. *Flappy Bird* [15]). Игра представља један веома погодан систем у погледу УП. Наиме, како и човек може да контролише играча (птицу) у овој игри, једноставним избором једне могуће акције (скок), тако би и виртуелни агент ову активност могао да обавља. Најједноставнији агент је такозвани случајни агент, који би у случајним моментима ову акцију извршавао. Наравно, овакав агент би се лоше понашао у датом свету.

5.1.1. ДЕФИНИСАЊЕ СИСТЕМА СВЕТА

Потребно је дефинисати систем у ком се одвијају активности агента (птице) у оквиру света у ком се налази. Играч (птица) у сваком тренутку може донети одлуку о скоку. Ова одлука узрокује да птица мења своју позицију и трајекторију. Тиме, птица пролази кроз низ стања света која је потенцијално воде ка новим деловима, уколико успешно пролази између препрека (цеви), или се судара са препреком, што доводи до краја игре. Играч је наравно бољи што више препрека пређе, односно, што их боље избегава.



Слика 12 – Игра лепршаве птице

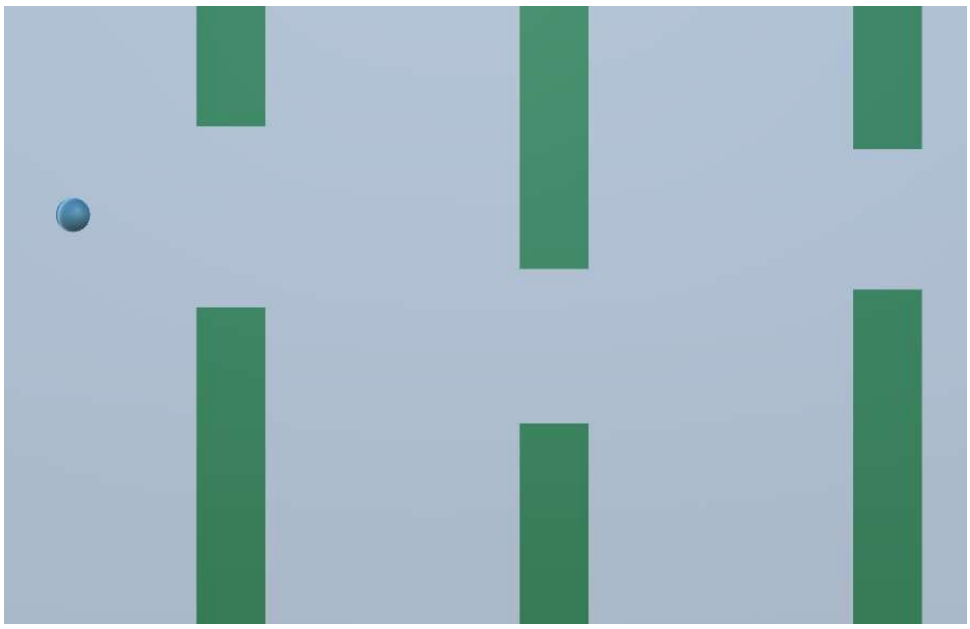
Из перспективе УП, птица добија награду када пређе препреку, и што више препрека пређе, награда је и већа. У оваквом систему управо постоји појава да агент не добија директно награду након доношења одлуке за поједину акцију. Рецимо да агент донесе одлуку да у одређеном моменту скочи, а налази се подоста даље од наредне

препреке. Природно питање је да ли је ова акција добра или није, односно, да ли је донета одлука добра или не. У овом моменту то се не може знати, и неће до тренутка када се примакне препреци. Уколико је рецимо птица била прениско, а скок ју је довео ближе простору куда може проћи, одлука је вероватно била добра. Међутим, уколико је птица била високо, самим тим што је скочила, сада се или налази још више, или још увек не пада, и вероватно неће ни имати довољно времена да физиком игре падне до висине где се налази простор куда је могуће проћи, па ће се ударити у цев и самим тим игра престаје. Стога, претходна одлука о скоку је била лоша. У пракси, ово је познатије као појам одложених последица (енгл. *delayed consequences*).

5.1.2. Издвајање карактеристика

Потребно је проблем сагледати из перспективе битних карактеристика које утичу на доношење одлуке. Ове карактеристике, често назване одликама (енгл. *features*), у општем случају могу бити одабране од стране пројектанта система, или од самог алгоритма учења, ако исти такву особину поседује. Интересантно је да НЕ (која се наравно користи као решење овог проблема), истовремено и поседује и не поседује ову особину. У овом делу, користиће се варијанта НЕ која не поседује ову карактеристику, већ је одговорност пројектанта система да пронађе битне карактеристике. Ово не мора да буде негативна особина. Баш напротив, у овој ситуацији (па и у другим), последице оваквог приступа су да НМ које се пројектују за овакве проблеме могу бити мање.

Идеја је да се НМ не оптерећују потребом издвајања битних карактеристика, јер у таквим ситуацијама морају бити способне да и то науче, односно да могу да науче које су битне одлике, а које не, а након тога и додатно да науче да доносе добре одлуке од битних карактеристика. Самим тим, вероватно је да морају бити веће, како би имале довољан капацитет за учење, а тиме се рачунарски захтеви повећавају и систем се успорава (у овим ситуацијама прибегава се јачем хардверу како би дате захтеве могао да испуни у реалном времену). Мада, овај концепт представља моћан приступ у тренуцима када је веома тешко или чак немогуће издвојити битне одлике.



Слика 13 – Поједностављење света лепршаве птице

Систем се може представити поједностављено, где би се графика система избацила јер је она небитна (дакле представља небитну одлику система). Од главних важности су наравно карактеристике саме птице, њено кретање, а такође и карактеристике самих препрека. На слици 13 може се видети поједностављена варијанта света без графике.

Већ је претходно делимично дискутовано о висини птице. Уколико је птица рецимо изнад празног простора препреке, вероватно не би требало да скочи. У супротном, вероватно би одлука о скоку била добра. Све у свему, може се закључити да је **висина** птице битна одлика система. Како она утиче на одлуку није предмет ове дискусије (издвајања битних одлика). Висина птице се заправо дефинише *y* координатом њеног центра у свету.

Наредна одлика која представља потенцијално битну је слободан простор између препрека куда птица може да прође. Свака препрека је заправо дефинисана помоћу две цеви, доња и горња цев. **Висинске позиције** њихових унутрашњих ивица потпуно дефинишу локацију и величину празног простора. Треба напоменути да се посматра само наредна најближа препрека, а након што се она прође, почиње се са посматрањем следеће, итд. Дакле, агенту није довољно само да зна своју позицију, већ му је потребна и информација о отвореном простору наредне препреке, како би (вероватно) своју висину могао да прилагоди тој висини. Ове позиције се наравно дефинишу као пар *y* координата у свету, где прва представља висину доње ивице горње цеви, а друга представља висину горње ивице доње цеви (редослед је заправо небитна одлика!).

Наравно, још једна битна одлика јесте **даљина наредне** (најближе) препреке од птице. Уколико је ова препрека далеко, у том случају локација њеног слободног простора не представља много битно информацију (али само у таквим ситуацијама), па агент може рецимо да одржава висину негде на средини како не би пао отишао превисоко (ако дотакне „дно“ или „небо“ игра се такође завршава у овој симулацији – од своје *y* позиције би требало да зна где се налази па да ове делове не дотакне). Такође, у тој ситуацији агент можда може да унапред избалансира своју висину што ће му олакшати каснији пролаз. У овим ситуацијама, агент практично развија тактике које му омогућују да касније добија већу награду, а последица је искључиво доброг обучавања. Даљина наредне препреке се може дефинисати као разлика између *x* координата средине наредне препреке (или можда левог краја) и *x* координате центра птице.

Дебљина препреке је такође једна битна одлика. Наиме, ако се *x* координата средине цеви користи у претходном излагању, агенту је потребна и информација о дебљини цеви како би знао где се налази крај леве ивице, како не би закаснио у доношењу одлуке о потенцијално потребном скоку. Међутим, ако се користи *x* координата леве ивице, овим се ништа не постиже, јер је и десна ивица такође битна (рецимо, уколико се агент налази између цеви, не би требало да скочи уколико није у потпуности прошао препреку како не би „закачио“ горњи део, или је у паду и „закачио“ би десну доњу ивицу) Оба пара (*x*-средина, дебљина) и (*x*-лева, дебљина) носе исте битне информације и било који пар од два наведена се може истоветно користити.

Овде се ипак догађа једна интересантна ситуација. Наиме, претходно је дискутовано да је дебљина препреке битна одлика. Особина ове одлике је да је ово један константан податак. Ако се мало боље погледа, све препреке су увек исте дебљине (таква је природа овог света). Константни подаци по својој природи не носе никакву битну

информацију за процесирање, па се самим тим и могу избацити, а модел би требало њих инхерентно да научи. Стога, у овој ситуацији, дебљина препреке ипак не представља битну одлику јер је овај податак константан, али ако би се свет променио и направиле друге препреке које могу имати другачију дебљину, ова одлика би у том тренутку постала итекако битна! Слично резоновање се може употребити и за радијус птице.

До сада, сакупљено је 4 одлике које се испостављају као битне. То су *y* позиција птице, *y* позиције ивица цеви у препреци (дакле две одлике), као и дистанца саме препреке од птице. Сада, можемо направити модел битних одлика и над њим тестирати да ли ове одлике заиста јесу битне. Рецимо да се птица налази мало испод отвореног простора препреке (ово је дефинисано помоћу прве три наведене одлике), а такође се и налази релативно близу препреке (четврта одлика). Питање се поставља, да ли у том тренутку птица треба да скочи? Очигледан одговор на ово питање је „да“. Међутим, ово није тачан одговор. Интересантно је да ни одговор „не“ не представља тачан одговор. Тачан одговор на ово питање не постоји, односно, немогуће је утврдити да ли је потребно скочити или не на основу датих података.

Одговор се крије у томе да још једна битна одлика недостаје. А то је практично да ли је птица можда у неком претходном моменту већ скочила. Уколико јесте, нема потребе да поново скочи, и одговор на претходно питање је „не“. Уколико није претходно скочила, одговор на исто питање је „да“. Заправо, битна одлика овде је њена брзина. Уколико птица има тенденцију да иде на горе, вероватно не би требало поново да скочи. У супротном, вероватно треба. Дакле, ***y* брзина** је такође битна, пета одлика. Слично се може извести и за *x* брзину, како птице, тако и препрека, али како ова два податка представљају константне вредности, губе смисао битности јер по себи не носе никакву корисну информацију.

Сакупљено је дакле 5 битних одлика које представљају модел који носи релевантне информације о доношењу одлука у конкретном проблему. Међутим, ово је само први корак. Ови подаци заправо представљају улазе у НМ, и то се могу спаковати у *1D* тензор (односно вектор) који ће дате информације носити. Битно је дакле да први слој овакве податке може да прихвати, или ако ово није случај, требало би пронаћи други начин како би се ови подаци спаковали (рецимо у неку матрицу?) како би сада улазни слој њих могао да прихвати. У овом раду, за први слој се узима потпуно повезани који може да прихвати описани вектор.

Интересантно је дискутовати о конвенционалном прогамирању и прављењу одређеног алгоритма који би над овим подацима вршио неку обраду и доносио одлуке (уместо коришћења НМ). Испоставља се да су овакви алгоритми веома комплексни, ако уопште, и могући за конструисање. У овом моменту НМ предсатвљају једно веома ефикасно решење проблема, али је њима потребно да кроз неко време науче како да доносе одлуке тако што ће актовати у свету и некако се обучавати (у овом раду управо помоћу НЕ).

Након што се подаци спакују, потребно их је и нормализовати. Наиме, као што је о овоме дискутовано, НЕ не улази у овај детаљ, па ни у само скупљање података. Иначе се скупљање (па и нормализација) може схватити као да постоје одговарајући сензорски елементи, аналогно са сензорским неуронима у биолошком бићу, који дакле скупљају релевантне одлике и након нормализације сигнале који носе ове информације прослеђују нервном систему за обраду. У програмирању, сам вид скупљања ових података и њихова

нормализација може представљати исту аналогију. Нормализација се овде може извршити тако што се за поједине одлике наредне технике примене:

- *y* позиција птице – скалира се на опсег [0, 1] тако што се прво одузме висина најниже тачке дела света где се птица може наћи (па и препреке), а затим се подели са висином овог дела;
- *y* позиције ивица цеви – на исти начин се скалира као и *y* позиција птице;
- *x* дистанца најближе препреке – дели се са дужином дела света где се препреке могу наћи, чиме се такође добијају вредности из опсега [0, 1];
- *y* брзина птице – представља податак који је мало тежи за нормализацију, а у овом раду се једноставно дели са неком предефинисаном вредношћу која једноставно смањује опсег на неки близак [0, 1].

5.1.3. Начин представе одлуке и њена интерпретација

До сада је неколико пута описан свет и начин на који птица, односно агент актује. Дакле, у питању је доношење одлуке да ли треба скочити. Ово се ефикасно представља помоћу вероватноће. Отуда, један начин представљања и тумачења одлуке јесте да постоји једна вредност у опсегу [0, 1] која диктира у којој мери се сматра да у датом тренутку треба скочити, па уколико је ова вредност рецимо већа од 0.5, заиста и извршити скок (одрадити дату акцију). Оваква представа и тумачење (интерпретација) се може имплементирати, али у оваквом систему, постоји и боље решење.

Потребно је боље анализирати који је то скуп акција који агент може да одабере у било ком тренутку (ради једноставности, може се рећи да у сваком тренутку агент има на располагању исти скуп акција, мада, постоји ограничење у оквиру игре да након скока, неки кратак временски период птица не може поново да скочи, па тиме и скуп акција није исти у сваком стању, али се ово може занемарити). Интересантан закључак је да не постоји само једна акција, а то је акција скока, у скупу могућих акција, јер када би ово било тачно, у сваком тренутку агент би могао и морао само да скаче. Друга акција је „акција настављања“, односно, акција нескакања. Дакле, птица може и да одлучи да не скочи у неком тренутку, што се такође сматра акцијом.

Отуда, постоје две различите акције које агент може да уради. Стога се одлука може представити помоћу два броја, оба из опсега [0, 1], такви да оба представљају меру у којој се сматра да је акција везана за њих добра, где први број представља меру колико агент сматра да је скок добра акција у том тренутку, а други да је „нескок“ акција добра у том тренутку, односно да се ништа не предузме. Акција која се бира као интерпретација је она мера која је већа (уколико су исте, може се узети било која акција, али је боље детерминистички увек бирати исту).

5.1.4. Пројектовање неуралне мреже

Из претходног излагања веома једноставно се може пројектовати прототип НМ који ће се користити у еволуцији. Наиме, потребно је да мрежа има 5 улаза, да први слој може ове улазе да прихвати као вектор, и да има два излаза. Најједноставнија оваква мрежа може бити мрежа са једним потпуно повезаним слојем са (обавезно) два неурона. Ово заправо није ништа друго него једнослојна мрежа описана у поглављу 2.1.2 (и практично деградира у две логистичке регресије). Такође, како би се испоштовао

критеријум представе одлуке, потребно је да функције активација овог слоја буде *Sigmoid*. Међутим, у овом раду, ово је само излазни слој, а додат је такође још један слој пре њега, такође потпуно повезан, који има 8 неурона и *ELU* функцију активације. Битно је да први слој буде потпуно повезан јер такав може да прихвати описане улазне податке, а излазни слој треба да има 2 неурона са поменутом функцијом активације.

5.1.5. Ток симулације

У оквиру симулације, прави се популација агената (птица) који ће паралелно актовати несметано у свету. Сваки агент када се сударе са препреком, уклања се из симулације. Када се последњи сударе, прекида се симулација на тренутак, јер се тада популација евалуира и од њих прави нова генерација, а затим се симулација поново на исти начин покреће са новом генерацијом, па на овај начин еволутивни процес напредује. У тренутку судара агента са препреком (у раду се овај тренутак назива „смрт“ агента), агенту се рачуна функција способности.

Слика 14 – Актовање популације агената

5.1.6. Пројектовање еволуције

Потребно је такође дефинисати остале хиперпараметре у еволуцији. Наиме, користи се популација од 200 агената, а почетна генерација је иницијализована НМ које се састоје од тензора са случајним вредностима у опсегу [-1, 1]. Такође, максималан број генерација је ограничен на 1000, мада, како је симулација игре направљена у *Unity 3D Technologies*, често се покретање и заустављање симулације вршило помоћу *Play/Stop* опције у *Unity Editor*-у.

Као избор, користи се пропорционална селекција, где је број родитеља за стварање потомака (укрштање) 1. Овде се може извести интересантна дискусија за начине укрштања. За сва три укрштања (*RUC*, *RFC* и *SEC*), када оперишу са једном индивидуом, понашање им се своди на копирање те индивидуе како би се потомак створио. Стога, небитан је избор самог алгоритма укрштања. Битно је напоменути да се изабрана индивидуа не користи за стварање читаве наредне генерације, већ се за сваког потомка поновно бира на исти начин нов родитељ. Мутација која се користи након „укрштања“ је наравно *RNM* мутација, а вероватноћа мутирања је 1%.

Интересантна дискусија се може извести за функцију способности. Наиме, већ је образложено на који начин она представља сигнал подршке у УП. Потребно ју је наравно моделирати на такав начин да агенти који се добро понашају добију позитивне сигнале подршке (награде), а они који се лоше понашају, добију лоше сигнале (казне). Један интересантан метод који се овде употребљава је дужина живота агента (енгл. *agent lifetime*). С обзиром да се може рећи да агент започиње свој живот настанком саме генерације (када и симулација за ту генерацију започиње), а да свој живот завршава „смрћу“ која је претходно дефинисана тренутком судара са препреком, а како симулација константно тече унапред, они агенти који дуже „живе“, односно чија је разлика тренутка смрти и рађања већа, такви агенти су инхерентно бољи јер су успевали више препрека да пређу. Ово представља апроксимативни приступ у моделирању функције способности. Може се и поставити питање да ли је могуће у потпуности пронаћи праву функцију способности у овом проблему.

5.1.7. Резултати

Описани систем се показао као не тако ефикасан у учењу. Наиме, прва генерација насумично доноси одлуке што је и за очекивати. Међутим, и после 150 генерација, напретка у еволуцији није било. Веома брзо, агенти почну слично да се понашају што може да указује на недостатак варијације у систему. Стога, покушано је да се повећа број родитеља у селекцији на 2, па и неколико, а да се за укрштање користи *RUC* алгоритам, с обзиром да овај алгоритам доприноси варијацији, а такође је и погодан с обзиром да се ради о малим мрежама.

Међутим, напретка и даље није било. Наиме, већина агената и после 100 генерација не успевају да прођу ни прву препреку, а неколицина дође до друге или највише треће. Проблема је било неколико у овом систему, а можда и најбитнији јесте мутација.

Свега 1% мутације се испоставило као јако мало. Агенти уопште не успевају на овај начин да истражују нове акције, а систем је превише ригидан јер се не даје агентима могућност да науче нешто ново, да истраже и искусе нове делове света актујући на другачији начин него како су до сада. Ово објашњава њихово слично понашање, јер се практично врло брзо долази до неког локалног минимума из ког се ни трунку даље не излази, нарочито када се користи само један родитељ, који се практично копира.

Повећањем мутације на 15%, агенти сада почињу да истражују нове делове света тако што ће повремено доносити мало другачије одлуке, а самим тим и другачије актовати. Овим се систем побољшао да ипак након 70 генерација агенти успеју да пређу неколико препрека више, али и даље не успевају да генерализују проблем и науче у потпуности да актују у свету. Истом идејом се може отићи даље, да се мутација повећа

на 25%, међутим, промена више није било, а када се мутација још више повећала, систем постаје хаотичан и агенти почну кроз сваку генерацију да актују скоро насумично, где готово нико не успе ни прву препреку да пређе.

Број родитеља за избор је и раније враћен на 1, а мутација у овом тренутку на 15%. Наредни проблем који је постојао јесте проблем у селекцији. Наиме, с обзиром да се функција способности рачуна као број секунди колико је агент живео, а како рецимо у првој генерацији готово сви агенти живе једнако, иако се неки ипак боље понашају, свима је додељена иста или скоро иста вредност за функцију способности, а у таквом случају, алгоритам пропорционалне селекције деградира у алгоритам униформног случајног избора. Чак и рецимо у ситуацији где једна индивидуа живи секунду више, након што се вероватноће избора срачунају, ова предност се не види уопште, нарочито што инвидивуално гледано, у такмичењу са остатком популације, ова индивидуа се уопште не истиче, већ напротив, има малу вероватноћу избора у односу на заједно остатак популације. А може се рецимо десити ситуација да је ова индивидуа била између цеви у препреци и рецимо малим крајем саме излазне ивице случајно закачила препреку, а све остале индивидуе су биле превише високо или превише ниско неколико тренутака пре него што ће се ударити са препреком. Ове индивидуе су ипак доносиле лоше одлуке, док је индивидуа која је била између цеви у препреци доносила много боље одлуке од свих осталих, а игром случаја, пошто је закачила ивицу, свега једну секунду више живела и ово истицање се у такмичењу касније не види.

Идеја је да, чак и за мале предности, вредност функције способности дате индивидуе треба много да порасте, јер се мора размишљати и о томе да, ако рецимо у популацији имамо 50 индивидуа са једнаким вреднсотима функције способности (а имамо и више!), посматрана индивидуа има 2% шансе да буде изабрана, иако је та расподела униформна, ипак има 98% шансе да не буде одабрана. Ово се постиже тако што се време животног века индивидуе степенује, са бројем 2 или 4 (често и 8). Како ова вредност не би отишла превисоко, број секунди се првенствено подели са 100 (или 60 како би се рецимо гледали минути), а затим се степенује са бројем 4. Након увођења овог побољшања, заједно са мутацијом, систем се драстично мења и еволуција почиње да се манифестује. Овде се такође види и потврда о томе да је критеријуму селекције (па и варијације преко мутације) битан. Наиме, већ након неколико генерација агенти почињу да уче и полако све више препрека да прелазе. Поједина покретања су после 40 генерација довела до појединаца да савршено актују и не греше, али оваква покретања су била веома ретка. Често се дешавало да после неког времена, агенти поново почну слично понашање да показују, као да поново постоји проблем варијације.

Ипак, проблем који је овде постојао је друге природе. У питању је била појава да еволуција упадне у локални минимум. Понашања која су се манифестовала је да се сви агенти у појединим ситуацијама сударају на исту препреку, али по дужини (висини). Оно што се заправо дешавало јесте то да су до тада најбољи агенти успевали да науче најбоље виђено, и ништа друго, боље, па су и остали чланови популације почињали да конвергирају ка њима, јер ништа боље није пронађено. Поново је деловало као да повећање мутације може да реши проблем, или да се избора промени да бира више родитеља, па тиме и укрштање, али још једном ово није давало никакве промене и побољшања, а превеликим повећањем варијације агенти су се поново дестабилизовали и почели хаотично понашати.

Проблем је био следеће природе. Рецимо да постоје два агента која актују, који су се сударили у исту препреку. Међутим, један је рецимо одмах изнад отвореног дела препреке, док је други изнад много више. У таквој ситуацији, оба агента једнако живе и исту вредност функције способности имају, док се ипак агент који је ближи отвореном делу препреке боље понаша. У том моменту, уведена је казна. Агент који је при смрти даље од вертикалне средине отвореног дела препреке, добија већу казну, док онај који је ближи добија мању. Ово је било кључно побољшање за решавање проблема локалног минимума, јер сада, агенти који се не крећу на неки начин ка отвореном делу препреке углавно завршавају далеко од њега па се и њихово понашање може окарактерисати лошим, а казна управо ово моделира у оквиру УП. Казна се рачуна као дистанца средине отвореног простора препреке (*y* координата) и центра агента (такође *y* координата), а која се на исти начин дели прво са бројем 100, а затим диже на степен 4. Након тога, од награде (дужина живота на исти начин трансформисана) се казна одузима, а коначна вредност функције способности се ограничава на неку веома малу реалну вредност (која није нула али блиска нули) како ова разлика не би била негативна (а ни нула, јер нула као вредност функције способности има лоше особине, а у суштини, небитно је да ли неке индивидуе имају негативну, нулту или малу вредност функције способности јер, свакако, те индивидуе се лоше понашају и у пропорционалној селекцији то ће се свакако одразити у веома малу шансу избора која је практично небитна, мада чак и ако буду случајно одабране, ово не представља нужно директно лошу особину).

Сада, са овим побољшањима, у готово сваком покретању, након 20-30 генерација, сигурно се налази барем један агент који се непогрешно понаша у овом свету, а ток еовлуције, односно обучавање у НЕ се може пратити кроз генерације онако како је то до сада спекулисано. Као додатак, направљен је генетарот тешког терена (генератор терена је иначе случајне природе), где се генеришу искључиво наизменично препреке чији су отворени делови веома високо па веома ниско. Овакав терен није био генерисан у оквиру обучавања агената, а испоставља се да су агенти успели да генерализују проблем и да се такође обучен агент на основном терену добро понаша и на новом, тежем (заправо такође безгрешно).

5.2. *STEERING AGENTS*

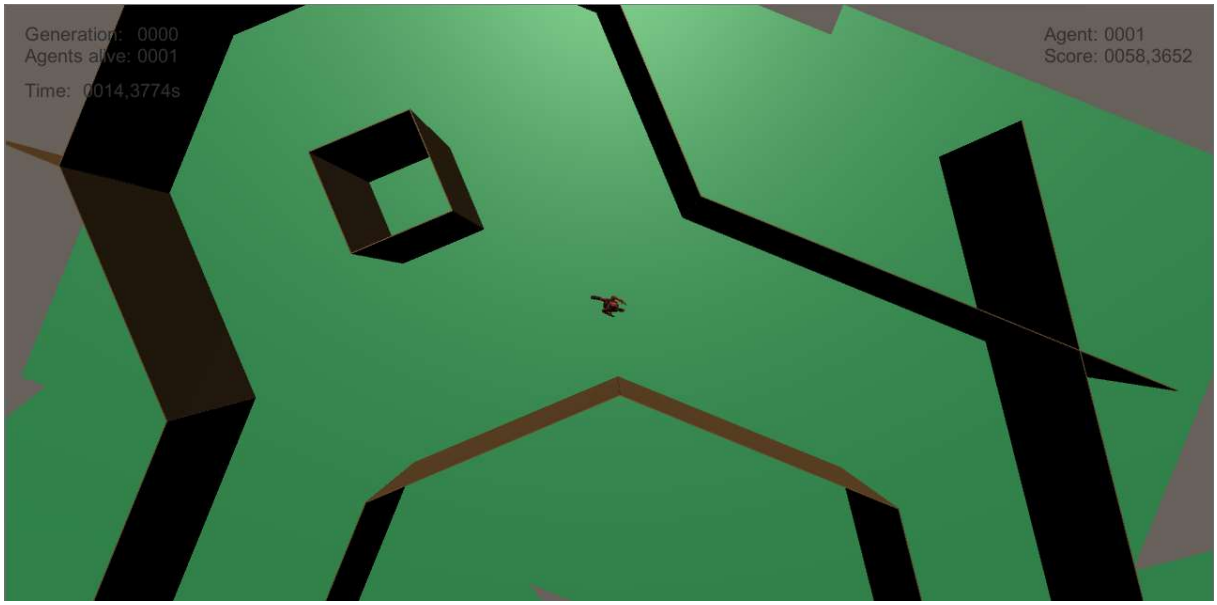
Још један проблем који представља тест библиотеке јесте проблем корманисања или навођења (енгл. *Steer*). Наиме, замислимо да је агент фигура која се налази у неком *3D* простору (за разлику од лепршаве птице која се налази у *2D* свету). Такође, рецимо да се један агент у њему креће без престанка, а да може да наиђе на препреке и границе простора у ком се креће где може да се сударе (као код лепршаве птице). Интересантно би било направити вештачког агента који зна да „корманише“ у оваквом простору и избегава препреке, зидове, и креће се у оваквом простору без грешке.

5.2.1. ДЕФИНИСАЊЕ СИСТЕМА СВЕТА

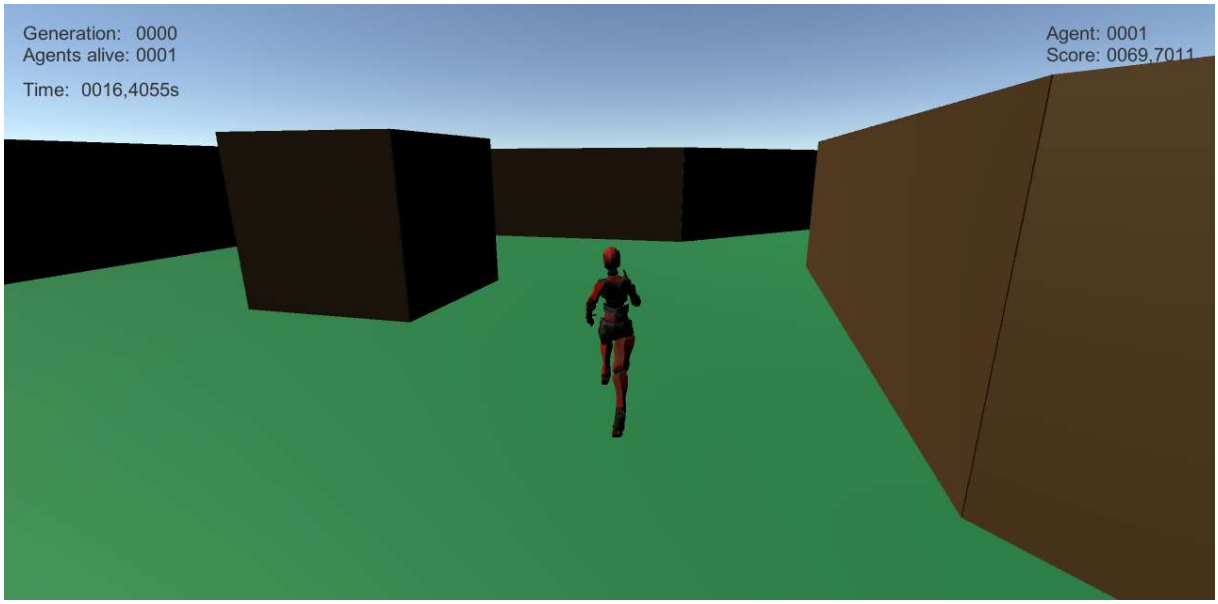
У оквиру овог проблема, свет се састоји од терена који се генерише неодређено дуго. Терен се генерише из насумичних делова (путања) који су представљени у виду ходника различите структуре, који имају улаз и излаз (почетак и крај). Сваки ходник има зидове око себе тако да агенти који трче не могу изаћи из терена. Терен се генерише тако

да се исти заокреће и тиме праве скретнице куда агенти треба да се крећу (трче). Поједини делови терена могу да садрже препреке.

Агенти немају могућност да престану да трче. Постоји одређени ходник одакле започињу трчање, а овакав ходник има затворени улаз (само излаз је отворен). Агенти наравно могу да се заокрећу одговарајућом брзином како би скретали, и то могу радити у сваком моменту. Слично као и код лепршаве птице, када се сударе са зидом или са препреком, завршавају са симулацијом („умиру“).



Слика 15 – Представа света трчећих агената



Слика 16 – Свет из перспективе трећег лица

5.2.2. ИЗДВАЈАЊЕ КАРАКТЕРИСТИКА

У овом проблему, за разлику од лепршаве птице, постоји веома много карактеристика и одлика које се могу издвојити као битне. Додатно, у тренутку креирања других типова генератора терена, нове карактеристике се потенцијално појављују, па тиме старе одабране постају недовољне.

Стога, у овом проблему, приступа се на другачији начин. У оквиру *Unity3D* пакета за развој игара, могуће је унутар сцене радити такозвано „бацање“ зрака (енгл. *raycast*), који када достигне до неког другог физичког објекта има могућност да срачуна (или сазна) дистанцу коју је прешао. Овом техником, могуће је око агента направити „сензорске компоненте“ које агенту достављају информацију о саставу његове непосредне околине. Наиме, у овом раду, агент сваког тренутка „испаљује“ 5 зрака, један усмерен лево од њега, други усмерен десно, један усмерен у правцу кретања, и два усмерена под углом од 45° лево и десно од његовог правца кретања (у истом смеру). Тиме, ови зракови покривају практично лук од 180° испред агента, али наравно не континуално (може се и увести више, али је ово довољно), међутим униформно.

Зраци имају унапред дефинисану максималну дистанцу коју могу да пређу. Наравно, уколико пре ове дистанце ударе (енгл. *hit*) неки физички објекат, њихово очитавање је дакле дистанца агента до тог објекта. Теоријски, уколико ни у шта не ударе, требало би направити неку вредност која ову информацију и носи, али се у овом раду сматра да у тој ситуацији агент прима информацију као да ипак неки објекат постоји на максималној дистанци, а ако је ова дистанца довољно велика, агент не би требало да обраћа пажњу на дати податак.

Стога, максимална дистанца која је уведена је 100 јединица мере (назовимо их метрима). Ради илустрације, на слици 15, ширина ходника у који агент управо улази је свега 5 метара! Одавде се може врло једноставно дефинисати релевантни подаци за доношење одлуке о навигацији агента. Једноставно се сагледају дистанце свих 5 зракова и ово упакује у вектор, на сличан начин као и код лепршаве птице. Интересантна разлика је да у овом проблему, прикупљени подаци представљају податке исте природе (дистанцу), док је код лепршаве птице ово било разнолико (координата односно локација, дистанца, брзина). Нормализација ових података је такође веома једноставна. Све што је потребно урадити је поделити добијене дистанце са максималном дистанцом која је постављена да зраци могу да путују (100).

5.2.3. НАЧИН ПРЕДСТАВЕ ОДЛУКЕ И ЊЕНА ИНТЕРПРЕТАЦИЈА

Већ је поменуто да агенти имају могућност да се заокрећу одговарајућом брзином како би скретали. Ово се дефинише као угао (по природи веома мале вредности) за који може да се заокрене у једном временском периоду (рецимо у оквиру фрејма, или рецимо 60 пута у секунди). Дакле, на почетку сваког таквог временског интервала, може да донесе одлуку да ли жели да задржи правац, или да се за наведени (предефинисани) угао заокрене у левом или десном смеру.

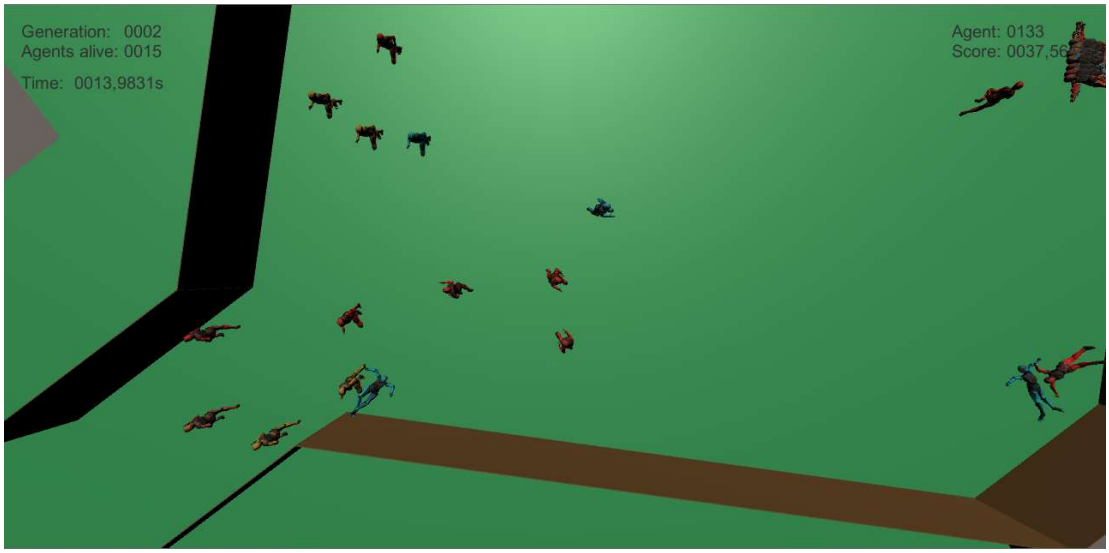
Одавде се дакле добијају 3 потенцијалне акције, које као код лепршаве птице, могу бити представљене вероватноћама, односно мерама колико агент сматра да су у том тренутку такве акције прикладне. Интерпретација се наравно обавља тако што се узме највећа вредност од три добијене и донесе одлука да се обави кореспондна акција.

5.2.4. Пројектовање неуралне мреже

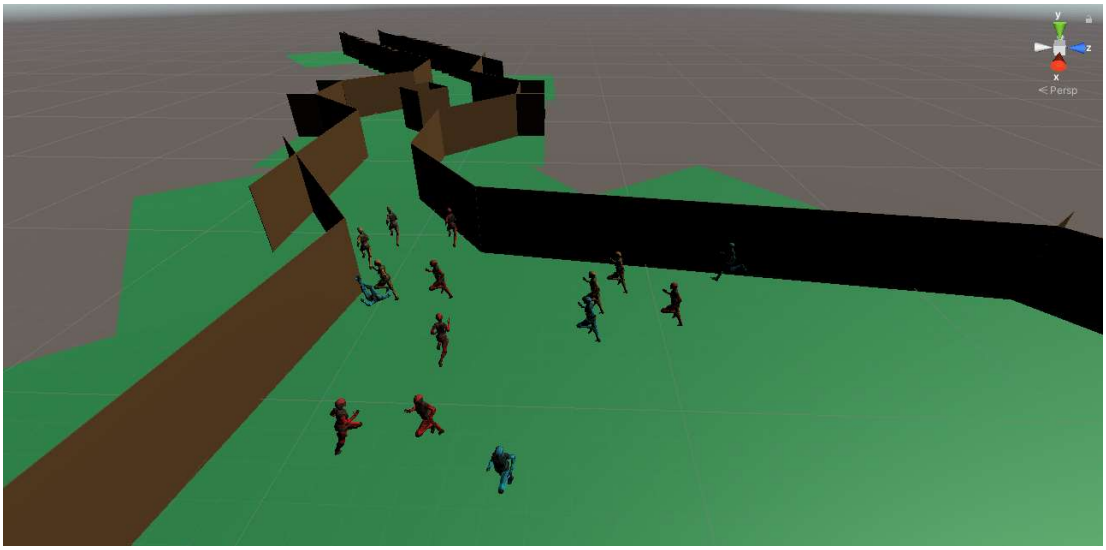
Мрежа се у овом проблему такође једноставно пројектује као и код лепршаве птице. Наиме, захтеви су скоро исти, једино што излазни слој мора да има 3 неурона. Па тако, направљен је прототип мреже која има улазни потпуно повезан слој са 8 неурона и *ELU* активационом функцијом, затим још један потпуно повезан слој са 4 неурона и истом активационом функцијом и на крају излазни слој са 3 неурона и *Sigmoid* активационом функцијом.

5.2.5. Ток симулације

Симулација такође започиње са популацијом агената који практично започињу свој живот и крећу се по терену. Тренутка када се агент сударе са зидом или препреком (и препреке су практично зидови), он и „умире“. Када се последњи агент сударе, симулација стаје и прави се нова генерација за коју се сада симулација отпочиње.

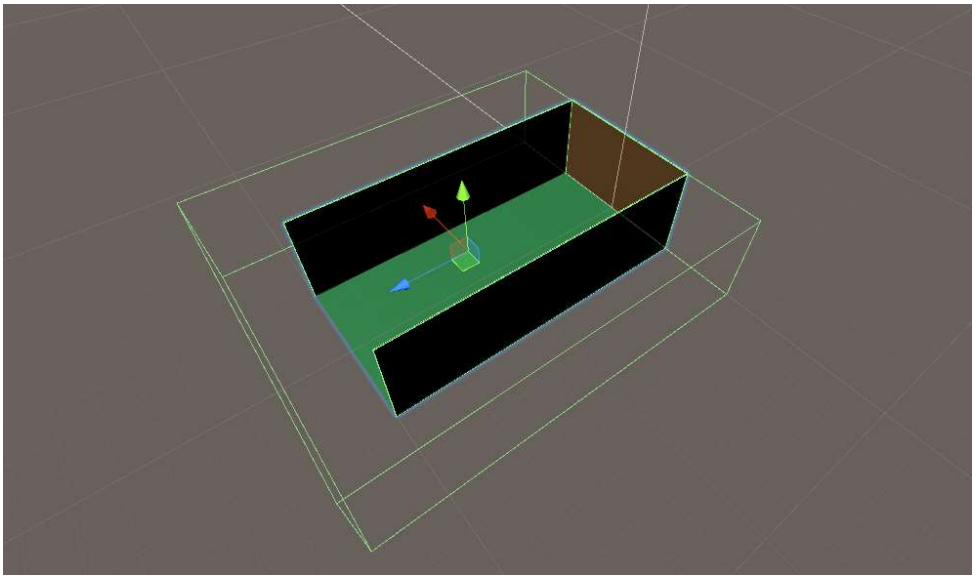


Слика 17 – Кретање популације агената



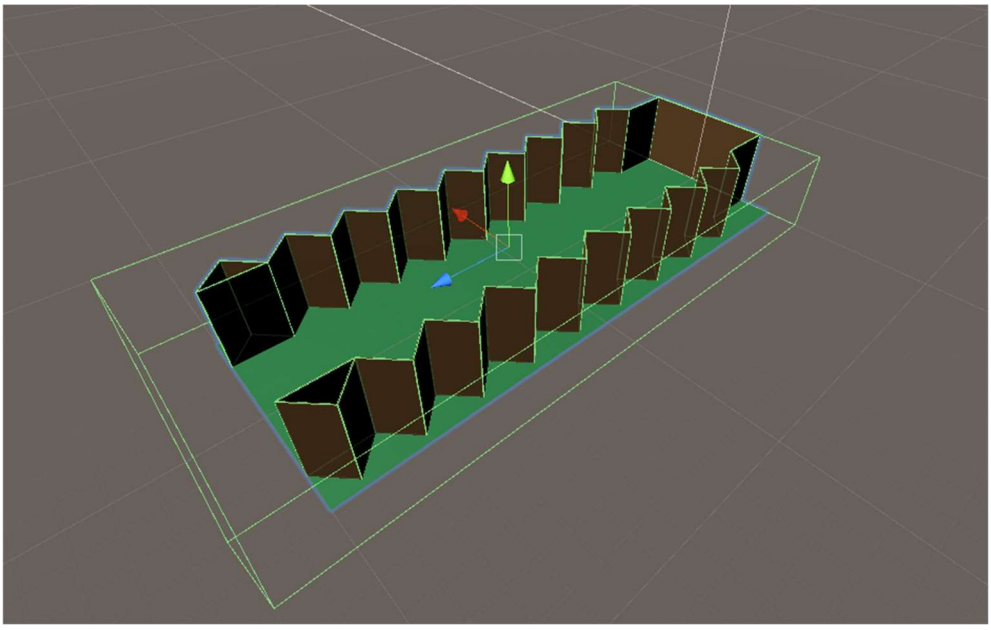
Слика 18 – Изглед сцене

У овом моменту, треба описати и начин како генератор терена функционише. Наиме, постоје 4 типа ходника која се генеришу, од којих је један специјалније природе. Ови ходници се разликују по структури, а сваки такође има и своју „тежину“, односно изазовност коју уводи агентима да овакав ходник пређу. Ходници су такође и параметризовани па се могу помало и контролисати (на пример, њихова дужина или генерално димензија у виду скалирања).

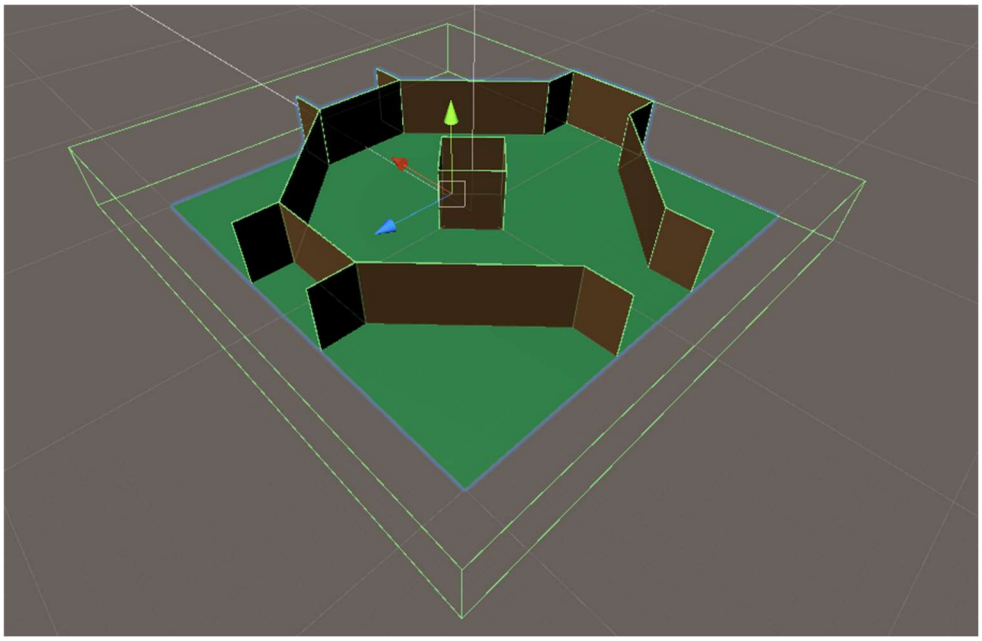


Слика 19 – Прав ходник

Најједноставнији ходник је такозвани прав ходник (енгл. *Straight*) који се може видети на слици 19. Окарактерисан је дужином (његов, као и улаз било ког ходника је затворен ако иза не постоји други ходник који је повезан са њим). Сличан овом терену, јесте цик-цак терен (енгл. *ZigZag*). Такође је окарактерисан дужином а може се видети на слици 20.



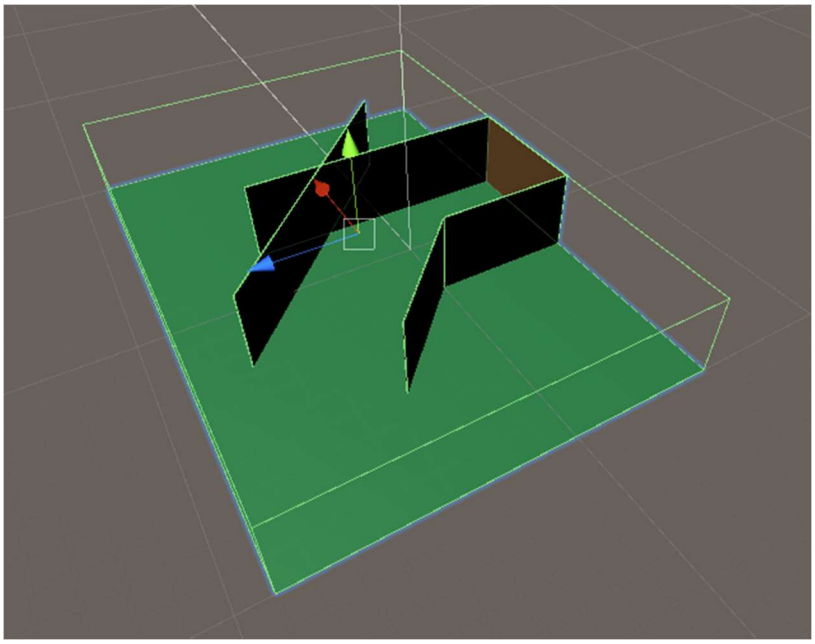
Слика 20 – Цик-Цак ходник



Слика 21 – Раширени ходник

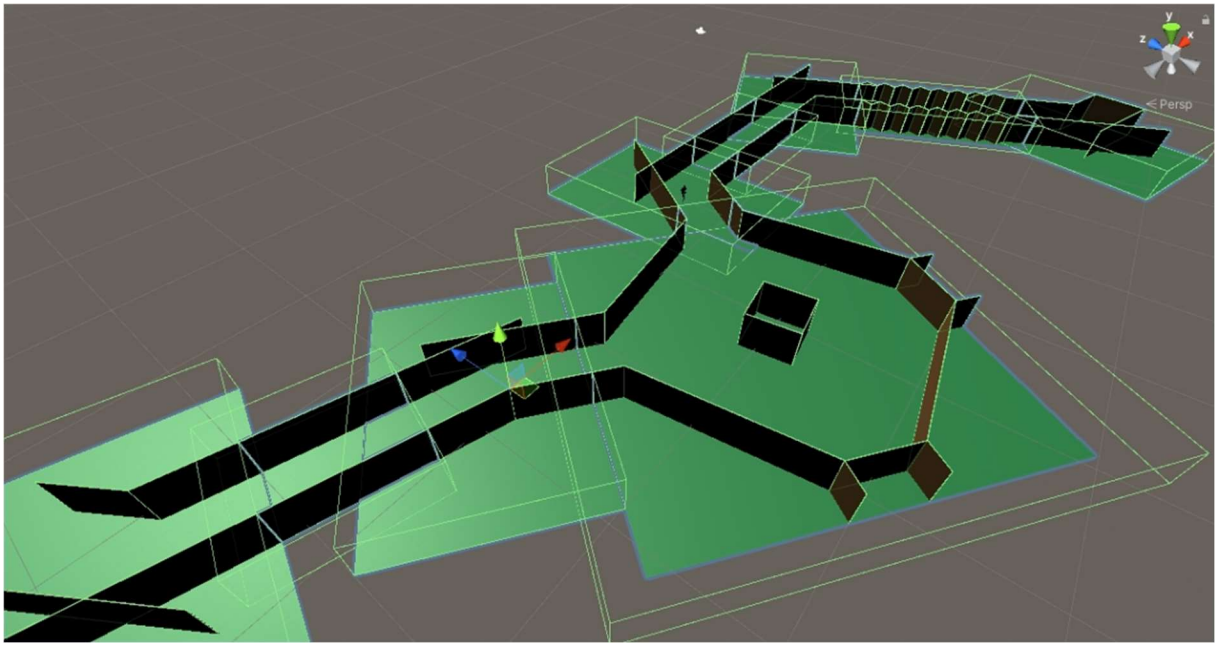
Са слике 21 се може видети раширени ходник. Овај ходник је мало комплекснији. Улаз му је затворен као и код осталих, а има 3 потенцијална излаза, од којих је само један отворен (у овом случају западна капија, односно западни излаз је отворен). Такође је окарактерисан скалом која дефинише његову распрострањеност (ширину), а препреке на његовој средини могу бити присутни, а такође могу бити и искључене.

Појединачно, наведени ходници не могу да граде терен. Потребно их је повезати специјалним ходником који се назива конектор, а приказан је на слици 22. Идеја овог ходника је да за неки угао споји два суседна ходника. Окарактерисан је управо овим углом.



Слика 22 – Конектор ходника

Генератор терена функционише на једноставан начин. Потребно је да изабере полазни ходник (који је заправо конектор коме је угао 0), затим неки насумично други ходник (да није конектор), па затим поново конектор (са неким насумичним углом), па затим поново на случајан начин неки други, итд. Када се одабере неки ходник (укључујући и конектор), његови параметри се такође у неком виду бирају на случајан начин (рецимо дужина). На слици 23 може се видети пример генерисаног терена.



Слика 23 – Генерисани терен

Постоје 3 типа генератора: лак, средњи и тежак. Лак генератор само у обзир узима праве ходнике (и наравно конекторе). Средњи терен (поред конектора) укључује сва три ходника, али тако да широки ходник не може да има велику димензију, не може да има зидове у средини и само северна капија може бити отворена (дакле, не уноси никакво скретање). Тежак терен за праве и цик-цак ходнике дозвољава мање дужине како би се конектори чешће јављали (односно били ближи), а тиме се омогућује да оштра скретања буду ближа (чешћа). Такође, раширени ходник може да има било коју капију отворену, може да има зидове у својој средини, а такође су му и дозвољене велике димензије како би се потенцијално агенти забунили. Терен се затим генерише тако што се испред најдаљег агента увек обезбеди да постоји неколико ходника генерисано, а такође са почетка се бришу ходници када их сви агенти прођу.

5.2.6. Пројектовање еволуције

У оквиру еволуције, прати се популација од 200 индивидуа по генерацији. Такође, број генерација је поново ограничен на 1000, али ово представља небитан податак. Као алгоритам избора се поново узима пропорционална селекција са једним родитељем (алгоритам укрштања је небитан). Вероватноћа мутације је 10%. Што се функције способности тиче, она се може поново дефинисати као нека варијанта животног века (из сличних разлога подигнута на неки степен). Слично је уведена и казна, тако да се посматра дистанца излаза текућег ходника и самог агента.

5.2.7. Резултати

Интересантно је да се у описаном систему никакво учење не одвија, штавише сви агенти који започну свој живот само се у правцу крећу без икаквог заокретања (наравно, сударају се са зидом на првом скретању). Наиме, прва помисао може бити да нема довољно варијације али попут лепршаве птице испоставља се да то није случај. На први поглед, решење које даје агентима награде када доносе одлуке да скрећу (с обзиром да се ради о терену који се на насумичан начин заокреће) може деловати као добра идеја, како не би били пасивни. Међутим, они агенти који вечито скрећу (чак и тамо где не треба) се фаворизују. Такви агенти углавном праве и веће путање и могу бити спорији од агената који покушавају да задрже правац када је то могуће (који би се окарактерисали као биљима, мада не постоји формалан услов да агент мора бити бржи од других, и што је „бржи“ да је то и бољи). У сваком случају, потребно је даље истражити овај систем.

5.2.8. Проблеми, решења и унапређења

Агенти су за почетак били обучавани само на лаком терену. Међутим, као што је већ поменуто, никакво обучавање се није ни дешавало. Разноврсни хиперпараметри су пробани, а у неким тренуцима агенти су и почели да скрећу и актују у свету али то је било веома безуспешно. Мутација је такође повећана где су се промене и могле видети али коначно је усвојено 15% (мада се у различитим ситуацијама мењала, углавном у опсегу од 10-15%). Чак је покушано и да се други улазни подаци мрежи доставе, као и да се излази, односно одлука другачије представи и интерпретира али ништа од наведеног није уродило плодом.

Главни проблем су заправо представљали улазни подаци и њихова нормализација. Наиме, иако је теоријски опсег улазних података заиста [0, 1], ширина правог ходника и конектора је свега 5 метара. Уколико је агент рецимо на средини, очитане вредности левог и десног зрака ће углавном бити око 2.5, што је веома мало у односу на 100 (све да је и 5), а слично се може рећи и за зракове заокренуте за 45°. Једино ће зрак који се креће у правцу трчања агента имати потенцијално нешто веће вредности, али 100 метара је заправо превелика дистанца за коју би агент требало да води рачуна. Стога, максимална дистанца зракова је смањена на 10 (па тиме и нормализација). Раније, са малим вредностима зракова, агент је практично примао информацију да су зидови око њега јако близу и да због оваквих информација НМ не може довољно добро да ради. Сада, након промене, не само да су агенти почели да скрећу и да „живе“, већ се и еволуција могла осетити, као и да се учење кроз генерације могло приметити.

Лак терен су агенти успевали да науче за у просеку 10 генерација, мада у појединим покретањима чак им је било потребно неколико генерација. У једном покретању, агент већ из прве генерације је био способан да се креће по терену заувек! Међутим, за средњи терен, било је потребно око 60 генерација како би успели да у њему науче да актују, а главни изазов им је био управо раширени терен (мада и цик-цак је успевао поједине агенте да забуни).

Након тога, покушано је тренирање агената на тешком терену. Агенти уопште нису успевали да науче да актују нити су показивали неки напредак у томе. Чак и после 200 генерација, нису показивали ефикасност чак и осредњег квалитета. Међутим, интересантна ствар се у овом тренутку десила. Агенти су почели у једном тренутку да

трче у круг, на раширеном терену, који је био поприлично велик (немају могућност враћања натраг јер када се ходник с почетка избрише, улаз наредног се затвара), тако да никад не настављају даље, већ се у њему врте довека. Ово се заправо савршено уклапа са функцијом способности. Наиме, као и код лепршаве птице, што агент дуже живи, то је он способнији. Интересантно је да за разлику од лепршаве птице, у овом проблему, овај начин представе функције способности може да изазове описани споредни ефекат. Агенти су дакле научили да на овај начин могу бесконачно да живе чиме су испунили критеријум који им је дат!

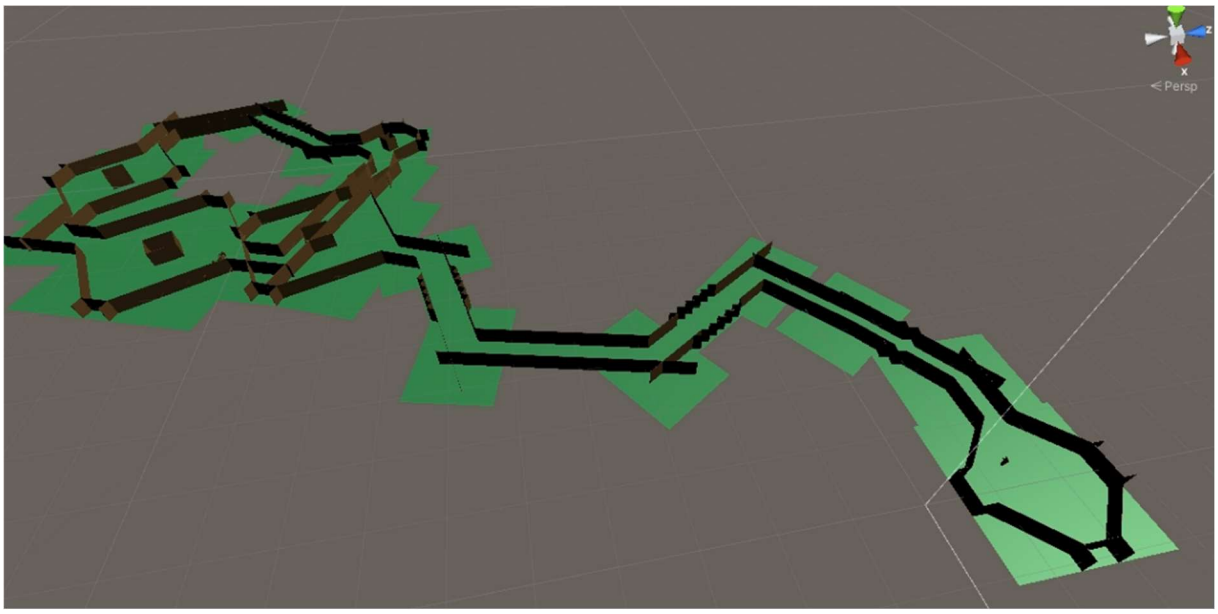
Овде се може још једно питање поставити. Ако су успели на овај начин да „надмудре“ систем, зашто не успевају да науче да доносе одлуке за трчање (односно корманисање), нешто што би по природи требало да је једноставније. Још једна интересантна ствар која се дешавала (и то на средњем терену) јесте да су агенти у одређеним тренуцима, када се нађу у ситуацији да је оштар угао (90°) испред њих и да треба нагло да скрену, исто савршено радили више пута узастопно, а затим у неком каснијем тренутку, идентичну ствар веома лоше изводили (претходно преко 150 агената је ово савршено изводило, а сада, у истој генерацији, дакле исти агенти, свега неколико ходника касније, падају на исту ситуацију, где је само један од њих успео да настави).

Истрагом је откривена веома интересантна ситуација. Наиме, постојао је одређени проблем, где скупљање података зрацима није функционисало како треба. Зраци се заправо сударају са „сударачима“ (енгл. *collider*), који представљају физички објекат (као што су зидови). Међутим, са слике 23, може се видети да зидови нису једини који поседују *collider*-е. Наиме, они се у развоју игара користе за разне сврхе, па око сваког ходника такође постоје велики *collider*-и чиме се прати улазак и излазак агената у исте како би се генератору терена могло јављати о току, присуству и позицијама самих агената. Наравно, ови *collider*-и не би требало да учествују у сударима са зрацима агената, а у развоју игара то је познат концепт као маскирање зрака (енгл. *raycast masking*). Међутим, ово маскирање није било добро подешено и зраци су се поред зидова сударали и са овим *collider*-има, што је представљало велику грешку у самом систему. Тиме, агентима се достављала лажна информација о постојању зидова на местима где их није било.

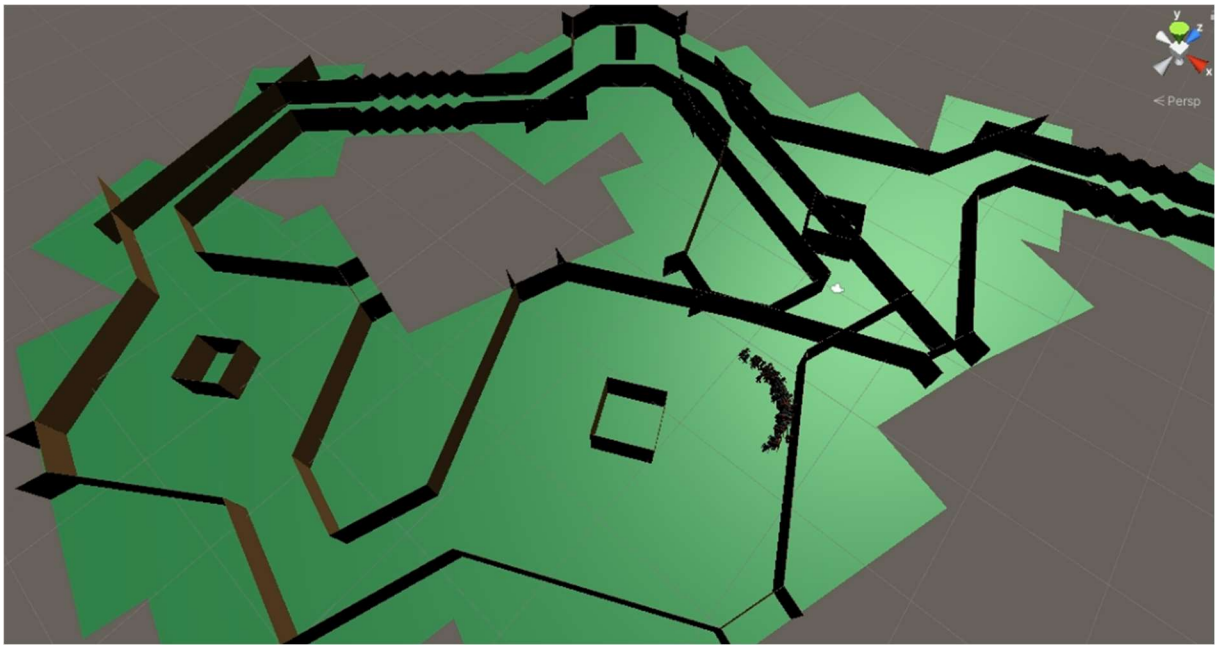
Интересантно је да су агенти ипак успели да на лаком и средњем терену науче да трче и поред ове грешке! Очито, НЕ се у овом моменту показала као веома ефикасна метода обучавања чак и на местима где се неочекиване ствари догађају. Штавише, ово није ни једино што су агенти успевали да постигну у току разних сесија обучавања. Све у свему, након исправка ове грешке, агенти су показивали невероватану способност учења, тако да им је сада у готово сваком покретању било довољно свега 2 или 3 генерације да не само један, већ оквирно три четвртине популације науче да се крећу не само кроз лак, али и кроз средњи терен. Штавише, у неким покретањима, било је да одмах у првој генерацији настане агент који бесконачно трчи и по средњем терену! Поред тога, сада, агенти су успевали да еволуирају и уче и на тешком терену, а потребно им је око 30 генерација да науче у потпуности да актују на истом.

Такође, агенти су успевали да нађу и друге неправилности које су могли да експлоатишу у сопствену корист. Наиме, успевали су да натерају генератор терена да престане са правилним радом, што је било у корист само појединих агената који су то и извели, тако да друге агенте натерају на прерано сударање и престанак живота у

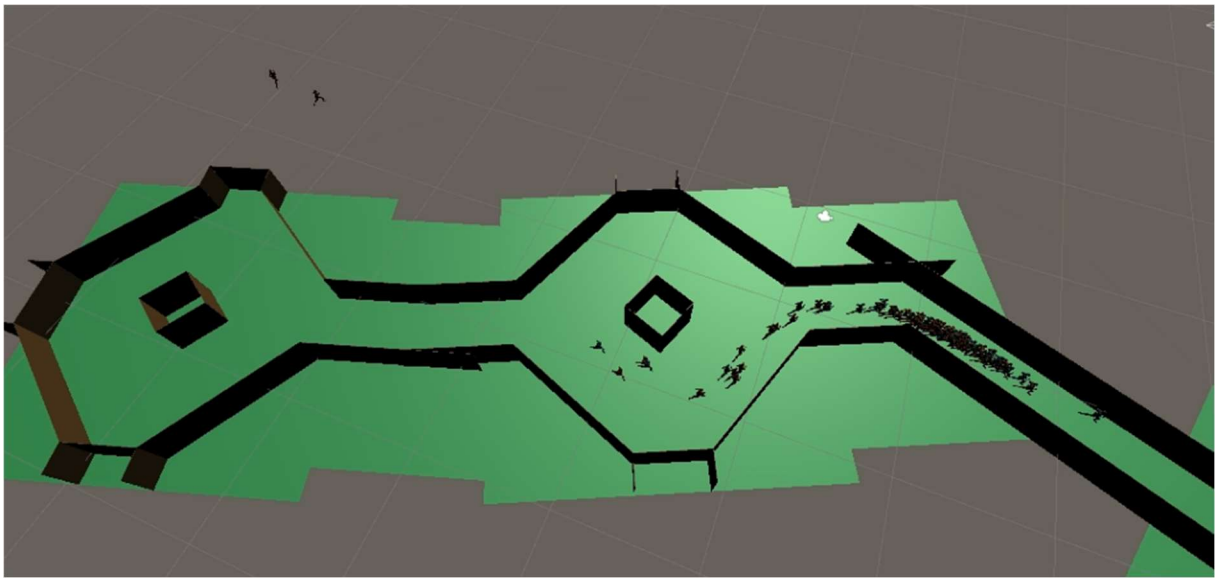
симулацији како би касније они сами били способнији (енгл. *fitter*) у еволуцији јер су дуже живели, па би се у селекцији такви и више бирали. Ово је поново била последица коришћења дужине живота као критеријум за способност, па је уведено бодовање (енгл. *score*) које представља главну меру награде агената (дужина живота се више не користи), а повећава се како они пролазе кроз више ходника (па уколико агенти почну да трче у круг, *score* им се неће повећавати јер не пролазе кроз наредне ходнике). Такође, у неким моментима, агенти су пронашли начин како да изађу ван терена, а како је тло само графика у овој симулацији (оптимизациони разлози), по читавом свету се заправо налази „тврдо копно“ по ком могу да трче, и сада су у могућности да слободно трче по бесконачном свету.



Слика 24 – Престанак исправног рада генератора терена



Слика 25 – Неисправно генерисан терен узрокован акцијама агената



Слика 26 – Приказ два агента који су зашли ван терена

Разне технике су коришћене како би се проблеми који су настајали решавали али се генерално не тичу саме НЕ. Наравно, НЕ је била узрок ових дешавања, дакле показује се као метода која је способна да у УП системима тражи веома добра, па и непредвидива решења.

5.2.9. АНАЛИЗА И ДИСКУСИЈА

Може се увести и одговарајуће објашњење за поједине тренутке где су агенти из прве генерације успели да (барем лак и средњи) терен „науче“. Наиме, иако се ради о случајној генерацији, одговарајућим избором хиперпараметара може се повећавати шанса да случајно генерисан агент одмах поседује довољно добро знање како би се понашао прихватљиво. Наиме, показано је да су улазни подаци веома битни у раду у оваквим системима. До сада је помињано да је проблем вероватно једноставнији уколико ови подаци представљају податке исте или сличне природе. Добрим сензорским (алгоритамским) прикупљањем, као и њиховом нормализацијом долази се до тога да је њихово процесирање једноставније. Тиме се губи и потреба за „јачом“ НМ.

Слично се може рећи и за начин представе одлуке (излаза), као и њено интерпретирање. Ако се ово добро дефинише, поново се смањује потреба за јачим процесирањем и обрадом улазних података јер се на лакши начин овакви излази могу добити од улазних података.

Такође, само пројектовање НМ је од велике важности. Ако се пројектује довољно велика (али не и превелика) мрежа, или заправо нешто већа од довољне, мрежа ће и даље бити адекватна проблему, а постојаће вероватно додатни параметри унутар ње који могу бити редундантне природе, за које је небитна вредност којом се иницијализује, па се тиме повећава шанса за случајно добро генерисаном мрежом, јер било која вредност параметра не утиче на остатак мреже. Овај параметар, међутим, не мора бити у потпуности непотребан, већ може имати улогу „помоћног“ параметра.

Неке од вредности параметара унутар овакве мреже могу да имају и „толеранцију“, односно, могуће је да не утичу превише на доношење одлуке, па за добро

понашање агента, довољно је да њена вредност буде у одређеном опсегу (рецимо од 0.4 до 0.8), што повећава шансу за генерисање доброг почетног случајног параметра.

Поред тога, не постоји само један начин понашања агената који је прихватљив, штавише, постоји мноштво понашања која се могу заиста окарактерисати као једнако добрим. Стога, ово повећава шансу да се генерише неко од ових решења одмах у првој генерацији.

Све до сада поменуто представља само резон како би се објаснило потенцијално могуће генерисање појединог агента на случајан начин, а да се он одмах поседује адекватно знање. Међутим, када се процес генерисања више пута понавља (а ово се и дешава с обзиром да се генерише читава популација за ГА), вероватноћа се још додатно повећава да овакав агент буде и генерисан (не треба цела популација да буде таква, већ је довољно да један буде). Сумирајући све наведено, може се извести лабав закључак да ипак добијање доброг агента у првој генерацији није немогуће. Ово се ипак не дешава за тежак терен што је и донекле рационално.

Свакако, сумирани свеукупни резултати излажу да се НЕ показује као веома ефективна и ефикасна метода са добрим особинама, а облик у ком је до сада описана је само основни облик који се даље може унапређивати како би се идеје могле пренети и на проблеме теже природе. Чак и у овом примеру проблема трчећих агената, који представља једноставан проблем, постоји концепт закаснелих последица који су много израженији у тежим проблемима, где је битно унапред направити неку тактику или метод како би се у неизвесности (енгл. *uncertainty*) оптимално понашало, а НЕ показује потенцијал у овоме, па чак и у овом проблему, јер су на пример, агенти показали развитак тактике код раширених ходника који су широки, када „не виде“ која од капија је отворена по уласку у сам ходник, па рецимо крену да трче пратећи десни или леви зид, чиме ће временом доћи до неког излаза и туда наставити.

6. НАПРЕДНЕ ТЕХНИКЕ У НЕУРО-ЕВОЛУЦИЈИ

До сада описани системи представљају само основну варијанту НЕ. У овом раду се поред ове варијанте истражују и друге могућности које представљају унапређења како ове технике, тако и перформанси посматраних система и истражује се потенцијал за коришћење НЕ у друге сврхе.

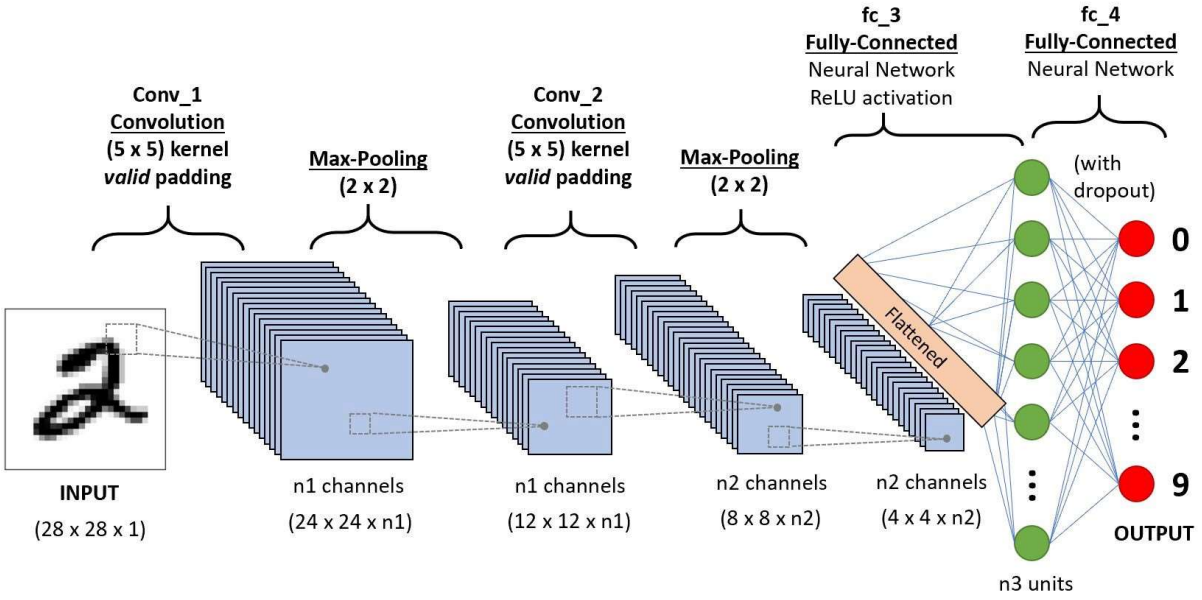
6.1. КОНВОЛУЦИОНЕ НЕУРАЛНЕ МРЕЖЕ

До сада описани типови слојева који су се користили у оквиру НМ јесу потпуно повезани слојеви. Међутим, то наравно нису једини слојеви који се у овом раду користе, нити једини који су имплементирани. У оквиру дубоког учења (енгл. *Deep Learning* [16]), нарочито у области рачунарске визије (енгл. *Computer Vision* [17]) популаран тренд последњих година тежи ка даљем истраживању и већем коришћењу конволуционих неурални мрежа (енгл. *Convolutional Neural Networks* [18]), односно ка коришћењу конволуционих слојева.

6.1.1. ПРИНЦИП РАДА КОНВОЛУЦИОНИХ СЛОЈЕВА

За разлику од потпуно повезаних слојева, главна идеја конволуционих слојева је да се направи такав процесирајући слој који може да одржи просторну структуру улазних података како би се и просторно скривене информације искористиле у учењу. Као што је већ описано, у тренуцима када улазни слој не може да прихвати улазне податке у таквом запису, потребно их је другачије представити. На пример, један од проблема који се често јавља у оквиру рачунарске (машинске) визије је класификација слика. Конкретно, рецимо да се ради о оптичком препознавању карактера (енгл. *Optical Character Recognition - OCR* [19]). Ради једноставности, претпоставимо да се само детектују цифре бројева. У проблему се дакле користи НМ која треба слику (или мали део слике) где се налази карактер (цифра) њу да класификује, како би се на дање овај број дигитално представљао у рачунару на њему природан начин. Овде се јавља проблем уколико се користи потпуно повезан слој као улазни, јер је слика *2D* или чак *3D* структура која се може представити кореспондно *2D* или *3D* тензором, а потпуно повезан слој може да прихвати само *1D* тензор. Један начин како се ово може решити јесте да се слика развије ред по ред или колону по колону и представи као *1D* тензор (вектор), али ово за последицу има губитак просторне структуре улазних података који такође могу да носе неке битне информације у доношењу класификационе одлуке.

Конволуциони слојеви управо претходно описани проблем и решавају. Односно, тако су пројектовани да су веома погодни за обраду слике. Поред тога, и најмање слике када се развежу, сваки пиксел практично представља 1 или чак 3 улаза у неуралну мрежу. Уколико се користи потпуно повезани слој, први слој би за сваки неурон имао огроман број веза (па тиме и параметара) чак и за слике које се данас сматрају изузетно малим. Конволуциони слојеви ово решавају на потпуно другачији начин. На слици 27 може се видети најосновнија архитектура конволуционе мреже за управо претходно описан проблем, односно за класификацију цифара.



Слика 27 – Конволуциона неуралне мрежа

Главна операција у конволуционим неуралним мрежама (прецизније слојевима) јесте операција филтрирања (енгл. *filter*, *kernel*). Филтери клизе по читавом простору улазних података и рачунају вредности појединих елемената активационих мапа у виду скаларног производа, као што индивидуални вештачки неурони то раде (један филтер је донекле аналоган једном неурону у потпуно повезаном слоју, али сада он ради израчунавање над више делова улазних података, а не над свим заједно, и то просторно блиских података). Математички модел се управо и може дефинисати као низ (прецизније матрица) одговарајућих скаларних производа (такође, и овде постоји концепт склоности). Конволуциони слој може да поседује произвољан број филтера, а број филтера диктира дубину излазног тензора, јер сваки филтер производи једну активациону мапу која је дубине 1.

Поред конволуционог слоја, постоје и други слојеви који се налазе у конволуционим мрежама. Често се користе такозвани узоркујући слојеви (енгл. *pooling layers*), чија је улога да њима улазни тензор „узоркују на доле“ (енгл. *downsample*) чиме се смањује број података и самим тим излазни тензор је мањи у односу на улазни. Често се користи *max* узорковање, мада може се користити и *avg*, као и други.

Главни разлог за смањење броја података је сам крај неуралне мреже. Наиме, конволуциони слојеви су веома згодни за прихватање улазних података, као и за обраду података где је просторна повезаност битна (слике), али се веома тешко пројектују корисни конволуциони слојеви такви да им излаз може бити лако интерпретабилан као код досадашњих проблема који су дискутовани. Овакву особину имају потпуно повезани слојеви. Стога се у конволуционим мрежама, након што се слика обради са конволуционим слојевима и слојевима узорковања, излазни тензор шири у једнодимензиону структуру, а затим пропушта кроз неколико мањих потпуно повезаних слојева, који сада могу да изврше рецимо класификацију написане (нацртане) цифре на слици. Када се улазни податак не би сужавао, поново би постојао проблем да сада ови потпуно повезани слојеви имају превише параметара. Како се помоћу узорковања

тензори кроз мрежу смањују, и сами излазни потпуно повезани слојеви не морају бити (и нису) превелики.

Нарано, као што и потпуно повезани слојеви имају активационе функције, тако и сами конволуциони слојеви имају исте. Често се у литератури издваја посебан слој који само представља активациону функцију а поставља се одмах након конволуционог слоја. Ово је суштински исто, а у овом раду се активациона функција не издваја као концептуално посебан слој. Популарно је користити *RELU* активациону функцију кроз конволуционе неуралне мреже.

6.1.2. Имплементација конволуционих мрежа

Конволуциони слојеви представљају практично једну операцију која је веома паралелизабилна (енгл. *parallelizable*). Стога, погодна је за имплементацију на графичким картицама, посебно како се и описује помоћу скаларног производа. Касније, експериментална тестирања су показала да и за мање мреже (а мање мреже су по правилу ефикасније на *CPU* јер немају додатан трошак око позива као операције на *GPU* који ипак нису занемарљиви), паралелна природа ових операција даје изузетно убрзање у коришћењу ових мрежа (нека процена је да би за и мање мреже било потребно тренирање од неколико недеља на *CPU*, док би се исти посао на *GPU* извршио за испод 24 сата). Ово убрзање се још више разликује у тренуцима када мреже расту (на *CPU* раст је практично линеаран, што је мрежа већа, то су и временски захтеви већи, док на *GPU* временски захтеви веома, веома споро расту – ако и уопште у неким моментима).

Слојеви узорковања су такође паралелизабилни и представљају једну веома једноставну операцију. Могу да раде у два режима рада, а то је наравно *max pooling* и *avg pooling*. Међутим, иако и ови слојеви показују велико убрзање у извршавању на *GPU* у односу на *CPU*, овај однос није толико велик колики је код конволуционих слојева, због природе једноставности ове операције.

Иако су ово једини претходно помињани слојеви, у овом раду, имплементиран је још један слој, који се не мора нужно користити само у конволуционим мрежама, већ се може употребити било где, али разлог за његову потребу се крије у раду конволуционих мрежа. Повремено, када се нанижу конволуциони слојеви, може доћи до експлозије вредности процесираних података, јер скаларни производ без ограничења практично сумира производе бројева. Ово се не осети када се посматра само један слој, али се итекако примећује када се и свега неколико конволуционих слојева нанижу. Проблем који настаје након овога је да се, уколико се велике вредности доведу до излазног потпуно повезаног слоја, ови подаци тешко интерпретирају јер се најчешће користе неке варијанте логистичких функција (*Sigmoid*, *Softmax*) за функције активације које у себи садрже експоненцијалне чланове, а како рачунар ипак има ограничену (иако велику) представу бројева, ови чланови се не могу представити и честа појава у програмским језицима је да се уместо тога такви бројеви представе „бесконачном“ вредношћу. Ово је лоше с обзиром да бесконачне вредности не носе никакву корисну информацију, а такође, често се у таквим ситуацијама врло брзо одговарајућом аритметиком добијају и не-бројеви (енгл. *NaN*, *Not a Number*), који су још мање корисни од бесконачних вредности.

Додатно имплементирани слој је такозваи адаптирајући слој (енгл. *Adapting Layer*), чија је улога да податке (на заправо неколико начина) адаптира како би даљи

слојеви овакве податке могли несметано да користе. Прва и основна адаптација која може да се примени је нормализација. Ова нормализација је мало другачије природе од нормализације улазних података. Наиме, тешко се овде може и дефинисати какви су подаци, односно која им је природа, како би се рецимо знао њихов опсег. Практично, овде се нормализација врши над произвољним подацима, а једна од техника која се у оваквом општем случају може употребити је техника статистичког рецентрирања и рескалирања дистрибуције података. Формално математички дефинише се тако што се од сваког податка првенствено одузме њихова просечна вредност (енгл. *mean*), а затим се подели са стандардном девијацијом ових података.

$$n(x_i) = \frac{x_i - mean(\boldsymbol{x})}{dev(\boldsymbol{x})}$$

Нормализација података није једини вид адаптације коју сам адаптирајући слој може да изведе. Наиме, други вид адаптације је преобличавање (енгл. *Reshaping*). У тренутку када се прелази из конволуционог домена у домен потпуно повезаних слојева, речено је да је неопходно развући податке како би сам потпуно повезан слој успео да их прихвати. Управо је ово други вид адаптације коју може адаптирајући слој да изведе. У општем случају, било какав вид промене облика тензора је дозвољен, па и *1D* на *2D* или *3D* трансформације су дозвољене, докле год је те тежина (енгл. *weight*) облика одржана (нпр. могуће је *1D* тензор дужине *27* трансформисати у *3x3x3 3D* тензор, али рецимо *2D* тензор димензије *7x4* није могуће трансформисати у *3D* тензор димензије *3x3x3*, као и обрнуто).

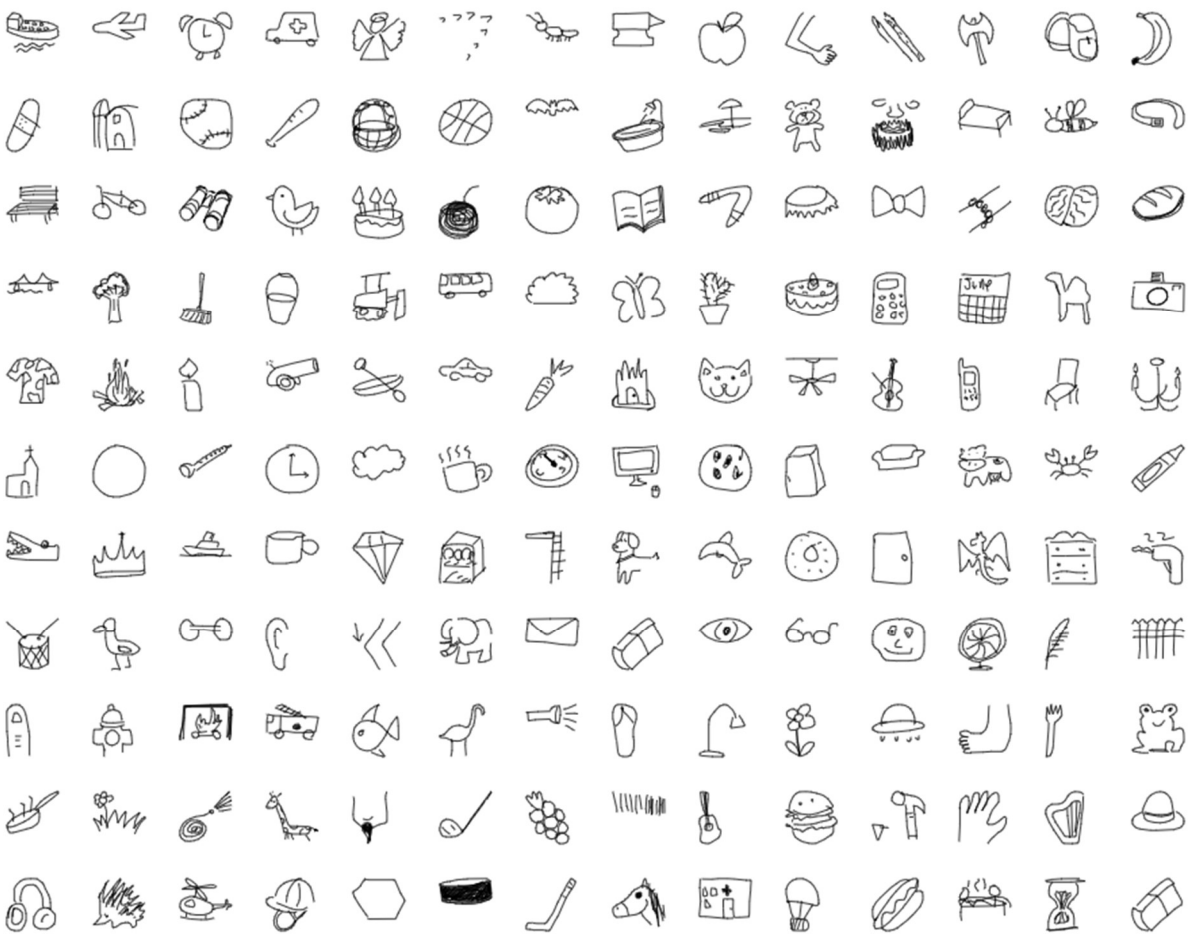
Још једна додатна функционалност уграђена у адаптирајуће слојеве је активациона функција. Заправо, узоркујући слојеви, па и адаптирајући, немају потребу за функцијама активације. Међутим, адаптирајућем слоју се ипак оставља могућност да врши активацију преко активационе функције, а разлог је следећи. Уколико кориснику не треба активациона функција, може употребити *LTU* у облику чисте линеарне функције (енгл. *purelin*), која је иначе и постављена овом слоју као основна (енгл. *default*) функција активације, која заправо не уноси никакву промену у подацима. Ипак, корисник овај слој може да користи на произвољан начин, рецимо уколико негде ипак жели да унесе додатну нелинеарност. Све 3 описане функционалности (нормализација, преобличавање и активација) су независне и може се, при креирању неуралне мреже када се адаптирајући слој додаје, бирати да све или само поједине функционалности буду присутне (може и ни једна). Па тако, корисник може изабрати да само жели податке да пропусти кроз неку активациону функцију. Или, може само да изврши нормализацију. Уколико тако жели, може и нормализацију и активацију, или све три (било која комбинација је дозвољена).

Сада, када поједини слојеви постоје, конволуционе мреже се креирају на исти начин помоћу градитеља неуралних мрежа као и потпуно повезане мреже, а наравно као додатне опције градитеља, могу се бирати претходно описани слојеви. За разлику од потпуно повезаних слојева, конволуциони слојеви заправо имају више параметара (тачније хиперпараметара) који се могу бирати, а такође постоје и одређена ограничења јер поједине комбинације вредности ових параметара немају смисла, па тиме и нису дозвољена.

6.1.3. НАДГЛЕДАНО УЧЕЊЕ

Иако је главна тема овог рада учење са подршком, то је само један вид машинског учења. Најраспрострањенији вид учења је заправо надгледано учење (енгл. *Supervised Learning* [20]). Овде, систем се састоји од модела који има способност учења и података који су лабелирани (енгл. *labeled*), а сам модел (односно алгоритам) треба да из датих података научи (генерализује) проблем и самим тим добија могућност распознавања наредних података који надолазе (рецимо, има способност класификације наредних слика на којима су написане цифре на различите, потенцијално до сада невиђене начине).

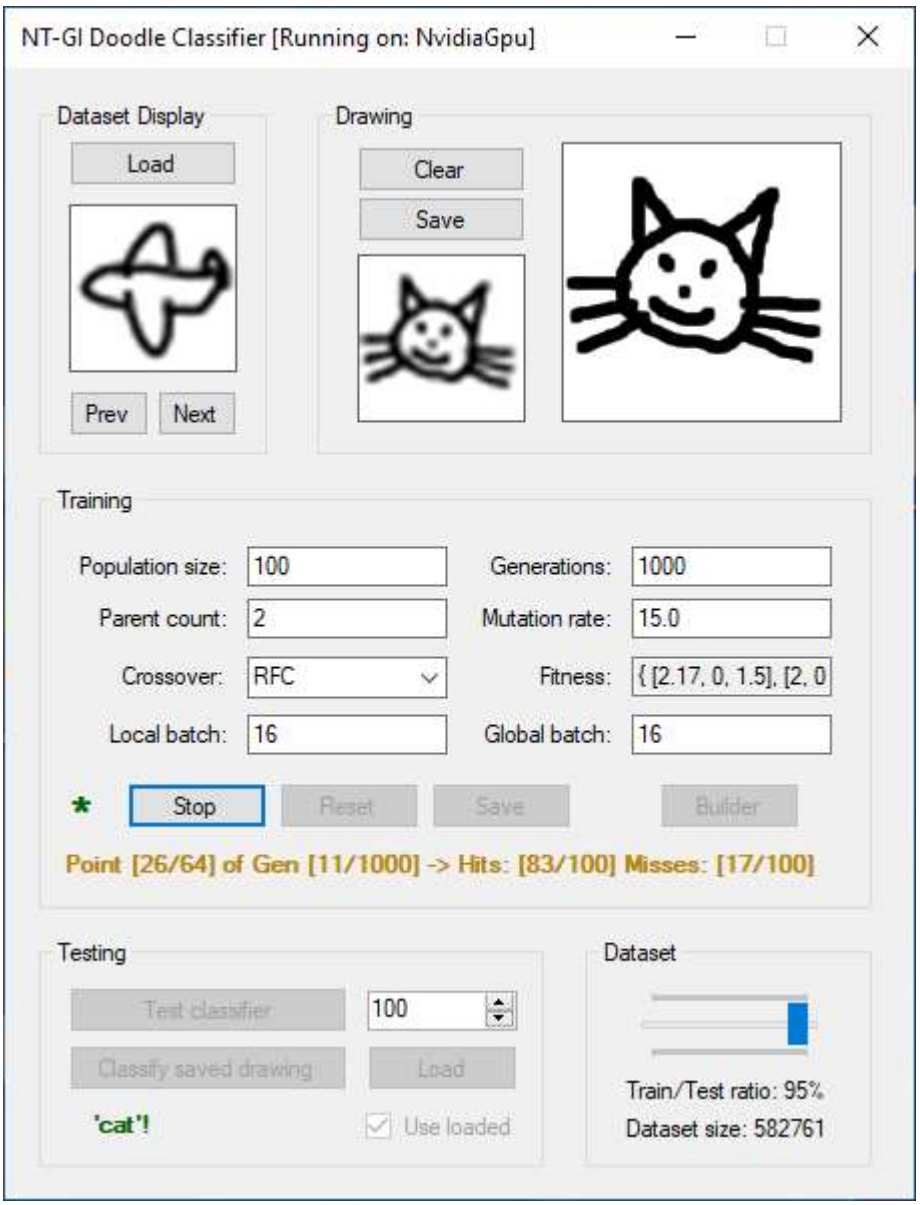
Као тест конволуционих мрежа, као и додатни тест за НЕ, у овом раду направљен је још један пројекат који сада представља систем надгледаног учења где се НМ обучавају помоћу НЕ. Идеја је да се покаже да се НЕ може користити и у оваквим сценаријима (у надгледаном учењу). Као проблем, изабран је *Google QuickDraw Dataset* [21], где би се направио класификатор малих слика сиве скале (енгл. *grayscale*), димензије *28x28*. У питању су мали цртежи из 345 различитих категорија (мачке, пси, јабуке, банане, анђели, змајеви, дијаманти, круне, аутомобили, …). Слике цртају људи из разних делова света, а подаци су погодни за примере машинског учења.



Слика 28 – Цртежи у *Google QuickDraw* скупу података

Потребно је прво дефинисати начин како би систем функционисао. Срж идеје је да се за функцију способности у ГА користи нека функција грешке у надгледаном учењу

(енгл. *cost function*). На делу скупа података који се користе за обучавање у надгледаном учењу (енгл. *train set*) перформансе читаве популације се евалуирају, али појединачно за сваку индивидуу. Рачуна се функција грешке (рецимо, *MSE*, *RMSE* у регресији у општем случају или тачност, прецизност или ф-мера у класификацији) како би се перформансе дате индивидуе исказале преко вредности која се сада у НЕ може користити као способност која се даље користи у еволуцији на исти начин како је претходно и разматрано (нпр. у пропорционалној селекцији за рачунање вероватноће избора).



Слика 29 – Алат за класификацију цртежа

Са слике 29 се може видети развијени алат за класификацију описаног скупа података. Преко њега, могуће је помоћу *Dataset Display* дела истраживати овај скуп. Такође, развијен је и мали алат који омогућује цртање нових цртежа оваквог типа (*Drawing* део). Могуће је сачувати цртеж и накнадно користити обучене моделе да изврше класификацију као што се може видети у *Testing* делу (раније учитан модел је извршио исправну класификацију нацртане слике, јер се на слици 29 такође може видети

и да је у том тренутку у току трајало обучавање новог модела, односно тренинг сесија је била у току).

Након тренирања, алат поседује могућност да на скупу података за тестирање изврши евалуацију и резултат прикаже у виду матрице конфузије (енгл. *confusion matrix*). На слици 30 може се видети резулаз мреже трениране свега сат времена овом методом. Треба наравно напоменути да су се трчећи агенти и агенти у лепршавој птици обучавани кроз неколико недеља, раширено кроз велик број сесија, мада је већи део тог времена потрошен у сврху тражења добрих хиперпараметара, а слика 30 представља прве резултате добијене практично насумичним хиперпараметрима. Такође, у коришћеном скупу података постоје подаци који нису валидни, чак потенцијално и неприкладни, а сами се не могу користити у исправној класификацији чак и за евалуацију од стране човека. Тиме, није ни потребно направити модел који ће успевати да сваки податак у скупу за тренирање, па и тестирање, класификује „исправно“.

NT-GI Doodle Classifier Evaluation <TRAINED> [Hits: 426/600 | Misses: 174/600]

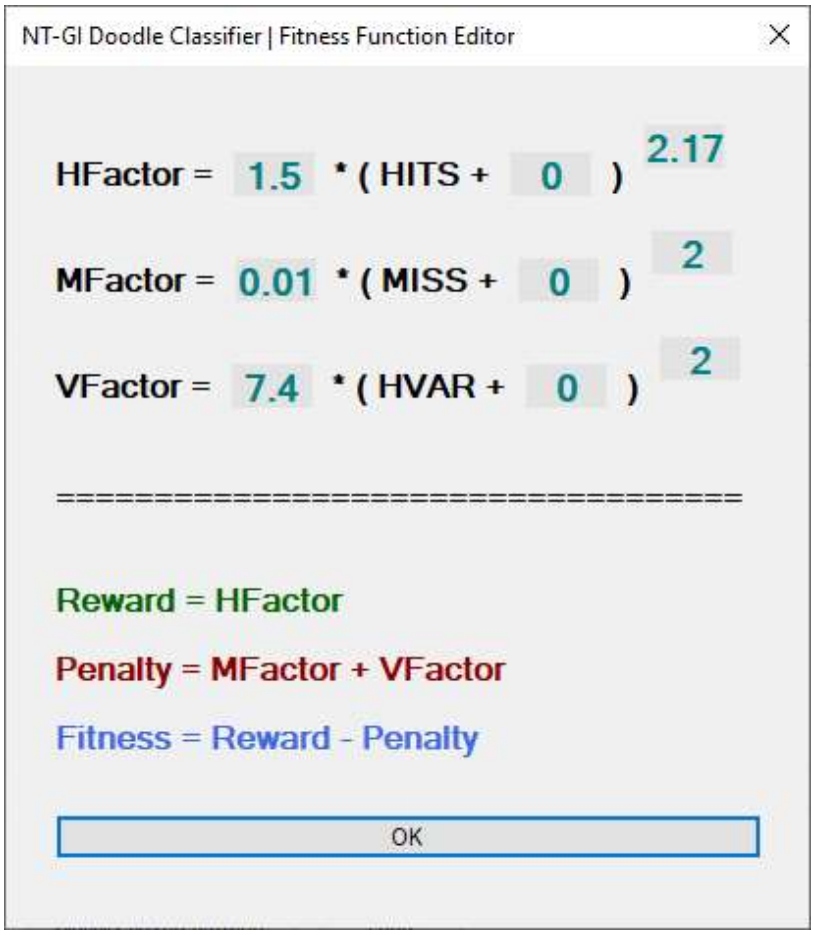
	airplane	banana	cat
airplane	146	15	60
banana	22	162	22
cat	32	23	118

Слика 30 – Представа матрице конфузије

Као што се са слике 29 може видети, корисник може да подешава хиперпараметре тренирања. Такође, може да подешава и функцију способности која се у овом проблему рачуна на следећи начин. Наиме, иако је у питању надгледано учење, донекле се користе принципи УП (награда и казна), уместо поменутих директних функција грешке (*cost* функција). Награда се дефинише као број исправно класификованих података у текућој евалуацији датог модела (*HITS*), уз додатне коефицијенте и степеновање како би се раније описани сродни проблеми овом техником разрешили. Иначе, како је скуп

података веома велик, не користе се сви подаци за евалуацију у једном кораку, већ се за сваку генерацију бира само подскуп података (на не потпуно случајан начин) који се у ову сврху користе (енгл. *mini batch*, попут техника које се користе у *SGD*). Казна се дефинише као збир два фактора. Први фактор је од мање важности и представља број неисправно класификованих података у текућој евалуацији (*MISS*). Дакле, функцијом способности (која је и оптимизациона функција), потребно је моделирати „доброту“ модела, па до сада описаним појмовима (број погодака *HITS* и број промашаја *MISS*), очигледно је да се преферирају модели који имају већи број исправно класификованих података.

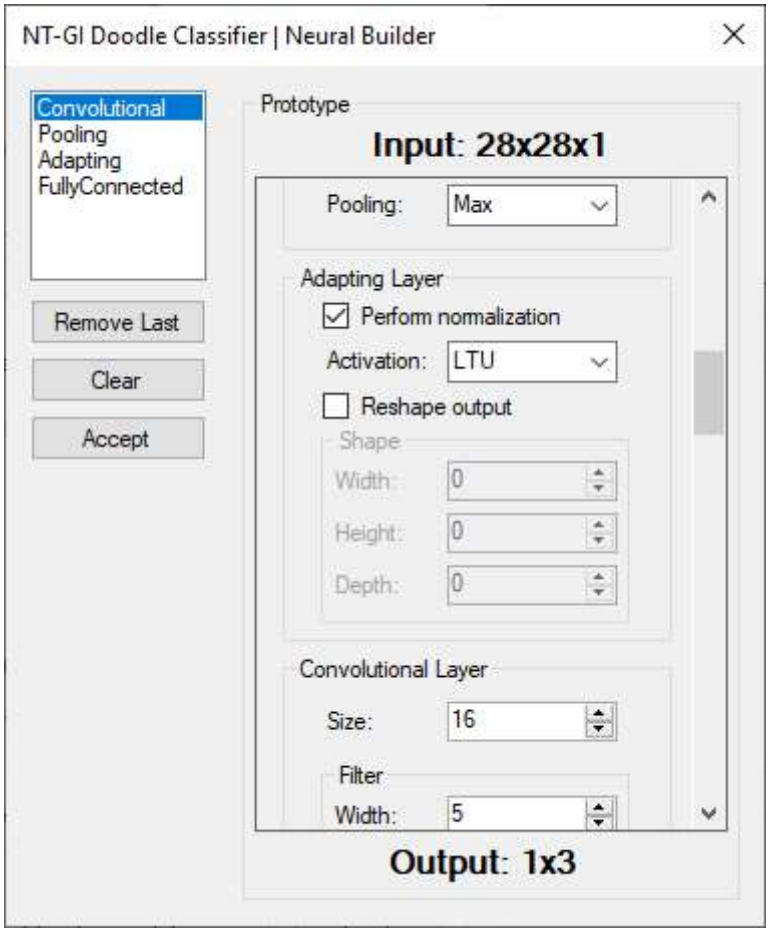
Ако би се само на овај начин функција способности дефинисала, НЕ би упадала у локалне минимуме где се појављује пристраност (енгл. *bias*) у бирању једне или подскупа класа од оних које постоје, што није добро. Проблем настаје у томе да модели који исправно више слика класификују постају по еволуцији бољи, али уколико су то у питању подаци исте класе, ово заправо представља лоше понашање! Стога, уводи се такозвани *HVAR* податак који представља други фактор казне, а идеја је да што је варијанса у исправно класификованим подацима већа, то значи да дати модел има поједине класе које исправно боље класификује у односу на друге класе, што је лоша особина. У идеалном случају, број исправно класификованих података треба да буде униформан по свим класама, чиме је дакле поменута варијанса 0, и казне у том случају нема. На слици 31 представљен је *GUI* где корисник може коефицијенте овог модела функције способности и да мења, чиме утиче на ток и ефикасност обучавања.



Слика 31 – Подешавање функције способности

Интересантно је дискутовати о начину укрштања у овој методи. Када се укршта више родитеља, *RUC* метода показује веома лоше особине јер се овде ради о већим мрежама (и кроз много генерација, нема напретка јер је варијација система превелика). *RFC* и *SEC* методе овде показују напредак, стим да се код *SEC* методе понекад губи варијација, а *RFC* метода не само да се показује као најбоља већ и оправдава тврдњу да поједини тензори у оквиру НМ садрже заједничко знање, а у оквиру конволуционих мрежа, сваки филтер је пректично један тензор који је независне природе у односу на друге филтере чак и у истом слоју, па је *RFC* метода природни избор за укрштање конволуционих мрежа.

До сада су само дискутовани хиперпараметри ГА. Такође, у алату је развијен и *GUI* градитељ неуралних мрежа како би корисник такође могао да прави произвољне мреже и у овом проблему их и користи. Наравно, и прототипе мрежа, као и саме истриниране мреже може да чува у датотеке, као и да их касније учитава. Интересантно је да је градитељ из библиотеке компатибилан са *XML* форматом и веома лако се може на овај начин и представити.



Слика 32 – Градитељ неуралних мрежа

6.2. НЕУРО-ЕВОЛУЦИЈА АУГМЕНТОВАНИХ ТОПОЛОГИЈА

Још једна напредна техника која је истраживана у овом раду јесте техника неуро-еволуције аугментованих топологија (енгл. *NeuroEvolution of Augmenting Topologies*, *NEAT* [22]). Ова техника се показује као веома ефективна у обучавању НМ у оквиру НЕ, нарочито за УП системе. У овом раду, испробана је на трчећим агентима, и показује побољшање у обучавању агената на тешком терену. Наиме, као и обична НЕ, на лаком и средњем терену, већ у првој генерацији се појављују агенти који се бесконачно по њему крећу, а код тешког терена, обичној НЕ је потребно око 30 генерација у просеку како би настало неколико агената који бесконачно трче по том терену, док је *NEAT* техници потребно 2-7 генерација у просеку да преко 150 од 200 агената бесконачно трче. Међутим, у овом раду није имплементиран директан *NEAT* алгоритам описан од стане аутора, већ су одређене измене уведене.

6.2.1. ОСОБИНЕ *NEAT* АЛГОРИТМА

Једна од главних предности *NEAT* алгоритма у односу на основну НЕ јесте та што *NEAT* алгоритам такође поседује способност структурног учења. До сада описана учења која су се одвијала су само параметарска учења, односно, само се параметри модела мењају (само они „уче“), док стуктура остаје иста (између осталог, читава популација у НЕ је морала да по дефиницији има НМ исте структуре). Поред потенцијала проналаска глобалног минимума, способност структурног учења је још једна предност НЕ у односу на методе градијентног спуста. Наиме, под структурним учењем се сматра да алгоритам има способност да додатно и структуру саме НМ мења, односно, да „учи“ које су то структуре добре за конкретан проблем, па тиме такве структуре и конструише.

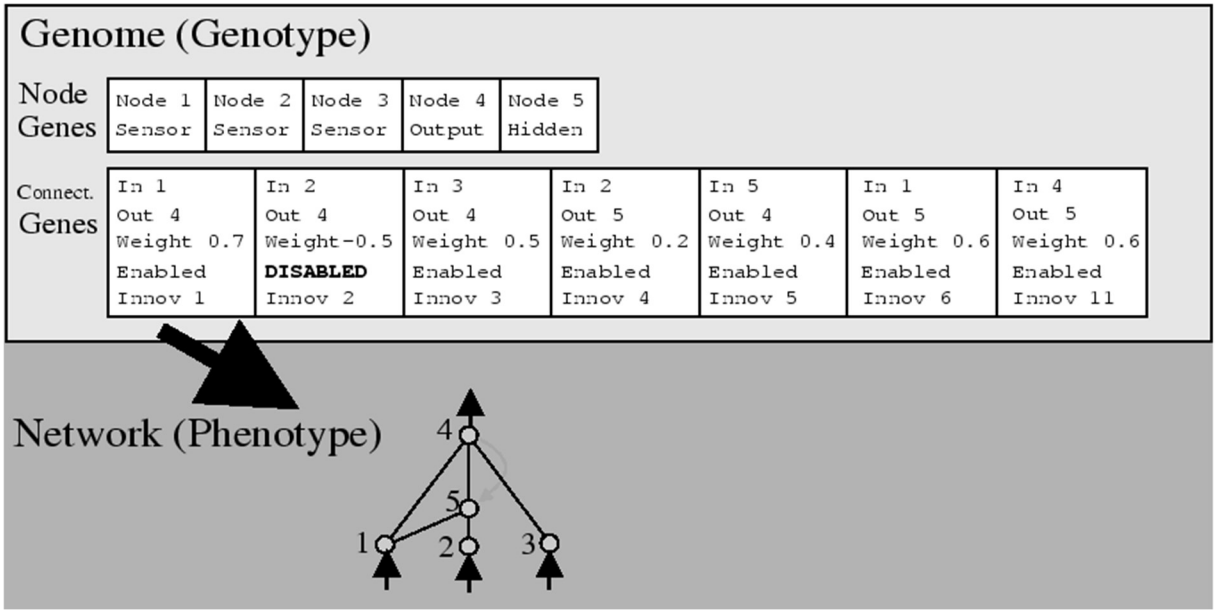
Неколико проблема настаје када се тражи систем који би имао способности као што је претходно описано. Први проблем представља потражњу генетичке репрезентације (генотипа) који би омогућио да се НМ разноврсних структура укрштају на неки смислени начин. Наиме, чак и у обичној НЕ где се ради о идентичним структурама, показало се да поједина укрштања имају веома лоше особине (*RUC* на великим НМ) иако представљају добре методе укрштања у другим применама ГА. Други проблем који се јавља је тај да стуктурне иновације у оквиру популације које настају иницијално у већини случајева обарају перформансе (односно функцију способности) исте индивидуе, па тиме не успевају да опстају дуже од свега неколико или чак једне генерације, а могу бити круцијалне за проналажење стуктура које се веома добро понашају на датом проблему (потребно је неко време да се прво оптимизују). Трећи проблем који се природно намеће је, с обзиром да сада структуре нису фиксиране, а како је већ до сада речено да превелике НМ не морају директно и бити добре за дати проблем, како да се пронађу минималне структуре које представљају потребна и довољна решења чиме се постиже не само стицање квалитетне НМ, већ и мање чиме се смањују рачунарски (временски) захтеви у њеном раду и самим тим, могућност примене јој се повећава.

NEAT алгоритам све претходне проблеме на веома ефикасан начин разрешава, а такође се показује и као веома ефикасан у пракси. Такође, *NEAT* се може сматрати и варијантом алгоритма у НЕ која одређене хиперпараметра саме НЕ претвара у (учеће),

сада параметре система. Ова особина је добра, чак и пожељна, јер се корисник мање оптерећује потребним подешавањима пре покретања алгоритма.

6.2.2. Принцип рада *NEAT* алгоритма

Потребно је прво објаснити генотип који се користи у овом алгоритму. За разлику од НЕ, генотип и фенотип се овде разликују, где је фенотип наравно НМ, у овом случају одговарајуће структуре (која је диктирана генотипом), а такође, у оквиру самог генотипа налазе се и информације о вредностима параметара саме мреже (појачања, склоности, …). На слици 33 може се видети пример мапирања генотипа у фенотип.



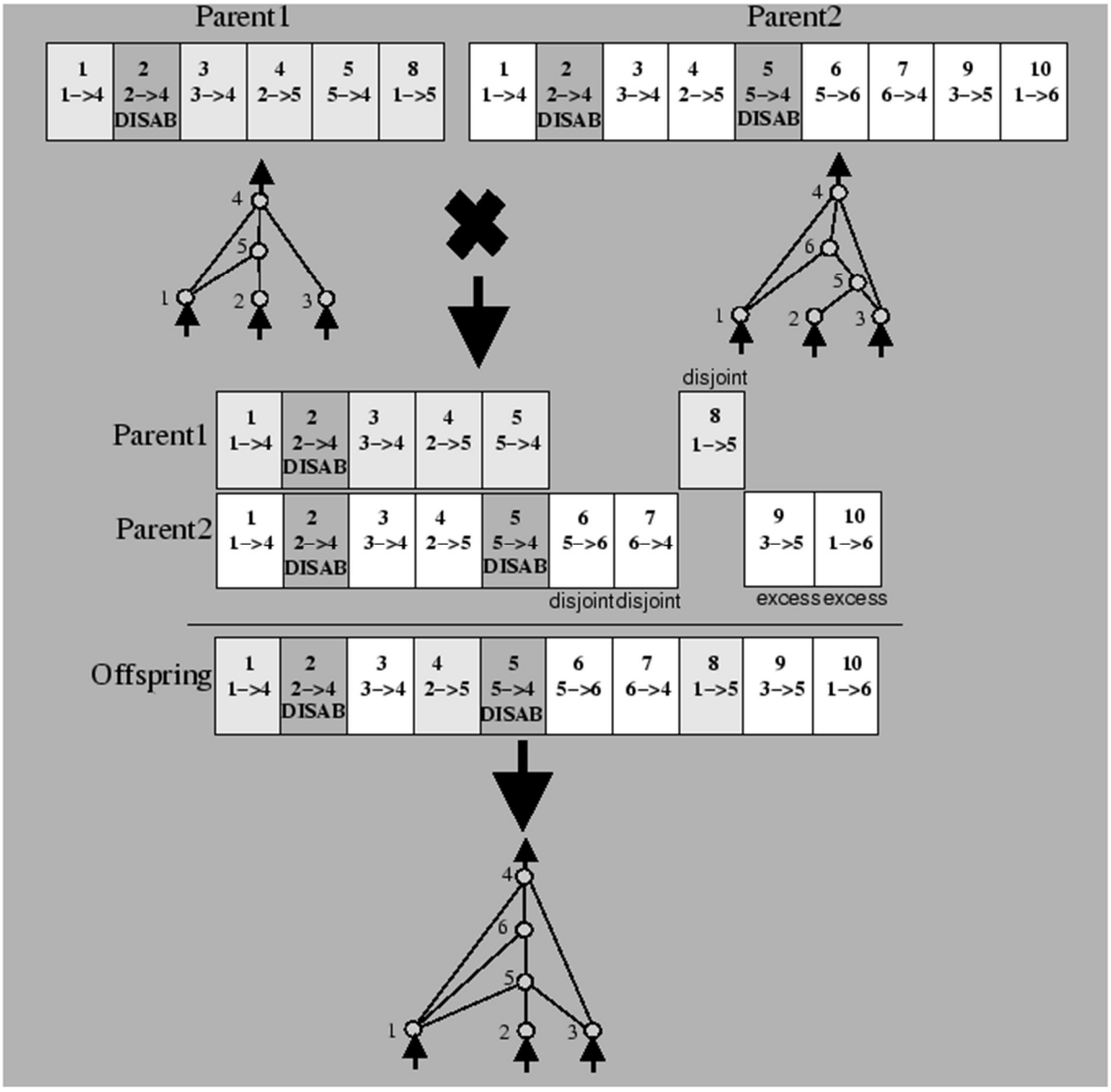
Слика 33 – Мапирање генотипа у фенотип у *NEAT* алгоритму

Овде постоје гени који носе информације о самим неуронима (чворовима, енгл. *Node*), као и гени који носе информације о везама (конекцијама, енгл. *Connection*). Мада, у пракси се може поставити питање да ли су гени чворова битни. Често је довољно имати информацију о броју улазних, излазних и скривених чворова, где се ови чворови донекле могу објаснити на исти начин као слојеви у вишеслојној неуралној мрежи описаној у поглављу 2.1.3. Поново, улазни неурони не врше никакво процесирање, а излазни представљају оне чији се излази користе као излази саме мреже. Скривени неурони су они који се налазе између ових. Иако улазни неурони не врше никакво процесирање, често се ипак користе у представљању мрежа добијених *NEAT* алгоритмом јер овде више нема потпуно повезаних слојева, односно, унапред се не зна какве конекције ће уопште постојати, па тиме, алгоритам можда поједини улаз уопште ни не повеже, јер је, на пример, пронашао да је тај улаз небитан (односно та одлика је небитна, па тиме алгоритам има и могућност да научи које су одлике од датих битне).

Гени конекција у себи носе 3 основне информације, а то су: неурон из кога потичу, неурон до ког се протежу (дакле од-до принцип, енгл. *from-to*, мада се у литератури користи *in-out* што може да буде контра-смислено), и обавезно само појачање везе (заправо учећи параметар). Поред ова три податка, ген такође и носи информацију о томе да ли је сам ген омогућен (енгл. *enabled*). Наиме, ово се користи у даљим операцијама у

оквиру ГА, а практично у тренутку када сам ген није омогућен, даје исту информацију као да је појачање везе једнако 0, али се ипак анулирање тог податка не користи јер у неким моментима ген може да се реактивира. Може се повући аналогија са природом да постоје „гени који спавају“ (енгл. *dormant*), а који се у неком моменту могу активирати.

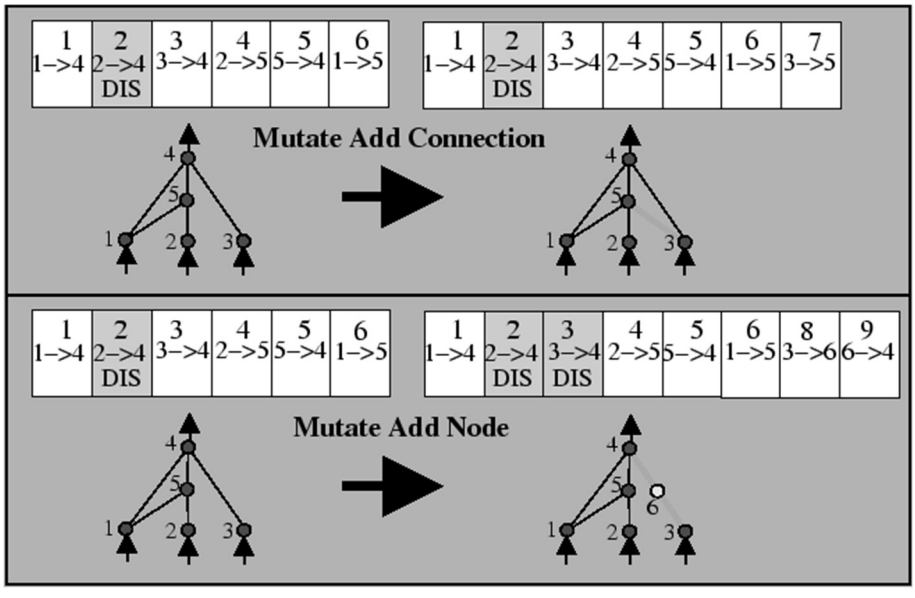
Најбитнији део *NEAT* алгоритма јесте такозвани иновациони број. Иновациони број се придружује стуктурној иновацији које настају у овом алгоритму, и то се практично придружује конекцијама (везама), на такав начин да се у будућности, исте стуктурне иновације могу детектовати тиме што имају исти иновациони број. Ово су гени који се у оквиру генома могу поравнати и укрштање се управо помоћу њих и одвија. Када се ген наслеђује, наслеђује се и његов иновациони број, чиме се омогућује да и наредне генерације прате иновације предака и знају како да их преносе даље. У литератури, ова појава се назива историјским маркерима (енгл. *Historical Markings*). На слици 34 може се видети начин укрштања два различита генома, који представљају по структури две различите неуралне мреже.



Слика 34 – Укрштање у *NEAT* алгоритму

Једна од битних карактеристика *NEAT* система је да све неуралне мреже морају имати исти број улаза и излаза (тј. улазних и излазних неурона). Ово иначе не представља ограничење јер, заправо, ови бројеви ни не треба да се мењају из перспективе употребе саме НМ. Међутим, број скривених неурона може да се произвољно мења, а како гени неурона не садрже битне информације, овде нису ни представљени. Гени конекција су од круцијалне важности у укрштању. Поравнање се врши тако што се гени са истим иновационим бројевима поравнају (а ови бројеви су увек у растућем редоследу записани), а затим се, рецимо, насумично бира један од родитеља чији ген се копира у исти ген потомка. Могуће је да се деси да неки ген који се налази у једном родитељу не постоји у другом (могуће је ово генерализовати и на више родитеља). Тада, може се поступати на више начина, па се овим рађају различите идеје за укрштања у овом алгоритму (у овом раду је коришћен алгоритам попут *RUC*, где када се у тренутку избора родитеља од ког се дати ген наслеђује, уколико тај родитељ такав ген и поседује, ген се копира, иначе се ген прескаче јер га изабрани родитељ нема и наставља се даље са наредним генима – у литератури алгоритам функционише слично, али не и исто).

Мутација се у у овом алгоритму јавља у два облика. Први облик је наравно исти као у обичној НЕ и функционише идентично као *RNM*. Ово су параметарске мутације. Други облик мутације је структурне природе. При оваквој мутацији, постоје две варијанте. Једна варијанта је да се при таквој мутацији бирају насумично два неурона између којих не постоји конекција, а затим се таква конекција додаје (*Add Connection Mutation*), а други вид стуктурне мутације је да се додаје нов неурон (*Add Node Mutation*). Оваква структурна мутација се у овом алгоритму може десити само на некој (насумично одабраној) вези (конекцији), тако да практично нов неурон настаје на месту изабране конекције, односно, нов неурон полови дату конекцију. Имплементационо, стара конекција се у овом тренутку онемогућује (енгл. *disable*), а стварају се две нове конекције, тако да се једна конекција протеже од полазног неурона до новонасталог, а друга од новонасталог до одлазног. Конекција од полазног до новог као појачање добија 1, а конекција од новог до одлазног неурона добија исто појачање као онемогућена, стара веза. На слици 35 представљене су структурне мутације.



Слика 35 – Структурне мутације у *NEAT* алгоритму

Када се структурне мутације дешавају, нове конекције добијају нове иновационе бројеве (ово је извор тражења других структура). Постоји глобални генератор ових бројева који се дељено користи од стране свих чланова популације, а функционише тако што у тренутку стварања нових конекција (практично нових гена), генератор додељује новом гену своју вредност, а потом је инкрементира, па се тиме новим генима даје повећавајућа вредност иновационих бројева. Могуће је такође да се у оквиру исте генерације засебно случајно десе исте мутације, па се у оквиру репродукције и прављења наредне генерације памте мутације које су се десиле, па уколико се на другом месту поново десе, исти иновациони број им се додели, како не би дошло до експлозије иновационих бројева. Међутим, исте мутације се могу десити и у наредним генерацијама, али ове мутације ипак у еволуцију могу да представљају нешто друго, што је иначе отворена дискусија за сам алгоритам.

У овом раду, структурне мутације се ипак имплементирају нешто другачије. Два вида структурне мутације се донекле групишу. Првенствено се гледа да ли долази до структурне мутације испитивањем генералне вероватноће мутације. Ако долази, насумично се бирају два постојећа неурона. Уколико између ових неурона не постоји веза, она се додаје (може јој се доделити насумично појачање али се у овом раду додељује 0), што практично представља *Add Connection* мутацију. У супротном, уколико између ова два неурона постоји веза, са 95% шансе ова веза мутира тако да се на њој додаје неурон, што представља *Add Node* мутацију и функционише исто као претходно описано, а са 5% шансе ген мења своју омогућеност (ако је био омогућен, онемогућује се, а ако је био онемогућен, омогућује се и тиме постаје активан, односно реактивиран). Овај поступак се за мутацију понавља 3 пута, како шансе, па и број настајања структурних мутација не би био премали у односу на параметарске (мада, вероватно треба да буде мањи).

NEAT алгоритам заправо започиње иницијалну популацију НМ са минималним структурама. Идеја је да НМ постепено расту кроз структурне мутације. Тиме се омогућује да алгоритам започне са најмањим могућим структурама НМ, па уколико су овакве структуре превише мале за дати проблем, еволуцијом ће се преферирати оне које су веће (које ће се и временом креирати), међутим, уколико су довољне, оне ће одмах бити добре. Овим *NEAT* алгоритам решава проблем проналажења минималних НМ које су адекватне за дати проблем. У овом раду, под минималном структуром, сматра се структура без и једног скривеног неурона, односно постоје само улазни и излазни неурони, а такође постоје везе између свих улазних и свих излазних неурона (попут потпуно повезаног слоја), јер се почетно сматра да су све изабране одлике, односно улази битни.

6.2.3. Избор у *NEAT* алгоритму

Као што је раније наведено, како би ГА био потпуно дефинисан, неопходно је дефинисати укрштање и мутацију јер су ове две операције зависне од генотипа и конкретног проблема, док избор представља општи појам. У претходној тачки су управо ове две операције дефинисане, међутим, када би се направио тако описан систем где би се рецимо користила пропорционална селекција (или у суштини било која друга), *NEAT* алгоритам не би показивао довољно добре перформансе јер, као што је већ поменуто, нове стуктурне иновације углавном као ефекат имају да иницијално слабе саму

индивидуу (односно, смањују јој способност). Рецимо, код *Add Node* мутације, уводи се додатна нелинеарност у тој грани која не мора и углавном и није одмах повољна с обзиром да је насумична.

Главна идеја у овом моменту је да се стуктурне иновације штите тако што ће се увести појам врсте или расе индивидуе (енгл. *species*). Структурно сличне индивидуе треба да припадају истој врсти, док структурно различите треба да буду одвојене. Када се индивидуе на овај начин разврстају, уместо да се такмиче на глобалном нивоу, оне се такмиче у избору унутар своје групе, чиме се штити њихова структура.

Потребно је само пронаћи начин како да се дефинише које индивидуе су „сличне“, а које су „различите“. Ово делује као проблем анализе структура графова, али, испоставља се да иновациони бројеви представљају поново добар механизам како би се могло установити колико су посматране две индивидуе различите. Све што је потребно погледати је поравнање гена конекција. Они гени који се налазе у обе индивидуе практично диктирају исте структурне иновације и по тим генима су ове две индивидуе сличније. Оно што указује на њихову разлику су гени који се налазе у једној индивидуи а не налазе се у другој. Број оваквих гена управо даје одређену меру која диктира различитост ове две индивидуе. У литератури се дефинишу два типа гена који се не поклапају (енгл. *mismatching*), а то су *excess* и *disjoint*. Такође, у литератури се и разлика у појачањима користи као додатни фактор мере различитости две индивидуе, али у овом раду, појачања се не користе из разлога што се сматра да појачања не представљају структуру НМ, а такође, разлика између поменута два типа гена који се не поклапају се такође не прави јер цео процес разврставања функционише на нешто другачији начин него како је у литератури описан.

Наиме, у литератури такође постоји концепт дељења способности (енгл. *fitness sharing*), односно, способност индивидуе се нормализује (дели) бројем индивидуа у тој врсти. Разлог овом је да се не дозволи једној врсти да преовлада читавом популацијом. Без овога, иако би се структурне иновације штитиле, животни век неке стуктурне иновације би се потенцијално продужио само за неколико генерација више у односу на конвенционалну селекцију без заштите, јер би нека врста могла да преовлада у читавој популацији. Овим се управо наведено избегава.

У овом раду, разврставање ипак функционише нешто другачије. Наиме, у литератури, само у оквиру једне врсте индивидуе се такмиче и размножавају, што не мора нужно да представља добру особину. У овом раду, избор се врши на следећи начин. Првенствено се дефинише колико пута ће се избор вршити. Рецимо, у популацији величине 200 јединки, уколико се 3 родитеља укрштају за стварање 1 потомка, избор ће се вршити 600 пута (практично по 3 родитеља за свако укрштање). Затим, један део овога ће бити коришћен за основни алгоритам избора (рецимо, 20% од 600 пута ће се вршити конвенционална пропорционална селекција), чиме се обезбеђује да било које врсте могу бити биране за укрштање. Преостали број избора се пропорционално расподељује на све врсте. Наиме, за сваку врсту, израчуна се просечна способност популације, а затим се у тој пропорцији у односу на остале додели колико пута се мора из ове врсте вршити избор (рецимо, од преосталих 80% укупних 600 избора, некој врсти се додељује 20 што је пропорционално просечној способности те врсте у односу на остале врсте). Овим се обезбеђује да (потенцијално) све врсте буду биране (уколико је некој врсти интензивно ниска способност у односу на остале, пропорционално је могуће да добију 0 као број),

чиме се врсте и штите. Када се у некој врсти врши одабирање, то се може радити основним алгоритмом (рецимо пропорционалном селекцијом, или неком другом). Такође, у овом процесу (дакле када се не ради о избору на глобалном нивоу) када се рецимо бирају 3 родитеља за укрштање, насумично се бира нека врста (са униформном вероватноћом) од оних које још увек нису биране дефинисан број пута, а затим се у оквиру те врсте бира нека индивидуа (са стандардним алгоритмом). Тиме, омогућује се да се такође у оквиру једног неглобалног укрштања не користи само једна врста већ се насумично могу укрштати и са другим, а наравно, случајно, могу се и исте укрштати.

И у овом раду је донекле укључен *fitness sharing*. Ово се манифестује управо јер се врсти додељује онолики број избора који је пропорционалан просечној способности саме врсте. Што у врсти има више индивидуа, то им се и сума способности дели са већим бројем (дели се дакле са бројем индивидуа у врсти, што представља слично дељење као и код метода описаног у литератури). Наравно, уколико све индивидуе у врсти представљају веома јаке индивидуе, просечна способност им неће деградирати јер ће им сума бити велика, па ће тиме и добити већи број бирања, што представља добру особину.

6.2.4. Фенотип

Имплементационо, у овом раду се фенотип генерише на нешто другачији начин. Идеја је иста, али постоје поједини детаљи који се ипак на другачији начин имплементирају. Потребно је свакако и пронаћи имплементациони запис самог фенотипа, јер ово сада више није потпуно повезани слој, штавише, унутар саме мреже се формирају „мини слојеви“ који су произвољне природе. Такође, могу да постоје и резидуалне везе (везе које практично прескачу слојеве, тачније неуроне у овом случају). Поред оваквих веза, *NEAT* алгоритам такође може и да развија рекурентне везе. У општем случају, рекурентне везе формирају циклусе у мрежи и често се у пракси, по имплементацији овог алгоритма, избегавају и чак забрањују. Међутим, у овом раду, рекурентне везе се дозвољавају и чак штавише представљају веома корисну карактеристику, о којој ће више речи бити у наставку.

Пре свега, рекурентне везе представљају и највећи проблем у креирању адекватног записа самог фенотипа. Додатно се отежава околност тога што читав алгоритам треба да се уклопи са до сада описаном библиотеком и начином представе самих НМ, а такође, да се нађе и ефикасан начин да се фенотип представи и на *GPU*. Све заједно представља изазов за имплементацију јер дакле постоје многа поменута ограничења, а нарочито то да је потребно направити перформантно решење.

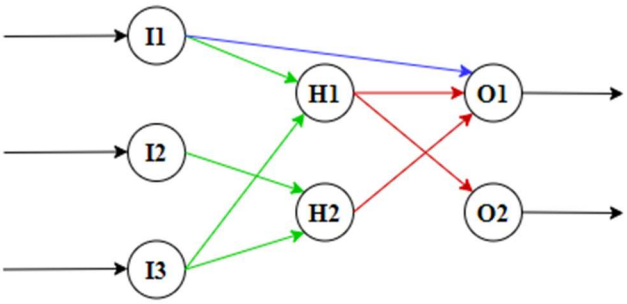
Реализација имплементације започиње првенствено са уклапањем идеје *NEAT* алгоритма у саму библиотеку, конкретно из перспективе фенотипа. Имплементиран је нов слој који се назива динамички слој (енгл. *Dynamic Layer*), и представља практично читав фенотип *NEAT* алгоритма. Дакле, НМ у *NEAT* алгоритму (за сада) се састоје од свега једног слоја који представља читав фенотип. Идеја је да се у будућности овакви слојеви могу користити и у комбинацији са другим. Уколико би дакле овакав слој могао да буде потпуно функционалан и дефинисан, генотип и генотипске операције би могле да се имплементирају као на претходно описан начин и биле би у потпуности независне од фенотипа, што је заправо и добра особина.

Треба даље дефинисати од чега се састоји динамички слој, како функционише и такође решити проблеме који у овом процесу настају. Потребно је посматрати фенотип

у опису *NETA* алгоритма. Ако се изоставе рекурентне везе, практично се добија дирекциони ациклички граф (енгл. *directed acyclic graph* или *DAG*). Графови се могу на више различитих начина имплементирати, а интересантно је да матрична репрезентација представља уједно и погодну репрезентацију за графичке картице, с обзиром да често матрице саме по себи носе добре карактеристике паралелизације. Величина ове матрице је наравно квадрат броја неурона у мрежи (слоју), али, касније се испоставља да је ипак могућа одговарајућа оптимизација у систему, која смањује ову захтевност.

Елементи матрице графа практично дефинишу конекције у мрежи. Индекси самог елемента представљају повезане неуроне, а вредност елемента представља појачање на тој конекцији. Индекс колоне матрице *j* представља полазни неурон, а индекс реда матрице *i* предсатвља одлазни неурон (ова конвенција је намерно уведена, а не контра, која је такође могућа, јер се испоставља као оптималнија и погоднија за одређене операције на *GPU*). Наравно, овом матрицом су дефинисане везе од сваког неурона до сваког, али, уколико је унутар елемента уписана 0, везе практично нема (множење са нулом даје нулу, а у наредном сабирању не даје промену). Као пример, уколико постоје 3 улазна, 2 скривена и 2 излазна неурона, матрица би могла да се конструише као на слици 36.

\	I1	I2	I3	H1	H2	O1	O2
I1							
I2							
I3							
H1	X		X				
H2		X	X				
O1	X			X	X		
O2				X			



Слика 36 – Матрична представа динамичког слоја

Иако су на слици 36 везе обојене различитим бојама, не треба између њих правити разлику, а на слици су боје само илустративно присутне. У матрици се могу видети кореспондни елементи који су маркирани са *X*. Претпоставља се да су то ненулте везе. Сви остали елементи у матрици имају вредност 0, па такве везе ни не постоје. Први проблем на који се овде наилази јесте начин како да се срачунају излази мреже када би се улази довели, али на ефикасан начин. С обзиром да структура графа (односно НМ) може бити произвољна, ово може представљати проблем (у смислу перформанси). Потребно је у овој ситуацији прво израчунати излазе *H1* и *H2* неурона, а затим се могу рачунати излази самих излазних, *O* неурона. Међутим, када би се рецимо везе од *H1* и *H2* до *O1* обрисале (дакле, према *O1* би само ишла веза из *I1*), у том случају, излаз *O1* би се могао рачунати истог тренутка кад и *H1* и *H2*. Практично се добија зависност редоследа рачунања излаза неурона у зависности од структуре, и то не само у оптимизационом смислу (да се неки могу паралелно рачунати), већ и обрнуто гледано,

када би се везе у претходном примеру вратиле назад, уводи се нова зависност која ограничава да сада излазни неурон *O1* мора да чека на рачунање *H1* и *H2* пре него што би он могао свој излаз да срачуна. Дакле, како се структура мења, мења се и редослед по ком се излази неурона морају рачунати (оваквих редоследа има више, а то диктирају они неурони који могу у паралели да рачунају своје излазе, тј. чији се редослед може изменити а да при томе евалуација НМ буде иста). Тражење овог редоследа може бити скупа операција.

Претходна идеја се врло брзо компликује у наредној ситуацији. Рецимо да се на слици 36 деси мутација и да настане нова веза која иде од *O1* ка *H1* (што је валидно по *NEAT*-у). Проблем који се сада догађа је тај што претходно описани начин рачунања излаза важи само уколико је дати граф заиста *DAG*. Међутим, сада се појављује циклус. Практично, као и раније, *O1* мора да чека на *H1* како би могао да започне своје израчунавање, а сада због нове везе, и *H1* мора да чека на *O1* како би започео своје израчунавање. Практично, ни један од дата два неурона не могу да започну рачунање јер се због постојања циклуса међусобно чекају (енгл. *deadlock*). Наравно, у овој ситуацији можемо одлучити да се прво рачуна *H1*, али ово је само специјални случај који се лако разрешава. У општем случају ово могу бити два скривена неурона за које је јако тешко одлучити који се први рачуна, а чак и да се ово разреши, потенцијално може постојати иста структура оваквих неурона као подмрежа у некој другој мрежи где би алгоритми, из разлога што је сама мрежа другачија, требали да донесу одлуку да се ипак други неурон раније срачуна.

Све и да се претходни многобројни проблеми разреше, поставља се питање да ли би оваква мрежа била стабилна. Уведимо претпоставку да постоји механизам који правилно одлучује који неурони треба прво да се рачунају у оваквој ситуацији (са циклусима). Нека су то два неурона *H1* и *H2* који се тим редоследом рачунају, а међусобно су завнисни. Након што се добије излаз *H1*, податак се води до *H2* и рачуна се *H2*. Након што се срачуна *H2*, излаз *H2* се наравно води даље, али такође и до *H1*. Сада, неурон *H1* има другачији податак на свом улазу и требало би излаз да коригује. Након тога, *H2* ће поново бити у сличном проблему. Ово се може генерализовати тако да проблем настаје било кад када се уведе циклус у граф (па граф престаје да буде *DAG* јер није ациклички).

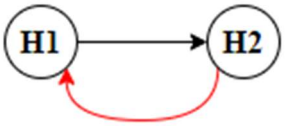
Дакле, са циклусима, динамички слој у претходно описаној варијанти је нестабилан и, све и да постоји механизам за разрешавање редоследа рачунања неурона (чак и са циклусима, што је веома тешко), мрежа никад не би давала стабилне податке на излазу. Управо је ово разлог зашто се рекурентне везе избегавају јер, практично, рекурентне везе уносе циклус у граф и мреже постају неупотребиве.

У овом раду, ипак, рекурентне везе представљају везе са веома лепим особинама и не избегавају се, штавише донекле се и фаворизују (у границама нормале), па тиме и „циклуси“. Наиме, решење свих проблема почиње управо од саме помисли да додавањем рекурентне везе, циклус у графу се и формира. Ово није увек једноставно детектовати, и потребне су разне анализе графа које могу бити скупе како би се ово заправо детектовало.

Започнимо решавање претходних проблема уназад. Да би се избегле комплексне анализе за детекцију рекурентних веза, у овом раду, оне се дозвољавају. Тиме, анализе постају непотребне и не имплементирају се. Међутим, ово са собом повлачи много проблема. Први уназад на који се наилази је нестабилност НМ. Идеја за решење овог

проблема је инспирисана тактом у дигиталним уређајима. Сваки неурон памти свој излаз у сваком моменту. Само на појаву „такта“, он гледа своје улазе и мења излаз. Сигнал „такта“ је практично дефинисан једном евалуацијом излаза када се улази доведу. Наравно, и даље је потребно да неурони у одређеном редоследу рачунају излазе (дакле, не рачунају се сви у исто време), али сада, у ситуацији када имамо два неурона који међусобно зависе један од другог, ако бисмо имали механизам који одлучује који је то неурон који се први рачуна, првенствено би се тај рачунао, а затим други, а сада, како је „такт“ за први неурон прошао, његов излаз је стабилно меморисан и мрежа постаје стабилна, па и даје стабилне излазе. Дакле, само једном се по наиласку улаза излази неурона рачунају. Ово понашање је јако слично леч (енгл. *latch*) и флип-флоп (енгл. *flip-flop*) колима у електроници.

Даље треба конструисати механизам за одлуку о редоследу рачунања излаза неурона. Све и да је ово једноставан проблем, временски захтеви имплементације могу врло брзо да ескалирају. Треба прво посматрати саме рекурентне везе. Наиме, то су везе које уносе циклус у графу и, без њих, анализом *DAG*-а, проблем и не би био толико комплексан, међутим, рекурентне везе су те које компликују дати проблем. Ипак, можемо искористити један врло једноставан принцип.



Слика 37 – Рекурентна веза

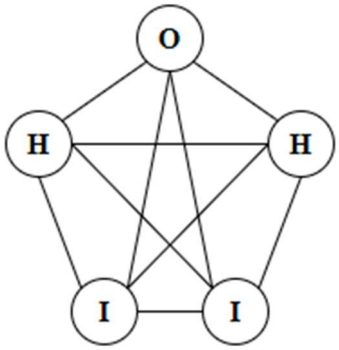
На слици 37 црвеном бојом је означена рекурентна веза. Рецимо да су ова два неурона делови веће мреже. Ова ситуација такође лепо илуструје и проблем где се локално гледано не може установити која од дате две везе представља рекурентну, али рецимо, када би се представила цела мрежа, веза црвеном бојом би била рекурентна. У тренутку када је познато која је веза рекурентна, може се применити следећи приступ. Наиме, ова ситуација није ништа другачија него претходно описана у решавању проблема нестабилности помоћу „такта“. У тој ситуацији, *H1* са слике 37 се прво рачуна, а затим *H2* (не гледајући у обзир то да је црвена веза рекурентна). Међутим, овде се интересантна ствар догађа. Податак који се користи на конекцији од *H2* до *H1* представља податак који је срачунат (и уморисан) на излаз *H2* у **претходном** тренутку (што има једну веома лепу особину која ће се накнадно описати). Након тога, када се срачуна *H1*, рачуна се и *H2*, чиме се добијају нове вредности. Идеја је да се заправо овде рекурентном везом прослеђује податак из претходног тренутка, а са рачуницом се наставља даље. Отуда долази и једноставан механизам за одлуку који неурони у оваквим ситуацијама треба прво да буду срачунати. Да је на слици 37 црна веза била рекурентна, прво би се рачунао неурон *H2*, а затим *H1*. Једноставније речено, рекурентне везе не утичу на редослед рачунања неурона, јер се у њима користи податак из претходног тренутка. Ово се може генерализовати на било какав циклус у графу и тиме добије једноставан механизам за одлуку о редоследу рачунања неурона у „циклусу“. Да принцип „такта“ не постоји, не би се прослеђивала информација из претходног тренутка,

па тиме ни претходна тврдња не би била тачна. Овако, природно се ове две идеје уклапају.

Разлог зашто прослеђивање података из претходног тренутка представља позитивну особину је управо то што се на овај начин реализује меморија у „мозгу“. Практично, чувају се подаци који носе неку информацију о томе шта се у претходном моменту дешавало. Ова информација се затим користи у рачунању других података који могу бити пренети кроз друге (или исте) рекурентне везе у наредним моментима, па се тиме информација заправо простире кроз више тренутака, односно, читав одређени временски период. Како се углавном ради о појачањима која су по апсолутјон вредности мања од 1, утицај података из претходних тренутака се кроз време смањује, односно, они подаци, тј. оне информације које су пристигле до „мозга“ давније су више „заборављене“ (утицај им је мањи), а оне које су раније пристигле су свежије, а вероватно и битније. Отуда рецимо, код тркача, рекурентне везе могу да им потенцијално дају информацију о томе за колико су се заокренули у односу на претходне тренутке. Мада ово можда не делује као битна информација, у неким другим проблемима, овај принцип итекако може да помогне.

Дакле, сада је могуће посматрати само *DAG* и разрешити редослед неурона анализом топологије графа. Међутим, враћен је проблем детекције рекурентних веза. Ипак је потребно детектовати које су то везе рекурентне како би се оне „избациле“ и тиме покренуо алгоритам анализе топологије *DAG*-а (интересантно је да реализацијом одговарајућег решења ни једно од ова два је заправо потребно!). У овом моменту, треба погледати матрицу конекција са слике 36. Сигурне рекурентне везе се налазе на дијагонали (то су, дакле, ситуације где неурони имају везу која од њих и потиче и ка њима се простире), а то су такозване сопствене везе. Међутим, у општем случају, остале рекурентне везе је теже детектовати. Ипак, постоји нека правилност у томе које везе могу бити рекурентне. Посматрајмо то са слике 37. Обе везе могу бити рекурентне, али само једна заиста и јесте (у суштини било која). Слично се може извести и за циклусе који су индиректно реализовани (дакле, више неурона учествују у формирању истог).

У општем случају, може се посматрати дирекциони граф од n елемената, у ком се налазе све могуће конекције (све могуће гране постоје). Из анализе се изостављају сопствене везе, а њих је n. Граф се може посматрати као један n-тоугао, где се везе између парова неурона могу представити једном бидирекционом граном (узмеђу осталог, ово је сада обичан недирекциони граф). На слици 38 може се видети представа овог графа (n-тоугла).



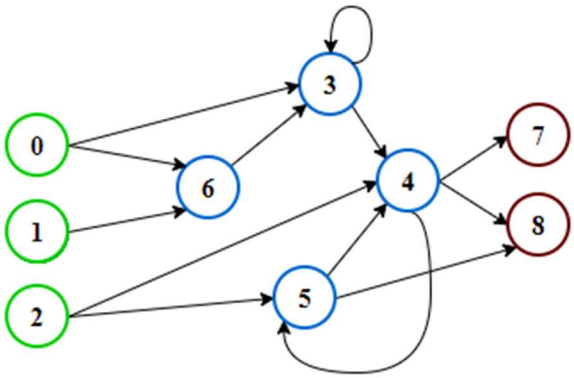
Слика 37 – потпуно повезан граф

Свака грана између два чвора практично представља једну директну и једну рекурентну везу (није могуће да су обе рекурентне или обе директне). Број оваквих веза једнак је збиру броја ивица *n*-тоугла и броја дијагонала. С обзиром да су неки чворови, односно неурони (барем један) улазни, а неки (поново барем 1) излазни, можемо рећи да граф садржи неки вид „усмерења“. За ово је веома битно да исти чворови не могу бити истовремено улазни и излазни. Додатно, рекурентна веза не може потицати од улазног неурона (ово нема смисла), нити може да иде ка излазном чвору (такође нема смисла). То не значи да овакве везе не постоје, чак напротив, то су обичне директне везе које потичу од улазних неурона и иду ка другим у мрежи, или су то поново директне везе које иду ка излазним неуронима од улазних или других скривених. Уз ове критеријуме, важи да када се за сваки пар чворова (дакле, за све гране у графу) избаци рекурентна веза (за улазне и излазне чворове зна се које морају бити директне, а за међучворове, односно скривене неуроне, може се произвољно изабрати једна), добијени граф остаје ацикличан (и представља *DAG*). Отуда, може се извести да је максимални број рекурентних веза у мрежи са *n* чворова једнак:

$$R_c(n) = n + D_n = n + \frac{n \cdot (n-3)}{2} = \frac{n \cdot (n-1)}{2}$$

Наравно, ово је у ситуацији где се не рачунају сопствене гране, али таквих грана је такође *n*. Ово доводи до следеће сугестије. С обзиром да се око скривених неурона може произвољно бирати која је веза рекурентна а која не, може се заправо фиксирати алгоритам који увек говори да је предодређена грана рекурентна, без обзира од топологије. Наиме, алгоритми који „траже“ рекурентне везе, заправо покушавају њих да избегавају, а у тренуцима када је то немогуће, морају неку грану да прогласе рекурентном (која реализује циклус), а ово практично представља тополошку „анализу“ графа која може бити скупа.

\	0	1	2	3	4	5	6	7	8
0									
1									
2									
3	X			X			X		
4			X	X		X			
5			X		X				
6	X	X							
7					X				
8					X	X			

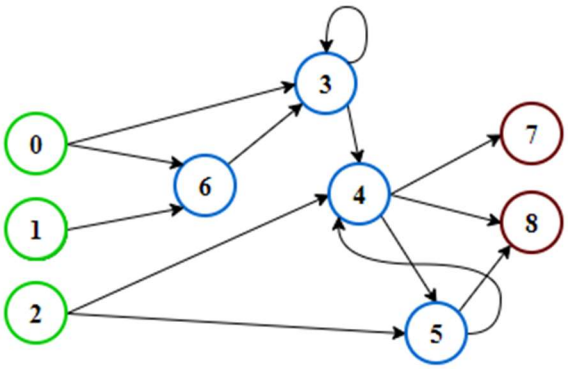


Слика 38 – Анализа рекурентних веза

Посматрајмо ситуацију на слици 38. Зеленом бојом означени су улазни неурони, плавом скривени, а браон бојом означени су излазни неурони. Може се поставит питање

које везе овде представљају рекурентне. Поред спствене везе на неурону 3, на први поглед, може се рећи да је веза од неурона 4 ка неурону 5 рекурентна. Међутим, када посматрамо пар неурона 4 и 5, претходно је дискутовано да је свеједно која је веза рекурентна. На слици 39, ово је и илустровано.

\	0	1	2	3	4	5	6	7	8
0									
1									
2									
3	X			X			X		
4			X	X		X			
5			X		X				
6	X	X							
7					X				
8					X	X			



Слика 39 – Представа рекурентне везе на други начин

Матрична репрезентација говори да је у питању иста мрежа, а ипак, на слици 39, веза која је претходно дискутована сада делује као директна веза, где претходно директна веза од неурона 5 ка неурону 4 сада делује као рекурентна. Главна идеја у овоме је да је графичка репрезентација саме НМ небитна, а да се у оквиру матрице, поред дијагонале, одабере фиксиран број $\frac{n\cdot(n-1)}{2}$ веза које би се унапред прогласиле рекурентним.

Ово као последицу за собом вуче ситуацију да се нека мрежа може прогласити потпуно директном (без рекурентних веза), а да се у стварности ипак нека веза у њој манифестује као рекурентна (уместо директне). На пример, рецимо да се са слике 39 фиксира да је веза од неурона 5 ка неурону 4 рекурентна. Након тога, рецимо да се мутацијом веза од неурона 4 ка неурону 5 онемогући. Веза 5 ка 4 и даље представља рекурентну, иако заправо у тој ситуацији она може представљати директну.

Тополошке анализе динамички рекурентне везе померају да оне не буду фиксиране, па тако, ако могу да прогласе мрежу потпуно директном, то ће и урадити (управо ово би се десило у претходно описаној ситуацији – практично ови алгоритми померају проглашење рекурентних веза да буду на местима где везе не постоје, колико је то максимално могуће, чиме их избегавају). Дакле, са таквом анализом, фаворизују се директне везе. Међутим, дискутовано је да рекурентне везе са собом носе заправо потенцијално добре особине, нарочито за УП проблеме (ако баш и нису погодне, еволуција би требало њих да уклони), а фиксирањем рекурентних веза оне се не избегавају, али ни не фаворизују (практично се стављају на једнакост са директним).

Сада само треба одлучити које везе (сем сопствених на дијагонали) представљају рекурентне (али ту такође постоји ограничење, да не могу на пример обе везе између

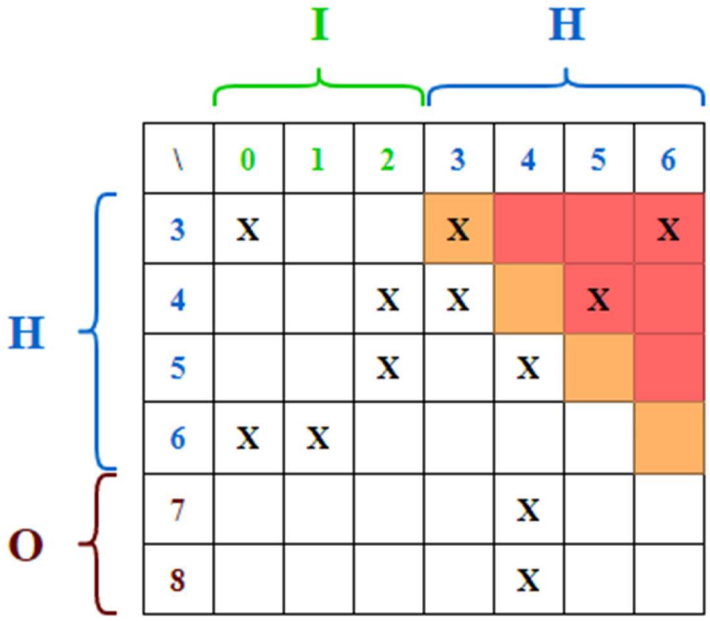
пара неурона бити рекурентне). Један начин је да се направи нова матрица индикатора (енгл. *flag matrix*) која би означила рекурентне везе. Међутим, постоји још један начин. С обзиром да је број рекурентних веза у мрежи једнак $\frac{n\cdot(n-1)}{2}$, а како је ово једнако броју елемената у горњем или доњем троуглу изнад, односно испод главне дијагонале матрице (па чак и споредне), управо ове везе се могу пгорлагисити рекурентним. Конкретно, узима се троугао изнад горње главне дијагонале.

Ово повлачи мноштво добрих особина. Наиме, овде је испуњен услов могућности проглашења таквих веза као рекурентним. Поред тога, ако се неурони нумеришу као на сликама 38 и 39, то значи да за сваку везу од неурона j до неурона i, веза је рекурентна ако и само ако је $j \geq i$. Другим речима, ако се ради о вези где је неурон из ког веза излази нумерисан мањим бројем од неурона ка ком се веза простире, веза је директна, иначе је рекурентна. Ово не само да представља веома ефикасан метод детекције рекурентних веза, него се и на врло лак начин такве везе могу санкционисати и тиме рекурентне везе забранити (уколико је то баш и потребно). Може се, међутим, поставити питање да ли ово повлачи и лоше особине. Наиме, на сликама 38 и 39, у овим ситуацијама, веза од неурона 6 ка неурону 3 би, уместо директне, сада постала рекурентна, иако до сада уопште није ни разматрана као рекурентна, већ је од почетка деловала као директна. Ипак, ово не би требало да представља проблем, јер као што се и раније дискутовало, овим методом се не преферирају ни директне ни рекурентне, а како је њихов број једнак (изузев сопствених веза које имају ипак мало другачије особине па се тиме и могу издвојити), самим тим овај аргумент нема толику тежину (после ће се и показати да је број реалних рекурентних веза ипак доста мањи, па је њихова фаворизација чак и добра особина како би се ситуација нормализовала).

Поред тога, још једна позитивна особина овог избора фиксирања рекурентних веза је та да се било које везе које се простиру ка улазним неуронима сматрају рекурентним, а како је раније дискутовано да овакве везе немају смисла (јер улазни неурони не врше никакво процесирање), директне везе од улазних неурона су безбедне и нетумаче се погрешно као рекурентне. Такође, све везе које се простиру од излазних неурона ка другим се такође сматрају рекурентним (супротне и треба да буду директне). Наиме, ове рекурентне везе од излазних неурона и могу представљати смислене структуре, али се може укључити наредни приступ. Сврха излазних неурона је да прикупе коначне податке процесирања и да их као одговарајућу одлуку представе кориснику саме мреже. Стога, може се једноставно забранити да постоје везе које излазе из излазних неурона. Овим се и поједностављују ситуације где излазни неурони могу међусобно бити повезани.

Питање се може поставити да ли ипак рекурентна веза која иде од излаза носи добре особине (јер су управо претходно овакве везе санитизиране). Међутим, увек може постојати онолико скривених неурона колико има излазних који се налазе одмах пре излазних и повезани су „1-1“, тако да практично ови скривени неурони глуме „излазне“ од којих се могу развијати повратне (рекурентне) гране као што је претходно постављено. Уколико ово носи позитивне особине за сам проблем, сам *NEAT* алгоритам би овакву структуру требало да развије. Практично, у потпуности се могу избацити све везе које се простиру ка улазним и које одлазе од излазних неурона, што се манифестује у скраћењу матрице за све редове које представљају управо везе које се простиру ка

улазним неуронима, као и све колоне које представљају везе које одлазе од излазних неурона, па се добија представа као на слици 40.



Слика 40 – Коначна представа матрице динамичког слоја

Са слике 40 се ипак може видети да постоји много мање рекурентних веза него директних, укључујући и сопствене, па се тиме показује да ипак рекурентне везе не могу да преовладају у оквиру еволуције. Сопствене су означене наранџастом бојом, док су остале рекурентне означене црвеном. Наиме, рекурентне везе су овде дозвољене само у оквиру скривених неурона, што заправо представља очекивану особину.

Решење са коришћењем овако фиксираним рекурентним везама има још једну изузетну особину. Преостали проблем у низу претходно наведених јесте бирање редоследа рачунања излаза неурона. Интересантно је да у оваквој представи, могуће је рачунати излазе неурона истим редом како су и нумерисани (дакле, „редом“ их рачунати). И директне и рекурентне везе ће се природно уклопити у овај начин рачунања, па тиме није ни потребно испитивати које су везе директне, а које рекурентне! Ово се такође уклапа и са принципом „такта“. На пример, посматрајмо неуроне 4 и 5. Првенствено се рачуна неурон 4. С обзиром да неурон 5 још увек није срачунат, од везе 5 ка 4 добиће стари податак из претходног тренутка (што је и добро јер ово представља рекурентну везу). Затим ће се рачунати излаз неурона 5. Неурон 5 ће од везе 4 ка 5 добијати управо нов, свеже срачунат податак који представља податак из текућег тренутка, што је и очекивано јер ова веза представља директну везу. На исти начин, било каквом анализом било какве мреже овог типа, добија се да је један од могућих редоследа рачунања управо растући редослед индексирања (нумерисања) неурона, чиме се заправо избегава и било каква потреба за анализом топологије мреже јер је практично увек познат редослед рачунања.

Наравно, за сваки скривени и излазни неурон, постоји и склоност која се посебно памти у вектору, тако да сваки елемент вектора практично презентује склоност једног од ових неурона. Такође, потребно је и памтити излазе скривених неурона јер само од њих

се могу формирати рекурентне везе, како би се у наредном тренутку (у наредној евалуацији) ови подаци могли користити. Овде се јавља једна интересантна ситуација. Заправо, може се поставити питање шта се дешава са првом евалуацијом, када претходног тренутка практично нема. Прва идеја је да се ове вредности (које се исто памте у једном вектору) у почетку поставе на нулу. Међутим, они се такође могу сматрати параметрима система где би алгоритам покушао да научи које је евентуално почетно стање овакве мреже оптимално из неког разлога. У овом раду се ипак ово не истражује и ове вередности се постављају на нулу.

6.2.5. Генотип

Како је сада фенотип описан на нешто другачији начин, треба поново сагледати генотип и генотипске операције. Наиме, испоставља се да се генотип може у потпуности имплементационо задржати као што је спекулисано, онако како је претходно описано, стим да се не могу генерисати поменуте везе које сада не постоје (треба само кроз мутацију додати ово ограничење).

Међутим, начин фиксирања рекурентних веза и сама нумерација неурона за собом повлачи још додатне позитивне особине које се простиру кроз генотип, па тиме и генотипске операције. Претходно су објашњавани историјски маркери као један веома моћан концепт за праћење гена гроз генерације. Они практично и представљају срж *NEAT* алгоритма. Међутим, сада се може извести једно унапређење.

Сами *Node* гени у *NEAT* алгоритму немају потребу да буду маркирани на начин како се то обавља код *Connection* гена. Међутим, ово је заправо отворено питање, а када би се ипак одлучило да се и *Node* гени маркирају, ту се може поставити мноштво питања око појединих имплементационих детаља, као и да ли дате имплементације истих детаља имају смисла. Стога, практично се дешава да се само конекције маркирају како би се кроз генерације гени пратили, али оно што је могуће, као што је претходно дискутовано, је да се одређена мутација деси у различитим генерацијама, а тада се она сматра као нешто потпуно различито (као што је већ речено, постоје аргументи који ово потврђују, али се и даље сматра отвореним питањем), међутим, оно што се кроз еволуцију свакако може десити је да саме стуктуре почну да дивергирају са тачке гледишта разврставања (енгл. *speciation*) у оквиру селекције, јер структуре које су потенцијално сродне не би се сматрале истим (због разлике у иновационим бројевима на структурама које су управо сродне). Такође, још један проблем овде јесте комбинација укрштања и овакве мутације. Наиме, одређена мутација може формирати нову конекцију између два неурона у једној генерацији, а затим се иста, сасвим одвојено, у наредној генерацији деси над неком независном индивидуом, и тиме ће добити други иновациони број (исти проблем као кроз претходну кратку анализу). Затим се у укрштању случајно могу наћи две индивидуе, једна која има ген од првонастале овакве конекције првом мутацијом неколико генерација пре, а друга индивидуа је у текућој генерацији мутацијом добила овакав ген, који сада након укрштања, уколико се и један и други ген наследе (с обзиром да нису поравнати), прављење фенотипа постаје двосмислено јер се не зна која од постојећа два појачања треба искористити (два гена са различитим иновационим бројевима који због природе мутације указују на исту везу, јер се мутација десила одвојено кроз различите генерације). Могуће је увек узети први или последњи такав ген, али, у тој ситуацији онда не коришћени гени постају „шкарт“.

При томе, начин на који се прате иновациони бројеви, па чак и мутације у оквиру исте генерације делују неприродно. Када настају нови гени у оквиру једне индивидуе мутацијом након укрштања, гледа се глобални број који је распрострањен кроз целу популацију.Слично и за саме мутације, ако у оквиру исте генерације поново настаје нека мутација, гледа се неки глобални контекст који прати овакве мутације. Наиме, у природи, индивидуе мутирају независно од других, па је помало концепт вештачки (што не представља нужно лошу ствар, јер је и цео овај алгоритам вештачки).

Као што је поменуто, фиксирање рекурентних веза и нумерација неурона претходне тврдње могу да потенцијално поправе (требало би заправо имплементирати и другу варијанту, односно основну варијанту *NEAT*-а и тестирати обе како би се упоредиле што у овом раду није утврђено, па се не може закључити да ли су тврдње заиста унапређења). Потребно је историјске марке посматрати из следећег угла. Они практично представљају идентификацију неког гена (енгл. *ID*). Практично, с обзиром да је читав систем стохастички, у једном поркетању, одређене конекције могу имати једне вредности маркера, а у другом потпуно друге. Конкретне вредности нису ни битне, битно је да овакве вредности постоје како би се гени поравнавали.

Како су сада неурони нумерисани, заправо, могуће је на јединствен начин, независно од других индивидуа, веома једноставно сваки ген конекиције идентификовати помоћу два броја, а то су управо индекси (нумерације) неурона од ког до ког се веза простире. У оквиру рада, с обзиром да се ради о *32-bit* бројевима, два броја се могу спојити у један *64-bit* број који ће сада представљати иновациони број овог гена. Сада, постојање глобалног иновационог броја (односно генератора) више није потребно, јер се на унапред познат начин ови бројеви генеришу (вредности су небитне, али уникатне, што је битно). Такође, и мутације које се у оквиру исте генерације дешавају, уколико су и оне исте, исти ће им се иновациони број генерисати (па нема потребе за глобалним контекстом). И не само то, већ и у наредним генерацијама, када се исте десе, добијаће исте иновационе бројеве. Ово је све захваљујући томе што се неурони практично нумеришу, односно, као да они добијају иновационе бројеве редом, али не из неког глобалног контекста, већ се индивидуално, чак и када на различитим местима истим или сличним мутацијама настане нов неурон (скривен, јер само скривени могу да настају), локално му се додели број који му је прикладан (дакле, једнак је збиру улазних и скривених неурона), а излазним неуронима се ова вредност повећава за један. Како би се у генотипу избегло да се излазним неуронима увек број повећава, прво се записују гени улазних неурона, затим гени излазних неурона, а на крају скривени (ово је иначе у реду јер се у генотипу не извршавају фенотипске операције, а како је ово мапирање „1-1“, при креирању фенотипа, излазни и скривени неурони мењају своја места како би се фенотип правилно изградио – имплементационо ни нема додатних трансформација, само се дефинише функција мапирања индекса генотипа на индексе фенотипа која је изузетно једноставна – један тринарни оператор и неколико сабирања).

Међутим, како сада гени конекција немају растући редослед, операција поравнања постаје скупа. Ипак, техника инверзног индексирања (енгл. *inverse indexing*) може дати проблем да ефикасно реши. Идеја је слична као код индекса у оквиру база података, где се практично прави одговарајуће *B+* стабло како би се претрага убрзала. Слично и овде, може се поред вектора конекционих гена, који носе све потребне информације, пратити и једна хеш табела која иновационе бројеве мапира у индексе истог елемента у низу. Наравно, елементи се не чувају двоструко (дакле и у вектору и у

хеш табели), већ се у табели само чува индекс тог елемента, преко ког се након тога долази до самих података о датој конекцији. Отуда и име за инверзно индексирање, јер се практично податком тражи индекс, али на ефикасан начин.

Може се и поставити питање да ли је чак и сам вектор неопходан, јер се практично сви елементи заправо и могу наћи у хеш табели. Имплементационо, честа је операција обиласка гена (нарочито у укрштању), па је због тога згодно имати вектор јер је он ефикаснији када је потребно проћи кроз све елементе.

У суштини, сами иновациони бројеви конекција и губе смисао, али су битни у оквиру *Node* гена, мада се ни у њима не морају чувати јер су инхерентни самом индексу у низу *Node* гена. Чак се ни тај низ не мора пратити из тог разлога јер је тачно познат редослед *Node* гена, односно, гени улазних и излазних неурона ће увек имати исте „иновационе бројеве“ без обзира на све, а скривени ће након претходних редом добијати растуће бројеве, па колико их има, дотле ће и „иновациони бројеви“ самих скривених неурона ићи. Практично, додела није потребна јер се они подударају са њиховом нумерацијом (поново, иако је у генотипу и фенотипу различита, она је компатибила и довољно је размишљати о једној, исти закључци ће важити и за другу представу). Како су сада неурони „маркирани“, гени веза се веома једноставно траже тако што се саме нумерације два посматрана неурона споје, а затим се на пример кроз хеш табелу у другом родитељу претражи оваква „марка“, па ако је она присутна, дакле и сам други родитељ садржи исти ген, и одмах се добија дакле тај исти ген са којим заправо и треба поравнање да се изврши, а ако не постоји такав податак у табели, детектована је ситуација где је добијен ген првог родитеља који нема свог пара у другом родитељу (наравно, треба погледати и друге родитеље, ако се рецимо такав ген потенцијално у њима налази).

6.2.6. Симулације

Као што је већ коментарисано, имплементација овакве варијанте *NEAT* алгоритма испитана је у оквиру трчећих агената и показала се као веома ефикасна. Једино побољшање које се и може видети јесте на тешком терену (на лаком и средњем нема сврхе дискутовати јер се и обична НЕ овде добро понаша), где је у обичној НЕ било неопходно 30 генерација како би се издвојило неколико индивидуа за које се може рећи да су успешно обучене, док је *NEAT* алгоритму потребно свега 2-7 генерација да се за и преко 150 агената у популацији може речи да су успешно обучени.

Да ли описане оптимизације заиста представљају побољшања не може се установити док се и *NEAT* у својој основној варијанти не имплементира (укључујући и начин разврставања, односно *speciation* у селекцији), а затим се перформансе истих мере и упоређују. Дефинитивно се може закључити да имплементирана варијанта има боље перформансе у оквиру једног проблема у односу на обичну НЕ, а требало би такође исто покупати и на другим.

7. Закључак

У овом раду, детаљно је описан и објашњен принцип рада неуро-еволутивних система у учењу са подршком. Првенствено је дата анализа основних концепата у неуро-еволуцији (неуралне мреже и генетички алгоритам). Затим је описана потребна инфраструктура на нижем нивоу која се манифестује у виду библиотеке, а која се даље користи као део пројеката који користе ове принципе. Потом је детаљно описан начин обучавања у неуро-еволуцији. Затим су описане симулације које представљају тестове библиотеке, а такође су дати и резултати истих. На крају, описане су напредне технике које се могу користити у неуро-еволуцији.

Истраживања овог рада су показала да се НЕ може користити као једна веома ефикасна метода у многим неконвенционалним проблемима. Угланвом су посматрани проблеми били системи где се учење одвија по принципу учења са подршком, али, такође је показано и да се НЕ може адекватно употребити у системима где се одвија надгледано учење. Поред тога, напредним техникама, могуће је реализовати и стуктурно учење. Ово, као и способност тражења глобалног оптимума, представљају главне карактеристике истраживаних принципа које указују на велик потенцијал у даљем коришћењу и истраживању.

Овај рад представља само полазну тачку истраживања методе од интереса. У самом раду је показано да се различити делови могу имплементирати на више начина, као и то да постоји одређени степен слободе где се решења могу на маштовит начин пројектовати. Често је наравно потребно нешто смислено ипак урадити, али не постоје јаке дефиниције како сами концепти треба да функционишу.

На пример, увек се може пробати са истраживањем нових операција мутација, укрштања, селекције, па и пробати различите функције способности. Што се самих НМ тиче, могу се такође имплементирати нови слојеви који би функционисали на другачији начин. Формално, битно је да се врши неко процесирање и да постоје неки параметри који могу ово процесирање да мењају. НЕ такође у овоме показује добре особине јер не обавезује да дефиниције многих ствари буду стриктне, па, иако слобода често представља лошу ствар у смислу оптерећивања због избора хиперпараметара, може представљати и добру особину за истраживање новина.

Сам ток ГА коришћен у овом раду није побољшаван. На пример, мутација у оквиру њега се само дешава тренутка када нова индивидуа (у оквиру нове генерације) настане. Наиме, природа функционише тако да се мутације непрестано дешавају. Односно, индивидуа такође мутира и у току животног века. У оквиру животног века, она и учи. С обзиром да се вид учења у досадашњем ГА такође манифестује кроз мутације, може се извести закључак да ова два концепта заправо представљају сродне појаве. Наиме, вероватно би се ово могло повезати са адаптацијом индивидуа на средину у којој актују. Током живота, у одређеним ситуацијама, могу нешто ново да науче, рецимо уколико су добили неку казну јер им је акција била лоша, адаптирали су се на такав начин да су научили да тако нешто не би требало да раде (а ово учење се манифестује кроз одговарајућу мутацију). При томе, сада се награде и казне могу моделирати на локалном нивоу, а не збирно над целим животним веком, чиме се потенцијално скривају одређене акције које се можда истичу на позитиван или негативан начин. Могла би се на одређени начин дефинисати функција мутације која диктира начин промене индивидуе (начин

адаптације?) кроз животни век. Можда би се и ова функција могла параметризовати одређеним „учећим“ параметрима, па се и кроз еволуцију оптимизовала.

Такође, начин како се генеришу нова решења у основном ГА може бити мало неприродан. Претходна генерација се у потпуности уклања, а нова наступа. Наиме, дискутовано је да рецимо чувањем најбољег решења потенцијално може да доведе систем до ране конвергенције ка локалном минимуму. Ипак, можда се може применити следећи приступ. Свака итерација алгоритма не мора да представља читаву генерацију, већ рецимо само једну годину. Свакој индивидуи се придружи старост (енгл. *age*), који се сваке године инкрементира за 1. Не прате се више генерације, већ се број индивидуа у популацији мења на годишњем нивоу (дакле популација није фиксна), по следећем закону:

$$P_{y+1} = r \cdot P_y\left(T - P_y\right)$$

- y – текућа година (енгл. *year*);
- r – брзина раста популације (енгл. *growth rate*), односно наталитет;
- P_y – број индивидуа у датој години;
- T – максимална величина популације (енгл. *total*).

Овим законом (или барем неком сличном дефиницијом) не дозвољава се да број индивидуа постане превелик, а такође ни премали, а да ипак кроз године постоји промена у популацији (закон практично говори колике величине ће бити популација у следећој години, у зависности од величине у текућој). Ако је број индивидуа у наредној години мањи, одумиру изабране индивидуе по неком критеријуму, рецимо, најстарије, или комбинација старости и способности, где се способност дели са старошћу, или да се ова вредност сада користи као пропорционална шанса у избору или нешто слично. У супротно, уколико је број индивидуа већи, нове настају, на неке сличне начине као што су управо описани. Такође, може се увести да, иако рецимо је нов број индивидуа мањи, ипак неке нове настају, али због тога више постојећих умире како би се број у оквиру популације правилно кориговао.

Сама функција способности представља детаљ који је веома важан и потребно му је посветити много пажње. Рецимо, можда би се у овом делу могла употребити фази логика (енгл. *Fuzzy Logics*) која би дефинисала способност индивидуе. Сама фази логика у овом контексту представља функцију способности, а често, функција способности и баш има особине које су наизглед „нејасне“, па се праве апроксимативни модели, а баш поменути „нејасан“ изглед праве функције способности представља појаву која фази логика веома добро моделира.

Поред описаних неуралних мрежа у овом раду, требало би истражити и рекурентне мреже (енгл. *Recurrent Neural Networks*). Наиме, ове мреже се углавном користе за процесирање природних језика, а обухватају слојеве као што су *LSTM*, *GRU*, *Attention* и други модели. Кроз *NEAT* технику су помало рекурентне конекције биле анализиране, а те представљају само зачетак рекурентних слојева који се могу накнадно истраживати.

Можда и прво наредно унапређење рада може представљати спој конволуционих неуралних мрежа у описаним УП проблемима. Наиме, било би интересантно уместо ручног обављања издвајања битних одлика (енгл. *feature extraction*), направити систем где се користе конволуционе мреже, а да се мрежама дају слике фрејмова саме

симулације (практично пиксели). Овде би агенти практично добили праву визију, и сами требали да на основу слике доносе добре одлуке, па тиме практично уједно и вршили одабир битних одлика. За ово вероватно треба јачи хардвер јер обрада оваквих слика представља веома захтевну операцију (слике су већих димензија), као и то да је сада немогуће пуштати агенте паралелно да актују јер ће утицати једни на друге.

Такође, *NEAT* алгоритам би могао да се генерализује на друге слојеве, тако што би мењао њихове хиперпараметре (нпр. величину потпуно повезаних слојева, као и друге хиперпараметре конволуционих слојева), па чак и до тога да убацује/искључује читаве слојеве у самој мрежи. Поред тога, комбинација динамичког слоја са другим би такође могла да потенцијално покаже добре резултате, где би претходно описан начин управљања читавих слојева представљао грубу претрагу, а *NEAT* над динамичким слојем затим представљао фину (енгл. *fine tuning*).

Сама библиотека, с обзиром да је писана на језику *C++*, могла би се употребити и у другим технологијама ван *.NET*, као што су *javascript*, *python* и други. За ове језике такође би било згодно направити неки вид омотача као што је то реализовано у *C#* језику, како би се даље библиотека ефективно користила и кроз, на пример, *web* технологије.

Литература

[1] Општа електронска енциклопедија *Wikipedia*, чланак о вештачкој интелигенцији, линк: https://en.wikipedia.org/wiki/Artificial_intelligence

[2] Општа електронска енциклопедија *Wikipedia*, чланак о *AlphaGo* систему, линк: https://en.wikipedia.org/wiki/AlphaGo

[3] Општа електронска енциклопедија *Wikipedia*, чланак о истраживачкој лабораторији *DeepMind*, линк: https://en.wikipedia.org/wiki/DeepMind

[4] Општа електронска енциклопедија *Wikipedia*, чланак о *Turing* тесту, линк: https://en.wikipedia.org/wiki/Turing_test

[5] *MIT News*, Неуралне мреже, појам и опис, линк: https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414

[6] *TowardsDataScience*, Увод у генетички алгоритам, идеја и објашњење, линк: https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3

[7] *Expert System Team*, *„What is Machine Learning? A definition“*, блог, линк: https://expertsystem.com/machine-learning-definition/

[8] *Risto Miikkulainen*, *Scholarpedia*, неуро-еволуција у машинском учењу, линк: http://www.scholarpedia.org/article/Neuroevolution

[9] *DeepSense AI*, *„What is reinforcement learning? The complete guide“*, блог, линк: https://deepsense.ai/what-is-reinforcement-learning-the-complete-guide/

[10] Сажетак и документација машинског учења, типови активационих функција, линк: https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html

[11] *Live Science*, *„Darwin's Theory of Evolution: Definition & Evidence“*, линк: https://www.livescience.com/474-controversy-evolution-works.html

[12] Паралелно рачунање, *Nvidia Cuda Toolkit,* Софтверски и развојни пакет, линк: https://developer.nvidia.com/cuda-zone

[13] Пакет за развој игара, *Unity3D Technologies*, линк: https://unity.com/

[14] Општа електронска енциклопедија *Wikipedia,* чланак о тензорима, линк: https://en.wikipedia.org/wiki/Tensor

[15] Игра *Flappy Bird*, линк: https://flappybird.io/

[16] *Jason Brownlee*, *Machine Learning Mastery*, *„What is Deep Learning“*, линк: https://machinelearningmastery.com/what-is-deep-learning/

[17] *Jason Brownlee*, *Machine Learning Mastery*, *„A Gentle Introduction to Computer Vision“*, линк: https://machinelearningmastery.com/what-is-computer-vision/

[18] *Sumit Saha*, *„A Comprehensive Guide to Convolutional Neural Networks – the ELI5 way“*, линк: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[19] *Margaret Rouse*, *OCR* (*Optical Character Recognition*), чланак и дефиниција, линк: https://searchcontentmanagement.techtarget.com/definition/OCR-optical-character-recognition

[20] *Jason Brownlee*, *Machine Learning Mastery*, *„Supervised and Unsupervised Machine Learning Algorithms“*, линк: https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/

[21] *Google Inc.*, *Quick-Draw Doodling Dataset*, скуп података намењен за истраживање у машинском учењу, линк: https://quickdraw.withgoogle.com/

[22] *Kenneth O. Stanley*, *Risto Miikkulainen*, *„Evolving Neural Networks through Augmenting Topologies“*, линк: http://nn.cs.utexas.edu/downloads/papers/stanley.ec02.pdf

[23] Проф. Др. Драган Олћан, ЕТФ, Инжењерски Оптимизациони Алгоритми, линк: http://mtt.etf.rs/si/ioa.htm

[24] Проф. Др. Бошко Николић и Доц. Др. Дражен Драшковић, ЕТФ, Проналажење Скривеног Знања, линк: http://rti.etf.bg.ac.rs/rti/ms1psz/

[25] Проф. Др. Горан Квашчев и Проф. Др. Александар Ракић, ЕТФ, Неуралне Мреже, линк: http://automatika.etf.bg.ac.rs/sr/13s053nm

Списак скраћеница

ВИ	Вештачка Интелигенција
НМ	Неурална Мрежа
ГА	Генетички Алгоритам
НЕ	Неуро-Еволуција
МУ	Машинско Учење
УП	Учење са Подршком
ЕА	Еволутивни Алгоритам