МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Навчально-науковий інститут комп'ютерних інформаційних технологій
Кафедра комп'ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
Савченко А.С.

"____"_____2020 р.

ДИПЛОМНИЙ ПРОЕКТ

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСНИКА ОСВІТНЬОГО СТУПЕНЯ
"МАГІСТР"

Тема: "Штучна нейронна система з використанням генетичних алгоритмів для її тренування"

Виконавець: Васильченко Корній Дмитрович
Керівник: професор Зіатдінов Юрій Кашафович
Нормоконтролер: Райчев Ігор Едуардович

Київ 2020

РЕФЕРАТ

Пояснювальна записка до дипломної роботи "Використання генетичних алгоритмів для тренування штучних нейронних мереж": 50 с., 22 рис., 12 інформаційних джерел.

ГЕНЕТИЧНІ АЛГОРИТМИ, НЕЙРОННІ МЕРЕЖІ, НАВЧАННЯ НЕЙРОННИХ МЕРЕЖ, ШТУЧНИЙ ІНТЕЛЕКТ.

Об’єкт розробки – процес навчання нейронної мережі.

Предмет розробки – автоматизована система навчання нейронних мереж на основі генетичних алгоритмів.

Мета роботи – покращення способів розробки та тренування нейронних мереж для подальшого їх використання у якості штучного інтелекту.

Задачі дослідження – дослідити типи ШНМ; проаналізувати існуючі методи тренування нейронних мереж; дослідити ефективність навчання за допомогую генетичних алгоритмів; розробити автоматизовону систему для навчання ігрових персонажів на основі генетичних алгоритмів.

ПЕРЕЛІК ПРИЙНЯТИХ СКОРОЧЕНЬ

БД - база даних

ШНМ - штучна нейронна мережа

ЗНМ - згорткова нейронна мережа

СУБД - система управління базами даних

ЕОМ - електронна обчислювальна машина

АРМ- автоматизоване робоче місце

ПК - персональний комп’ютер

UI - user interface або інтерфейс користувача

ЗМІСТ

ВСТУП

Одними з найважливіших областей досліджень і розробок сучасної кібернетики є області машинного навчання та штучного інтелекту, які широко використовуються для вирішення досить великого спектру задач: від прогнозування і класифікації образів до більш складних таких як системи прийняття рішень.

В дипломному проекті досліджується використання звичайних багатошарових нейронних мереж в контексті їх навчання за допомогою генетичних алгоритмів для моделювання поведінки мікроорганізмів в замкнутому середовищі. Такі задачі досить важко вирішувати за допомогою звичайних “не гнучких” алгоритмів, так як вони можуть потребувати коригування параметрів, а бо навіть повної заміни алгоритму в залежності від параметрів моделюмого середовища і тому дана робота ставить за ціль розробити евристичний нейроеволюційний алгоритм для вирішення даного класу проблем моделювання.

РОЗДІЛ 1 НЕЙРОННІ МЕРЕЖІ

1.1. Штучна нейронна мережа

Штучна нейронна мережа є дуже спрощеною моделлю звичайної біологічної нейронної мережі, що зустрічається в природі, і вона складається з розташованих у різних топологіях шарів штучних нейронів, що мають в собі певну активаційну функцію і які об’єднано у одне ціле за допомогою з’язків, що мають певні вагові коефіцієнти. Нейрон у нейронній мережі визначається виходом функції, що враховує сигнал від інших нейронів помножений на вагові коефіцієнти зв’язків. З самого початку розвиток технології ШНМ відбувався досить інакшим від канонічних способів шляхом, досить часто повністю піддаючи сумніву сучасне уявлення сукупної проблематики теорій глубокого навчання, задач прогнозування і розпізнавання патернів, в той же час створюючи суттєвий вплив на теорію, термінологію і методологію даних дисциплін. Через певний час після першого розвитку простих моделей ШНМ, сталися значні події що стосувалися науки про нейромережі та види топологій архітектур мереж і методи їх навчання.

1.2. Архітектура ШНМ

У більшості архітектур ШНМ функції активації нейронів є статичними, а вагові коефіцієнти штучних синапсів є змінними параметрами мережі, що коригуються при тренуванні. Буває так, що певні входи конкретних нейронів є зовнішніми входами всієї мережі, а деякі виходи нейронів, відповідно, виходами мережі.

Основна задача нейромережі заключається в перетворенні вхідного вектора даних у вихідний вектор, що відбувається при проходженні сигналу через вагові коефіцієнти і топологію мережі. Існують різні класифікації ШНМ за рядом ознак. По одному з топологічних ознак ШНМ можна класифікувати як:

• Повнозв'язні ШНМ - кожен нейрон пов'язаний з іншими нейронами в мережі, включаючи себе самого

• Багатошарові ШНМ - нейрони об'єднуються в шари, нейрони попереднього шару пов'язані з нейронами наступного шару

• Слабозв'язані ШНМ - нейрони розташовані в вузлах прямокутної або гексагональний решітки

1.3. Види багатошарових штучних нейронних мереж

Різниця процесів обчислення в нейронних мережах дуже часто залежать від її топології. За низкою параметрів на цей час багатошарові топології ШНМ можна виділити: статичні і динамічні. Кожен вид нейронної мережі може мати сотні різних підвидів. Мережі з алгоритмом прямого поширення, в яких немає динамічних компонентів і де існує тільки односторонній зв’язок між шарами мережі, а вихідний вектор повністю задається вхідним і ніяким чином не може залежати від минулах станів або входів мережі - такі мережі відносяться до статичних. Прикладом статичних архітектур ШНМ може бути:

- Перцептрон
- Нейронна мережа Кохонена
- Когнітрон

- Згорткова нейронна мережа
- Звичайний неокогнітрон

І звичайно, як протилежність статичним, існують динамічні версії ШНМ, які мають рекурентну топологію, що включають в себе механізм зворотних зв'язків, завдяки чому стан структури в кожний наступний момент часу T залежить від минулого стану. Рекурентні ШНМ як правило є одним із видів перцептрона. Динамічні рекурентні ШНМ з зворотними зв'язками:

- мережа Хопфілда
- мережа Коско
- мережа Джордана
- мережа Елмана

1.3.1. Перцептрон елементарний

Структура перцептрона створена на основі сенсорних даних, що подаються на вхід мережі. Основними сутностями є: S-елементи, асоціативні елементи - A-елементи, і елементи, що реагують на виході - R-елементи. S-елементи пов'язані з A-елементом утворюють так звану “асоціацію”, і A-елемент вмикається одразу як тільки певне число сигналів від S-елементів поступає на вхід. A-елемент направляє сигнал з урахуванням вагових коефіцієнтів на R-елемент що робить зважену суму сигналів, і дивлячись на те, чи перевищує зважена сума певний поріг, R-елемент формує рішення роботи перцептрону (Рис.1.1.). Перцептрони які мають багато шарів будуються з додаванням прихованих шарів із A-елементів, розташованими між S-елементами і R-елементами.

Ступінь складності проблем, що можна вирішити за допомогою багатошарового перцептрона, є найвищою для класу перцептронів. Навчання елементарного і багатошарового перцептрона полягає в ітеративному коригуванні

вагових коефіцієнтів зв'язків між шарами A і R. Перцептрон здатний працювати в режимі розпізнавання або узагальнення.

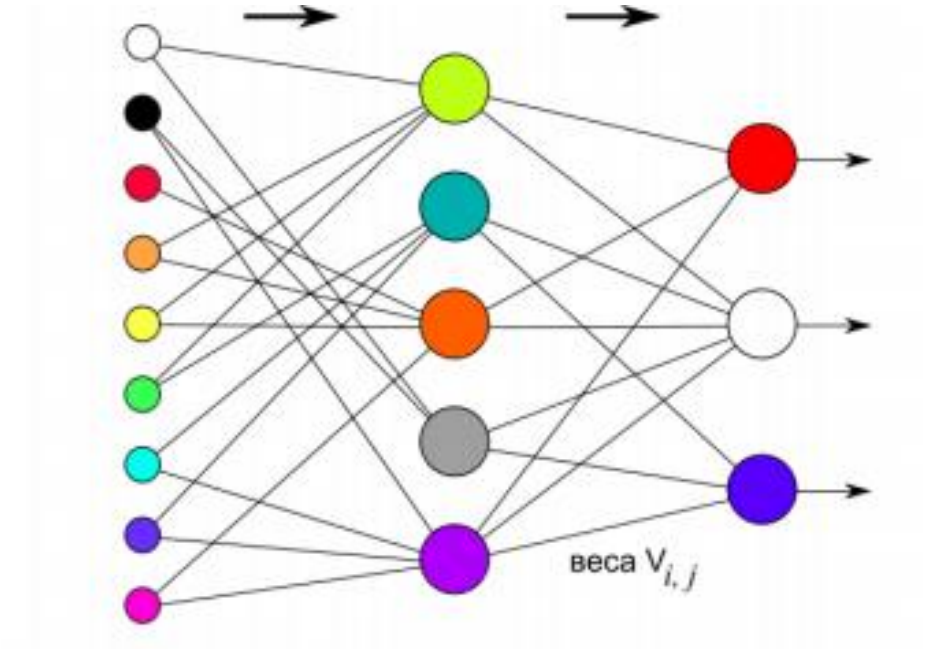


Рис.1.1. Перцептрон

В простому перцептроні нейрони, що є вхідними мають зв’язки напряму з вихідними нейронами, S <--> A зв’язки існують за принципом прямої відповідності. Перцептрон у якого всього 1 шар є просто окремим випадком елементарного перцептрона, але має велику кількість обмежень, наприклад, неможливість вирішувати певні класи задач такі як функція “Виключного АБО”.

1.4. Топологія згорткових нейронних мереж

Сучасні глибокі згорткові нейронні мережі засновані на ідеях, що лежать в основі неокогнітрона, і сьогодні застосовуються для вирішення широкого кола

завдань: від промислових, корпоративних та дослідницьких до повсякденних побутових, включаючи завдання, які вирішуються мобільними пристроями.

Згорткова нейронна мережа, що в перекладі з англійської Convolutional Neural Network являє собою особливу архітектура ШНМ, що намагається імітувати функції зорової кори мозку людини, що включає в себе декілька багатовимірних шарів і призначена для ефективного розпізнавання складних зображень. Вперше така модель була запропонована Яном Лекуном (Yann Lecun) в 1998 році і призначалася для розпізнавання рукописних символів. Особливість запропонованої Лекуном моделі нейронної мережі полягала у введенні в архітектуру багатошарового перцептрона згорткових шарів, в яких кожен нейрон був пов'язаний тільки з невеликою областю нейронів попереднього шару. Така особливість дозволяла виділяти на оригінальному документі примітивні особливості, а на наступних шарах мережі, об'єднуючи виділені ознаки, отримувати все більш складні елементи таким чином детектуючи складні об'єкти на зображеннях.

Згорткові нейронні мережі можуть мати багатовимірні шари (в основному використовуються двомірні, наприклад, в мережах, що обробляють зображення, і тривимірні, наприклад, додавання декількох колірних каналів для зображення) декількох типів:

- Вхідний шар: вхідне зображення, включаючи кілька колірних каналів.
- Згортковий шар (*Convolution*): всі нейрони шару, на відміну від перцептрона, пов'язані тільки з частиною нейронів попереднього шару.
- Шар субдискретизації (*Pooling, Subsampling*): виділення найбільш значущих ознак попереднього шару і значне скорочення розмірності наступних шарів мережі.
- Повнозв'язний шар (*Fully-connected*): являє собою прихований шар штучної нейронної мережі типу перцептрон.

Також, після згорткового шару і повнозв'язного шару може застосовуватися функція активації нейрона, яка перетворює сигнал нейрона і формує вихідний сигнал.

Нижче всі перераховані типи шарів описуються більш детально. Вхідний шар. Найчастіше у задачах розпізнавання образів та патернів у двомірному векторі вхідний шар подається у вигляді 3-вимірної сітки розміри якої залежать від розміру вхідного вектору - формула (1.1).

$$In = W * H * D \qquad (1.1)$$

де In - розмірність вхідного шару мережі;

W - ширина вхідного вектору; H - висота вхідного вектору;

D - глибина або кількість колірних каналів зображення.

Згортковий шар. Одним з найголовніших шарів штучної нейронної мережі є згортковий шар, який потрібен для класифікації певних ознак зображення та їх трансформації, і які, після чого, на наступних шарах, використовуються щоб отримати складніші ознаки для визначення класу розпізнаного об'єкта.

Важливою ознакою такого згорткового шару є фільтри - що являють собою 2-мірні чи 3-вимірні матриці ваг зв'язків нейронів попереднього шару з нейронами згорткового шару. Вони носять назву фільтри тому, що операція згортування - отримання вихідного сигналу нейрона згорткового шару - має певні схожості з операцією фільтрації зображення: кожне значення сигналу нейрона попереднього шару, розташованого в певній галузі, відповідної ядру фільтра, множиться на відповідне значення ядра фільтра (в ЗНМ значення ядра фільтра називаються вагами зв'язків нейронів згорткового шару: w_{00} , w_{01}, ... $w_{(F-1)}$, $w_{(F)}$, де F - розмір квадратного фільтра); а результатом фільтрації є сума всіх отриманих творів [5]. Процес формування сигналів нейронів сверточного шару показаний на рис. 1.2.

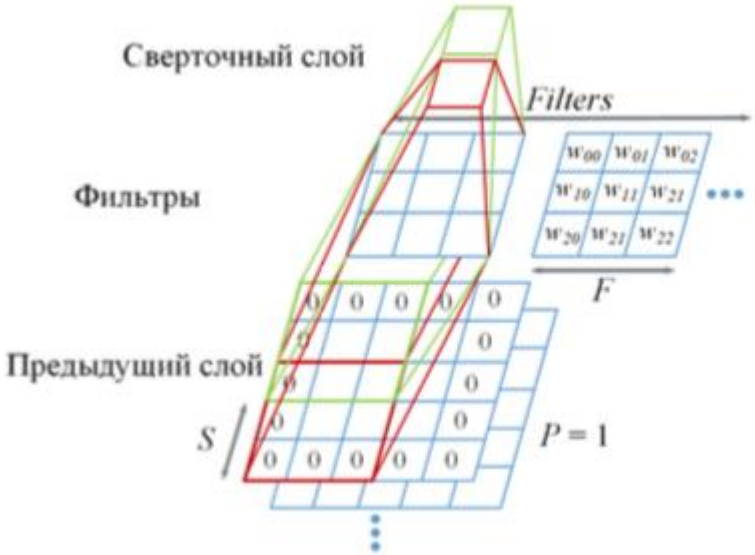


Рис. 1.2. Формування згортального шару згортувальної нейронної мережі

Іншими важливими параметрами згортального шару мережі є тип і кількість використовуваних фільтрів. Тип фільтра визначає, для виділення яких ознак зображення він використовується, наприклад, виділення прямих ліній під певним кутом (в процесі навчання мережі ваги фільтрів змінюються, впливаючи на характер виділяються ними ознак). Зазвичай використовуються квадратні фільтри, розміром F. Кількість використовуваних фільтрів визначає глибину сверточного шару, яка позначається на рисунку 1.2. як filters.

Також згортковий шар має і інші параметри:

S - зміщення (stride) - визначає на скільки позицій зміщується фільтр для формування сигналу наступного нейрона згорткового шару.

P - додавання нулів (padding) - додавання нейронів з нульовими значеннями, що не надають вплив на формування сигналу згорткового шару, по краях попереднього шару для регулювання розмірності згорткового шару мережі.

Розмірність згорткового шару визначається його параметрами і розмірами попереднього шару мережі, відповідно до формул (2) і (3).

$$W = \frac{W_p - F + P*2}{S} + 1, \qquad (1.2)$$

$$H = \frac{H_p - F + P*2}{S} + 1 \qquad (1.3)$$

де W і H - ширина і висота згорткового шару відповідно;

W_p і H_p - ширина і висота попереднього шару відповідно;

F - розмір квадратного фільтру сверточного шару мережі;

P - кількість доданих нулів по краях попереднього шару;

S - зміщення фільтра.

Для формування вихідного сигналу нейронів згорткового шару в ЗНМ використовуються функції активації нейрона. Дані функції за певними правилами перетворять сигнали нейронів. У ЗНМ найчастіше використовуються три види функцій активації: порогова - формула (1.4), сігмоіднf - формула (1.5) і тангенс - формула (1.6).

$$y = max(0, x), \qquad (1.4)$$

$$y = \frac{1}{1+e^{-x}}, \qquad (1.5)$$

$$y = tg(x), \qquad (1.6)$$

Шар субдискретизація. Даний шар згорткової нейронної мережі використовується для скорочення розмірності даних з метою зменшення ймовірності швидкого перенавчання, а також для скорочення обчислювальних витрат і витрат пам'яті. Зазвичай даний шар використовується після проведення операції згортки і перетворює сигнали сверточного шару, виділяючи найбільш значущі, за певними критеріями.

В даному шарі використовується вікно певного розміру (U) для виділення областей нейронів попереднього шару, які будуть пов'язані з нейроном шару Субдискретизація. Потім, за певними правилами вибирається і розраховується один сигнал нейрона шару Субдискретизація. Найбільш типові способи формування сигналу даного шару - це або шляхом знаходження максимального значення серед сигналів попереднього шару (max), що знаходяться в межах вікна Субдискретизація; або шляхом розрахунку середнього значення (avg). Процес формування сигналів шару Субдискретизація показаний на рисунку 1.3.

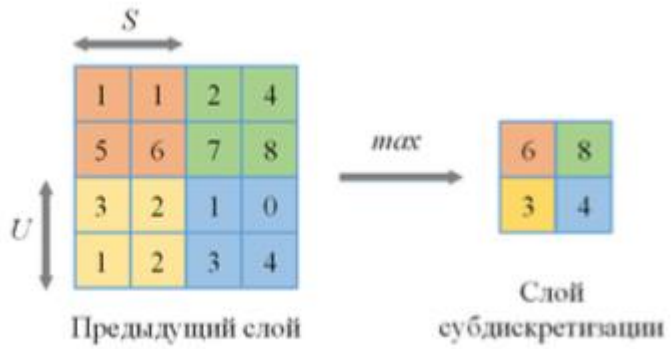


Рис.1.3. Формування шару Субдискретизація за допомогою операції max

Розмірність шару Субдискретизація визначається його параметрами і розмірами попереднього шару мережі, відповідно до формул (1.7) і (1.8).

$$W = \frac{(W_p - U)}{S} + 1, \qquad (1.7)$$

$$H = \frac{(H_p - U)}{S} + 1, \qquad (1.8)$$

де W і H - ширина і висота шару Субдискретизація відповідно;

W_p і H_p - ширина і висота попереднього шару відповідно;

U - розмір квадратного вікна шару Субдискретизація мережі;

S - зміщення вікна шару Субдискретизація.

Важливим параметром шару Субдискретизація є розмір вікна U. Чим він більший, тим сильніше руйнівну дію операції Субдискретизація на виділені на сверточное шарі ознаки, чим він менший, тим сильніше перенавчання мережі.

Повнозв'язний шар. Даний шар є одновимірним і в ньому кожен нейрон пов'язаний з кожним нейроном попереднього шару на всіх рівнях, якщо у попереднього шару присутній параметр глибини.

Основне призначення даного шару - перетворення сигналів, отриманих на згорткових рівнях мережі до одновимірного увазі і виділення ознак на одновимірному рівні. Даний шар також може використовуватися в якості останнього (вихідного) шару ЗНМ, результатом якого є ймовірність приналежності вхідного зображення певного класу. Схема даного шару представлена на рисунку 1.4.

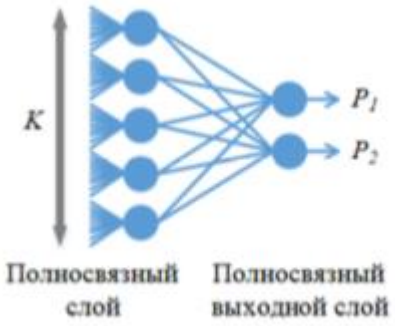


Рис. 1.4. Два повнозв'язних шари згорткової нейронної мережі, один з яких використовується в якості вихідного шару

Єдиним параметром даного шару є кількість нейронів K. Зазвичай воно вибирається виходячи з розміру попереднього шару і кількості класів (необхідної кількості виходів) мережі.

1.5. Навчання згорткової нейронної мережі

Навчанням штучної нейронної мережі називається процес підстроювання значень ваг зв'язків між нейронами мережі. У ЗНМ використовується навчання з учителем, що припускає використання навчальної вибірки для порівняння вихідного сигналу мережі з еталонним значенням навчальної вибірки, розрахунок помилки і підстроювання ваг зв'язків мережі з метою зменшення значення цієї помилки. У ЗНМ як алгоритм навчання використовується алгоритм градієнтного спуску або зворотного поширення помилки і його модифікації. Для повнозв'язних і згорткових шарів розраховується значення помилки вихідного сигналу мережі за загальною формулою (9).

$$E(\omega) = \frac{1}{2}\sum_{i,k} (f_{ik} - y_{ik})^2, \qquad (1.9)$$

де *E (w)* - функція помилки мережі;

f_{ik} - значення вихідного сигналу k-го нейрона мережі при подачі і-го зразка навчальної вибірки;

y_{ik} - очікуване значення вихідного сигналу k-го нейрона мережі при подачі і-го зразка навчальної вибірки.

Навчання мережі направлено на мінімізацію функції помилки. Значення функції помилки для шару мережі визначається за загальною формулою (10).

$$\delta_i^{(q)} = (f_{ik}^{(q)}(S))' \sum_j w_{ij}\delta_j^{(q+1)} \qquad (1.10)$$

де $\delta_i^{(q)}$ - помилка і-го нейрона в шарі q;

$(f_{ik}^{(q)}(S))'$- похідна функції активації і-го нейрона шару q для k- го зразка навчальної вибірки;

S - сигнал, що подається на вхід і-го нейрона мережі;

w_{ij} - вага зв'язку, що з'єднує і-ий нейрон шару q та j-ий нейрон шару ($q + 1$),

$\delta_j^{(q+1)}$ - помилка j-го нейрона в шарі ($q + 1$).

На основі розрахованої помилки нейрона визначається зміна ваги зв'язку цього нейрона за формулою (1.11)

$$\Delta w_{ij}^{(q)} = -\eta \delta_j x_i, \quad (1.11)$$

де $\Delta w_{ij}^{(q)}$ - значення зміни ваги зв'язку, що з'єднує і-ий нейрон шару q і j-ий нейрон шару ($q + 1$);

η - значення, що визначає швидкість навчання, j - значення помилки j-го нейрона в шарі q;

x_i - значення і-го вхідного сигналу *[7]*.

У згорткових шарах ЗНМ вищенаведені формули використовуються для настройки ваг зв'язків для кожного фільтра.

Для шарів субдискретизації розрахунок помилки не проводиться, тому що дані шари не беруть участі в навчанні мережі. При використанні функції шару субдискретизації max (вибір максимального значення) помилка з шару мережі, розташованого після цього шару, відразу переміщається на згортковий шар мережі, що передує даному шару субдискретизації, на вихідне значення згорткового шару, яке відповідає «виграв» значенням субдискретизації шару. При використанні функції шару субдискретизації avg (розрахунок середнього значення) помилка з шару мережі, розташованого після цього шару, ділиться на кількість елементів вікна шару і переноситься на всі значення згорткового шару, що передує даному.

1.6. Алгоритм формування ефективної архітектури згорткової мережі

Завдання побудови архітектури згорткової нейронної мережі для класифікації вхідного кольорового зображення зводиться до «згортання» вхідного шару мережі до верствам з найменшими розмірами, наприклад, 2 x 2 x D або 1 x 1 x D, де D - глибина шару, і подальшого перетворення до C одновимірним сигналам ймовірності приналежності зразка, що надійшов на вхід мережі, до одного з C класів.

Пропонований підхід являє собою алгоритм, який регламентує вибір параметрів архітектури ЗНМ виходячи з основних характеристик вхідних даних на кожному етапі послідовного формування шарів мережі. Алгоритм передбачає опціональне введення основних параметрів мережі і випадкове завдання деяких значень параметрів, що означає, що результатами алгоритму в результаті його виконання для різних вхідних даних може бути набір архітектур ЗНМ, кожна з яких буде ефективною і буде має високі показники точності класифікації на вихідних даних .

В описі алгоритму прийняті наступні позначення:

N - розмір квадратного попереднього шару мережі (для першого згорткового шару є розміром вхідного шару).

D - глибина вхідного шару мережі (кількість колірних каналів зображення)

C - кількість класів, належність до яких визначається класифікатором.

P - кількість рядків і стовпців, що містять нулі, що додаються до межами шару, що передує згортковому шару (параметр архітектури згорткового шару).

S - зміщення між фільтрами при формуванні сигналів нейронів згорткового шару / зсув між вікнами при формуванні сигналів шару Субдискретизація.

F - розмір квадратних фільтрів згорткового шару, причому F може приймати значення 3, 5 або 7.

Filters - глибина згорткового шару (кількість фільтрів).

Filtersp - глибина попереднього згорткового шару (глобальний параметр циклу формування шарів мережі).

AFc - функція активації нейронів згорткового шару.

U - розмір квадратного вікна для шару Субдискретизація.

Subf - тип функції шару Субдискретизація (max - максимум, або avg - розрахунок середнього).

K - кількість нейронів в повнозв'язну шарі.

AFfc - функція активації повнозв'язного шару

Рівень мережі - актуальна послідовність з n згорткових шарів і m шарів Субдискретизація, причому для рівня дані змінні приймають цілочисельні значення з діапазонів: n [1; 2], m [0; 1].

d - глибина мережі - кількість рівнів мережі (глобальний параметр циклу формування шарів мережі).

Текстовий опис етапів алгоритму формування архітектури ЗНМ

1. Опціональне введення основних параметрів алгоритму: F - ніж великі особливості необхідно детектувати на зразках даних, тим більше рекомендується ставити значення F (за замовчуванням F не ставить), параметри рівня мережі: n, m - чим складніше зображення, тим більше рекомендується ставити значення n (за замовчуванням n = 1, m = 1).

2. Формування параметрів вхідного шару мережі: введення розміру вхідного зображення, що має квадратну форму, зі значенням розміру боку, рівним N, причому N має багаторазово ділитися на 2 аж до однозначних чисел. Введення глибини вхідного шару D, зазвичай рівного кількості колірних каналів зображення. Введення значення кількості класів C.

3. Ініціалізація глобальних змінних алгоритму: d = 0, Filtersp = 0.

4. Ініціалізація змінної циклу формування згорткових шарів in = 0.

5. Якщо значення in не дорівнює n, то проводиться формування параметрів згорткового шару, інакше проводиться перехід до пункту 9. F вибирається залежно від N за формулою (1.12). Filters вибирається залежно від значень

параметрів F і d по формулі (1.13). Після вибору значення F, проводиться операція Filtersp = Filters.

а. Якщо in = 0, то проводиться формування першого згорткового шару, інакше перехід до пункту 5.б: P і S розраховуються шляхом вирішення системи рівнянь - формула (1.14), отриманої з формул (2) і (3) для шарів в яких ширина і висота рівні N і розмір попереднього шару дорівнює розміру згорткового шару.

$$F = \begin{cases} 7, if\,(N \geq 64) \\ 5, if\,(32 \leq N < 64) \\ 3, if\,(N < 32) \end{cases} \qquad (1.12)$$

$$\begin{cases} S = (N - F + P * 2)/(N - 1) \\ P = ((N - 1)S - N + F)/2 \end{cases} \qquad (1.13)$$

(1.14)
$$FIlters = \begin{cases} 8, if\,(F = 7 \,\&\&\, d = 0) \\ 16, if\,(F = 5 \,\&\&\, d = 0) \\ 24, if\,(F = 3 \,\&\&\, d = 0) \\ Filters_p \,||\, 1{,}25 * Filters_p, if\,(d > 0) \end{cases}$$

б. Проводиться формування параметрів інших згорткових шарів: P вибирається мінімальним цілочисельним з відрізка [0; F] таким чином, щоб розміри згорткового шару були цілочисельними відповідно до формул (2) і (3). S вибирається мінімальним цілочисельним з відрізка [1; F] таким чином, щоб розміри згорткового шару були цілочисельними відповідно до формул (2) і (3).

6. Розрахунок N для сформованого згорткового шару за формулою (15), отриманої з формул (1.2) і (1.3), де ширина і висота зображення рівні.

$$N = \frac{(N_p\, F + \, P * 2)}{S} + \, 1 \qquad (1.15)$$

де *Np* - розмір квадратного попереднього шару мережі.

7. Вибір в якості функції активації нейронів згорткового шару порогової функції, згідно з формулою (1.16).

$$AF_c = max(0, x), \qquad (1.16)$$

де *x* - сигнал нейрона згорткового шару, отриманий в результаті фільтрації.

8. *in = in + 1* та перехід до пункту 5.

9. Якщо *m = 1* і *N > 1*, то проводиться формування параметрів шару Субдискретизація, інакше перехід до пункту 11: U вибирається рівним значенню 2. S вибирається рівним U. Subf вибирається як функція max - вибір максимального значення.

10. Розрахунок N для сформованого шару субдискретизації мережі за формулою (1.17), отриманої з формул (1.7) і (1.8), де ширина і висота зображення рівні.

$$N = \frac{N_p - U}{S} + 1, \qquad (1.17)$$

де *Np* - розмір квадратного попереднього шару мережі.

11. *d = d + 1.*

12. Ухвалення рішення про формування наступного рівня мережі: якщо N >= 3, то проводиться формування нового рівня мережі - перехід до пункту 4, інакше проводиться перехід до пункту 13.

13. Формування параметрів повнозв'язного шару мережі: K вибирається рівним C. Як функції активації нейронів повнозв'язного шару використовується сигмоїдна функція згідно з формулою (1.18).

$$AF_{fc} = \frac{1}{1 + e^{-x}}, \qquad (1.18)$$

де *x* - сигнал нейрона повнозв'язного шару, отриманий в результаті зваженого підсумовування вхідних сигналів нейрона.

14. Формування вихідного шару мережі, призначеного для розрахунку ймовірностей приналежності вхідного зображення представленим класів, кількість яких одно C.

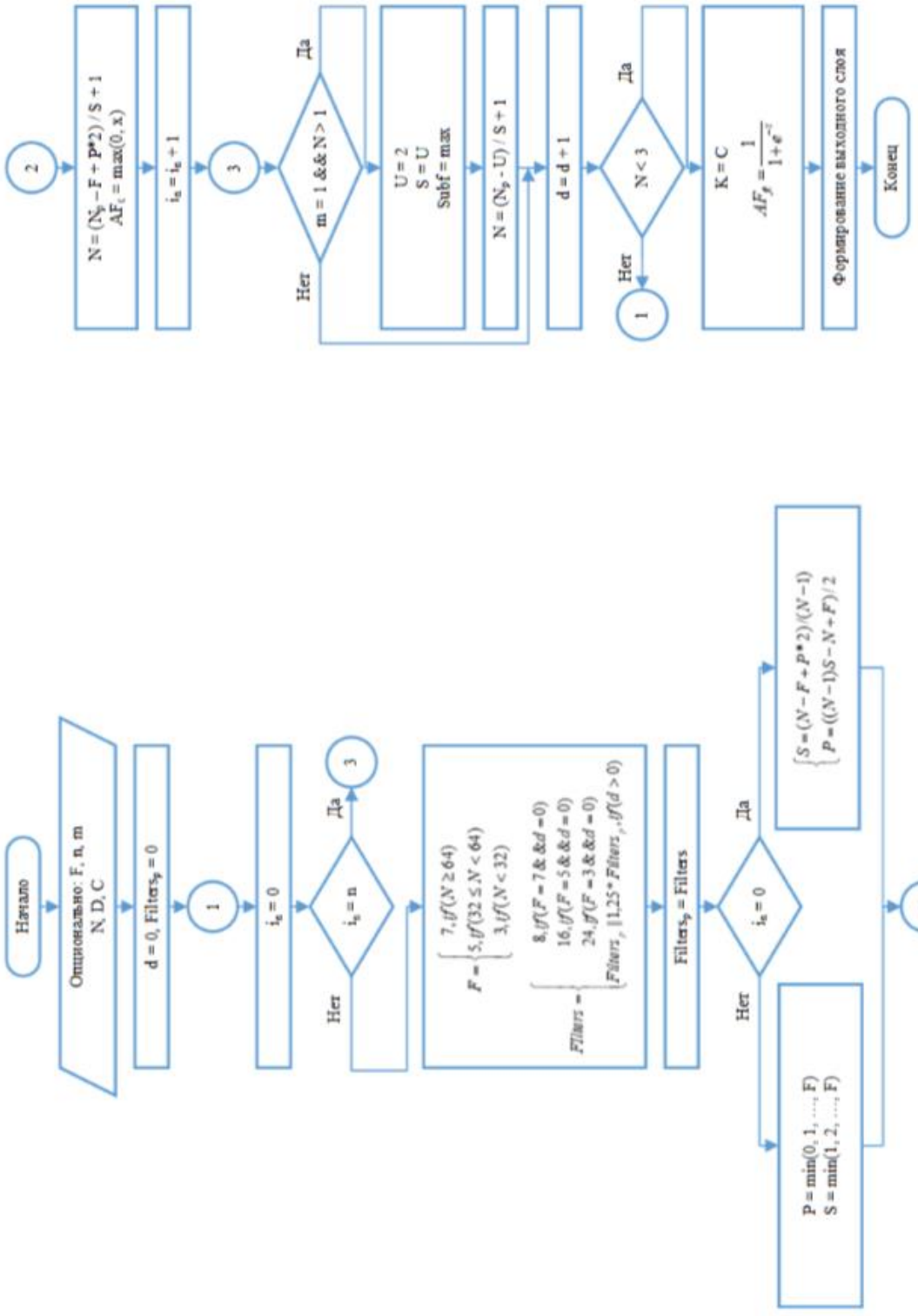


Рис. 1.5. Блок-схема алгоритму формування архітектури ЗНМ

РОЗДІЛ 2 НАВЧАННЯ НЕЙРОННИХ МЕРЕЖ

Процес навчання ШНМ розглядається як налаштування архітектури і ваг зв'язків між нейронами (параметрів) для ефективного виконання поставлених перед ШНМ завдань. Існує два великих класу навчання ШНМ: клас детермінованих методів і клас стохастичних методів.

В клас детермінованих методів входять методи, в основі яких лежить ітеративна корекція параметрів мережі, в ході поточної ітерації яка грунтується на поточні параметри. Основним детермінованих методом і найпоширенішим методом навчання ШНМ сьогодні взагалі є метод зворотного поширення помилки.

В клас стохастичних методів входять методи, що змінюють параметри мережі випадковим чином і зберігають тільки ті зміни параметрів, які привели до поліпшення результатів. Стохастичні алгоритми навчання реалізуються за допомогою порівняння помилок і деякі з них пов'язані з проблемою «пастки локального мінімуму», розв'язуваної за допомогою деяких ускладнень стохастичних алгоритмів.

2.1. Навчання з вчителем

Під час навчання ШНМ з учителем кожному наприклад з навчальної вибірки відповідає вектор, що характеризує однозначний правильну відповідь, що подається відразу на вихід мережі в обхід всієї її архітектури. Після отримання власного результату мережі, алгоритм порівнює результуючий вектор з правильною відповіддю, на основі чого відбувається корекція подальша помилки. Правило корекції помилково, на якому заснований метод зворотного поширення помилки, є класичним прикладом навчання з учителем.

2.2. Навчання без вчителя

Навчання без вчителя в разі ШНМ реалізується природним чином в процесі навчання, коли автоматична настройка параметрів мережею призводить до появи однакових результатів її функціонування при досить близьких вхідних значеннях, що на практиці можна порівняти зі зниженням розмірності даних в результаті ітераційного методу головних компонент. Правило Хеба, засноване на гіпотезі про посилення зв'язків між біологічними нейронами в разі їх одночасного збудження, і методи навчання при змаганні нейронів шляхом порівняння інтенсивності їх реакції є класичними прикладами методів навчання без учителя.

2.3. Метод зворотного поширення помилки (Backpropagation)

Метод є класичним методом навчання з учителем, заснований на правилі корекції помилково і був розроблений як метод навчання багатошарового перцептрона. Основна ідея методу полягає в поширенні сигналів помилки після її обчислення на виході мережі в напрямку, зворотному прямому поширенні сигналів під час звичайного обчислювального процесу, від виходів мережі до її входів. При зворотному проході синаптичні ваги налаштовуються з метою мінімізації помилки.

$$E(\{\omega_{i,j}\}) = \frac{1}{2}\sum_{k \in OUT} (t_k - o_k)^2 \qquad (2.1)$$

де □ - функція помилки;

$\omega_{i,i}$- синаптична вага між нейронами □, □;

□*UT* - безліч виходів мережі;

□$_k$ - правильні відповіді мережі, □ ∈ OUT;

□$_k$ - *вихід* □*-го нейрона.*

Фактично, реалізується стохастический градієнтний спуск, відбувається рух в багатовимірному просторі ваг до мінімуму помилки в сторону, протилежну градієнту. На підставі кожної групи правильних відповідей, настройка синаптичних ваг здійснюється шляхом обчислення:

$$\Delta\omega_{i,j} = -\eta\frac{dE}{d\omega_{i,j}}, \qquad (2.2)$$

де $0 < \eta < 1$ - множник швидкості руху.

В процесі навчання циклічно вирішуються однокритеріальне завдання оптимізації. Для можливості реалізації методу передавальна функція нейронів повинна бути диференційована. Оскільки метод є модифікацією класичного методу градієнтного спуску, то може розглядатися як градієнтний спуск по поверхні помилки.

Метод зворотного поширення помилки можна розділити на 4 окремі блоки: пряме поширення, функцію втрати, зворотне поширення і оновлення ваги. Під час прямого поширення, береться тренувальне зображення, це матриця N x N x 3 - і пропускається через всю мережу. У першому навчальному прикладі, так як всі ваги або значення фільтра були ініційовані випадковим чином, вихідним значенням буде щось на зразок [.0 .0 .0 .0 .0 .0 .0 .0 .0 .0], тобто таке значення, яке не дасть переваги якомусь певному числу. Мережа з такими вагами не може знайти властивості базового рівня і не може обґрунтовано визначити клас зображення. Це веде до функції втрати. Припустимо, перше навчальне зображення - це цифра 3. Ярликом зображення буде [0 0 0 1 0 0 0 0 0 0]. Функція втрати може бути виражена по-різному, але часто використовується СКО (середньоквадратична помилка), це 1/2 помножити на (реальність - передбачення) в квадраті:

$$E = \sum 1/2(target\text{-}output)^2 \qquad (2.3)$$

Ми хочемо досягти того, щоб прогнозований ярлик (висновок згорткового шару) був таким же, як ярлик навчального зображення (це означає, що мережа зробила правильне припущення). Щоб такого досягти, нам потрібно звести до мінімуму кількість втрат, яке у нас є. Візуалізуючи це як завдання оптимізації з математичного аналізу, нам потрібно з'ясувати, які входи (ваги, в нашому випадку) найбезпосереднішим чином сприяли втрат (або помилок) мережі. Завдання мінімізації втрат - відрегулювати ваги так, щоб знизити втрату. Візуально нам потрібно наблизитися до найнижчій точці чаше-подібного об'єкта.

Незважаючи на широке успішне застосування методу, він не позбавлений ряду недоліків, таких як вкрай низька швидкість навчання і принципова невдача навчання мережі (наприклад, нескінченна навчання), можлива по ряду причин, серед яких параліч мережі, проблема локальних мінімумів і проблема розміру кроку.

Ряд цих недоліків усувається різними модифікаціями методу, наприклад, модифікація RProp (Resilient Propagation) значно прискорює процес навчання за рахунок обчислення тільки знаків приватних похідних для підстроювання вагових коефіцієнтів, замість обчислення повних похідних. RProp використовує так званий метод навчання по епохах. Інші методи, що прискорюють процес навчання: QuickProp, метод сполучених градієнтів, метод Левенберга — Марквардта.

РОЗДІЛ 3 ГЕНЕТИЧНІ АЛГОРИТМИ

Генетичні алгоритми, це велике різноманіття евристичних алгоритмів що найчастіше використовують для підбору параметрів, коефіцієнтів, оптимальних рішень для різного рівня задач моделювання при цьому користуючись методами та механізмами схожими на процес природного відбору в природі. Входить до класу еволюційних методів, якими обчислюють оптимізацію з використанням методів схожих на процес еволюції: селекція, мутації і схрещування. Важливою властивістю, що присутня у генетичних алгоритмах є обов'язковість шагу схрещування, який слугує оператором рекомбінації кандидатів, що дуже схоже на роль схрещування в звичайній природі.

3.1. Узагальнений опис алгоритму

Задача формується так, щоб її рішення було представлено у вигляді вектора. Цей вектор називається генотипом, де ген (склада генотипу) може бути певним бітом або якоюсь іншою величиною в залежності від задачі, що вирішується. У канонічних варіантах реалізацій генетичних алгоритмів генотип завжди має одну й ту саму довжину, тобто вона є фіксованою. Але варто відмітити, що в сучасних версіях часто немає ніякого обмеження по довжині генотипу.

Для першої популяції, як правило, створюється велика кількість випадкових генотипів. Їх оцінка відбувається за допомогою так званої функції пристосування (або англійською *fitness*), яка визначає наскільки ефективним є генотип.

Важливим критерієм вибору функції пристосування є її рельєф - необхідно, щоб він був досить “гладким”.

Після запуску моделі з кінцевої кількості рішень вибираються “переможці” за значенням функції пристосування у кожного генотипа. До цих рішень застосовуються оператори схрещування і мутації і як результат отримуються нові набори рішень. Для них в свою чергу теж вираховується значення функції пристосування, і проводиться відбір рішень в наступне покоління, що показали найкращі результати.

Цей алгоритм є ітеративним і таким чином відтворюється еволюційний процес, що може тривати певну кількість поколінь в залежності від заданих критеріїв його роботи. Одними із видів критеріїв є:

- глобальний оптимум знайдено
- досягнуто граничну кількість поколінь
- завершення по тайм-ауту

Генетичні алгоритми використовують в більшій мірі для знаходження апроксимованих рішень в багатовимірних системах і просторах.

Тому зазвичай виділяються слідуючі кроки для загального генетичного алгоритму:

1) Визначити *fitness* функцію для всіх генотипів
2) Згенерувати першу популяцію генотипів
3) Провести операцію схрещування
4) Випадкові мутації
5) Обчислення *fitness* функції
6) Селекція та формування наступного набору генотипів
7) І т.д.

3.2. Опис основних операцій генетичних алгоритмів

Однією з основних операцій, що виконується під час роботи генетичного алгоритму є операція операція селекції. Під селекцією мається на увазі те, що після певної кількості моделювань вибираються найбільш ефективні генотипи серед усіх, що є в системі. Ця операція допомагає прискорювати моделювання і швидше знаходити вірне або близьке до вірного рішення за рахунок відкидання найменш ефективних генотипів.

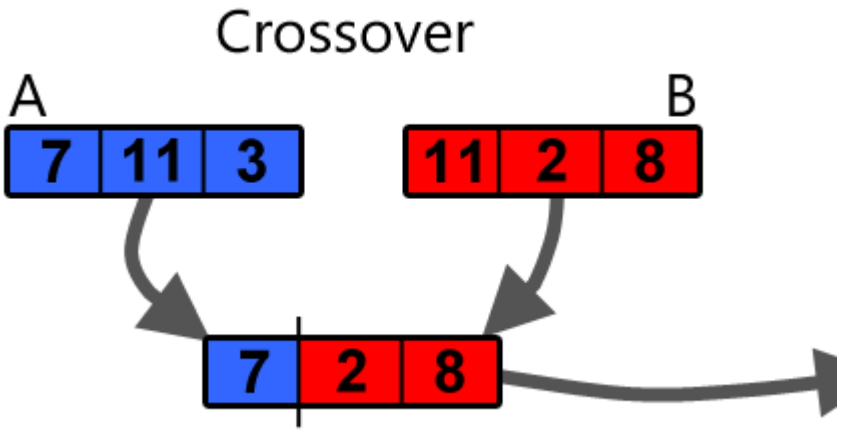


Рис. 3.1. Приклад механізму схрещування на випадковому генотипі

Іншою важливою операцією являється операція схрещування, яка часто виконується після селективного відбору найефективніших генотипів - це дає змогу отримати новий генотип, що заснований на декількох найефективніших. Операція схрещування може призводити до формування як і одного, так і декількох нових генотипів.

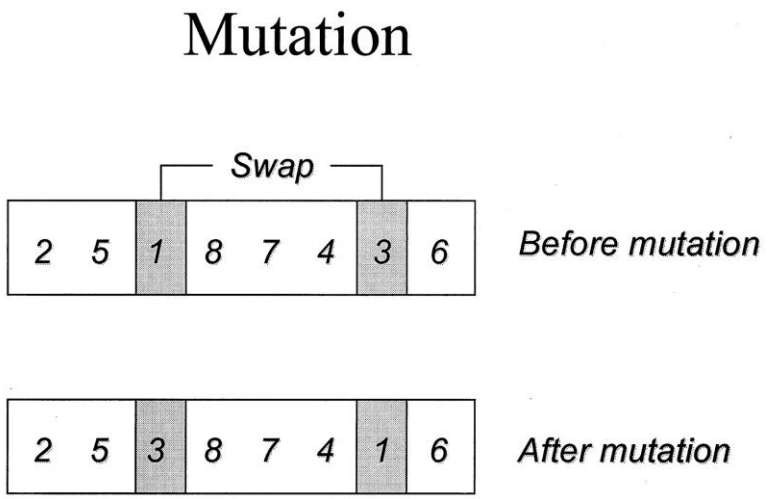


Рис. 3.2. Приклад механізму мутації на випадковому генотипі

Третьою операцією є операція мутацій, що випадково приміняється до випадкових генів у генотипі. Цей евристичний елемент дає змогу генерувати нові випадкові генотипи, які потенційно можуть призвести до отримання більш ефективних варіацій генотипів. Мутації можуть являти собою будь-які випадкові зміни у генотипі, наприклад, зміна генів місцями, або множення певних генів на випадкові коефіцієнти.

Майже кожна реалізація генетичних алгоритмів включає в себе усі три операції, що максимізує його результат.

3.3. Тренування нейронних мереж за допомогою генетичних алгоритмів

Генетичні алгоритми також досить добре вирішують задачі для тренування нейронних мереж, знаходження оптимальних топологій та оптимізації існуючих вагових коефіцієнтів.

Такі алгоритми можуть дозволяти створювати повністю працюючі рішення для нейромереж маючи у розпорядженні тільки сам набір даних для навчання, і, що найголовніше, не потребує від користувача глубоких пізнань у сфері штучного інтелекту.

Хоча необхідно завжди пам’ятати, що при навчанні мережі з вчителем на статичному наборі інформації за допомогою більш класичних алгоритмів таких як алгоритм зворотного поширення помилки часто можна досягти навіть кращих результатів ніж за допомогою нейроеволюційного варіанту.

Нейроеволюційні алгоритми краще підходять для випадків, коли методи, що включають в себе градієнтний спуск не можуть вирішити задачу якісно. Наприклад, задачі навчання мережі для орієнтації в середовищі, що динамічно змінюється або знаходження стратегій поведінки “агентів” у середовищі. Важливим фактором є те, що при такому моделюванні значення необхідних виходів нейронної мережі вже відомі. Таким чином якість роботи мережі вимірюють за допомогою цільової функції.

Однією з переваг даних алгоритмів є те, що вони слабо залежать від конкретного виду задачі і їх можна приміняти для вирішення цілого спектру різних проблем.

Генотип нейронної мережі можна закодувати наступними способами:

- За допомогою вагових коеф. зв’язків
- За допомогою інформації про конфігурацію нейронів
- За допомогою кодування шарів

Існує відома проблема, що виникає, наприклад, при вирішенні задачі пошуку оптимальної топології мережі, що називається проблемою конкуруючих результатів (або рішень). Проблема полягає в тому, що мережу можна представити у генотипі зовсім різними способами і це не дає інколи змогу провести операцію схрещування належним чином.

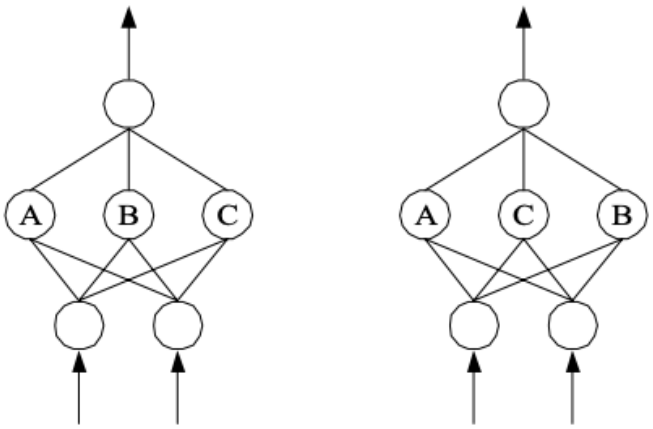


Рис. 3.3. Приклад проблеми конкуруючих результатів

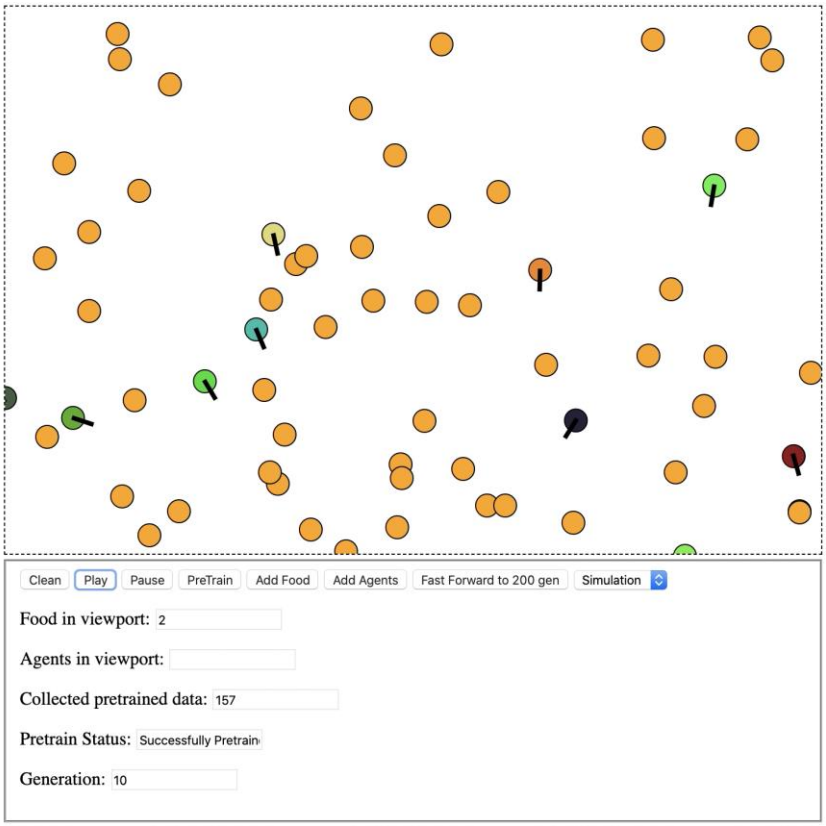
Якщо генотипи відповідають мережам, що зображені на рис.2.3, і вагові коефіцієнти у нейронів у схованому шарі відповідають один одному, то при схрещуванні результуючий генотип мережі буде включати в себе наступні конфігурації ABB та ACC.

Дану проблему можна вирішити декількома способами: наприклад, за допомогою більш “розумного” алгоритму схрещування, що буде ідентифікувати однакові зв’язки у нейронах схованого шару і потім формувати новий генотип враховуючи ці дані. Іншим варіантом, буду нумерування кожної ітерації і вираховування унікального індексу для усіх зв’язків, що дасть змогу знаходити аналогічні топології.

Також очевидно для різних варіантів архітектур нейронних мереж необхідні свої оператори схрещування та мутацій, щоб забезпечити переміщення інформації від “батьків” до їх нащадків. В незалежності від кількості отриманих нащадків, то отримані генотипи повинні включати в себе генетичну інформацію про кожного з батьків, тобто не повинен відбуватися процес втрати інформації отриманого в результаті еволюції моделі.

РОЗДІЛ 4 ПРЕЗЕНТАЦІЯ ТА ОПИС СТВОРЕНОГО ДОДАТКУ

Результатом роботи над даним дипломним проектом є створення додатку, основна мета якого полягає у моделюванні середовища з агентами (або



“мікроорганізмами”) та їжею, що випадковим чином розподілена по всьому полю.

Рис. 4.1. Інтерфейс створеного додатку

Сам проект являє собою web-додаток створений за допомогою HTML та TypeScript. TypeScript - це сучасна мова програмування створена компанією Microsoft, що являє собою розширену версію мови JavaScript зі статичною типізацією.

4.1. Опис інтерфейсу створеної програми

Інтерфейс програми складається з декількох основних блоків:

1) “Ігрове поле” - розташоване у верхній частині інтерфейсу користувача. Відображає поле наповнене агентами, що контролюються нейронними мережами, а також їжу за якою навчатимуться полювати агенти.

2) Кнопки для контролю моделювання і перемикання між різними режимами роботи.

3) Інформація для відладки та статистика розташована у нижній частині інтерфейсу.

Програма має 3 режими роботи: “*Претренування*”, “*Симуляція*”, “Пришвидшена симуляція”.

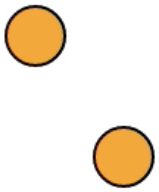
Режим “*Претренування*” необхідний, щоб задати вагові коефіцієнти для нейронної мережі, що буде використана для генерування першого покоління агентів. Програма дає можливість на прикладі того, як керує агентом користувач навчити її базовим реакціям на оточуюче середовище. Керування агентом відбувається за допомогою класичних для комп’ютерних ігор клавіш W, A, S, D - які дозволяють керувати прискоренням і кутом нахилу агента.

Також є можливість завантажити вже натренований *serialized* в форматі JSON екземпляр нейронної мережі і сформувати на її основі перших агентів.

У режимі “*Симуляція*” першим чином формується стартове покоління агентів і запускається процес моделювання, який можна спостерігати на ігровому полі у реальному часі.

Процес еволюції може займати певний час, тому існує ще третій режим роботи “*Пришвидшена симуляція*”, який дає змогу виконати багато симуляцій без відображення на ігровому полі, що в свою чергу дозволяє швидше ідентифікувати чи стають ефективнішими агенти з покоління у покоління.

“*Ігрове поле*” реалізовано за допомогою svg елементів, які керуються NPM пакетом Two.JS, який є обгорткою високого рівня для маніпуляції векторними



елементами та їх анімаціями.

Рис. 4.2. Вигляд їжі у середовищі

Їжа, що розподілена випадковим чином по середовищу виглядає як коло помаранчевого кольору.

Рис. 4.3. Вигляд агента, що моделюється

Агенти ігровому полі виглядають як зображено на рисунку 4.3 - вони складаються з основного “тіла”, а також вектору напрямку зору який позначений чорною рискою. Колір агента відповідає відповідає його геному, це зроблено для ідентифікації мутацій - тобто якщо користувач програми бачить, що агенти різного кольору, то це буде означати, що конфігурація нейронних мереж у цих агентів різна (і навпаки).

4.2. Архітектура нейронної мережі та опис генетичного алгоритму

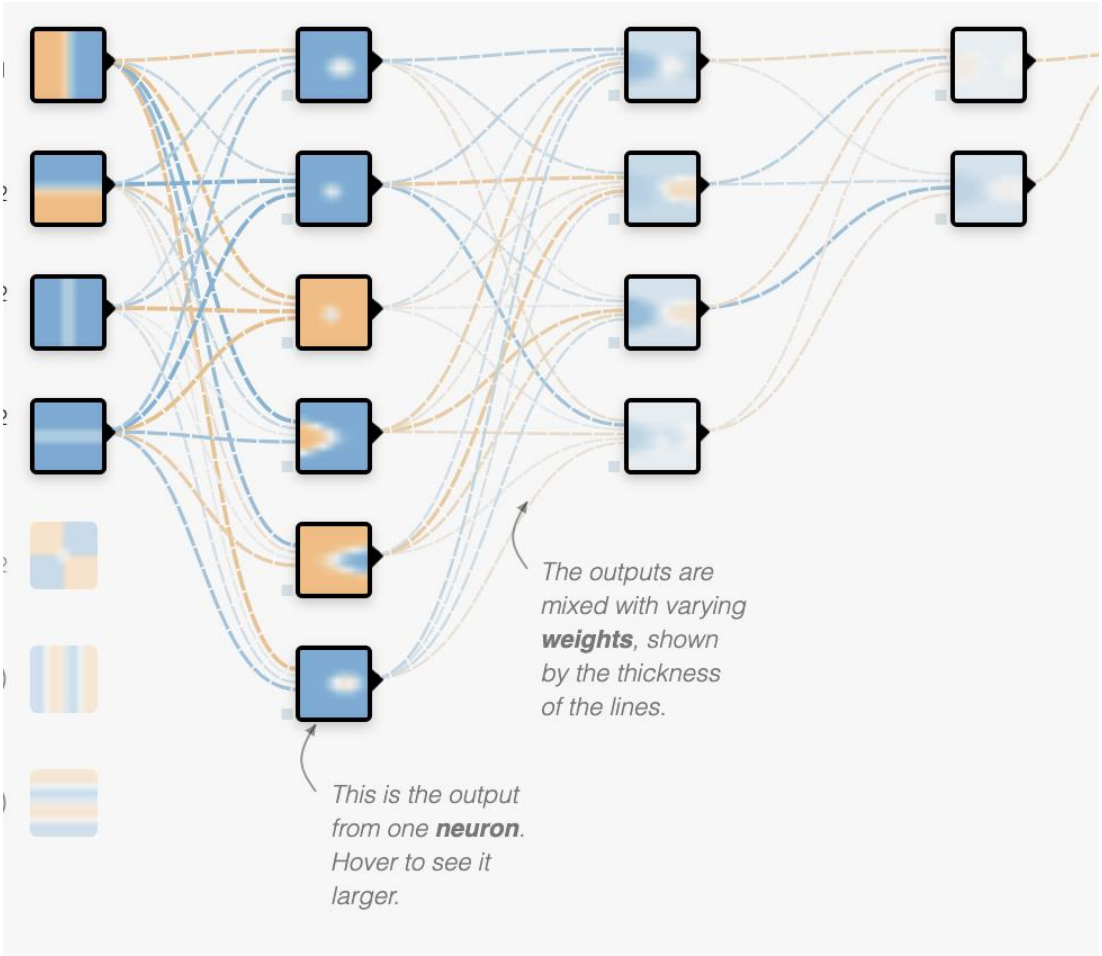


Рис. 4.4. Схема нейронів і зв’язків

Архітектура нейронної мережі (рис. 4.4), що моделювалася - це звичайний багатошаровий перецептрон з:

- 4 вхідними нейронами
- 6 нейронами в першому прихованому шарі
- 4 нейронами в другому прихованому шарі
- 2 вихідними нейронами

Для реалізації нейронних мереж була використана бібліотека під назвою Brain.JS. Вона реалізує цілий ряд алгоритмів навчання та дозволяє

експериментувати з різними типами нейронних мереж, а також серіалізовувати і десеріалізовувати натреновані моделі.

```
// provide optional config object (or undefined). Defaults shown.
const config = {
  binaryThresh: 0.5,
  hiddenLayers: [3], // array of ints for the sizes of the hidden layers in the network
  activation: 'sigmoid', // supported activation types: ['sigmoid', 'relu', 'leaky-relu', 'tanh'],
  leakyReluAlpha: 0.01, // supported for activation type 'leaky-relu'
}

// create a simple feed forward neural network with backpropagation
const net = new brain.NeuralNetwork(config)

net.train([
  { input: [0, 0], output: [0] },
  { input: [0, 1], output: [1] },
  { input: [1, 0], output: [1] },
  { input: [1, 1], output: [0] },
])

const output = net.run([1, 0]) // [0.987]
```

рис.4.5. Приклад опису найпростішої мережі

У якості активаційної функції була використана лінійна активаційна функція ReLU

(4.1) $$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geqslant 0 \end{cases}$$

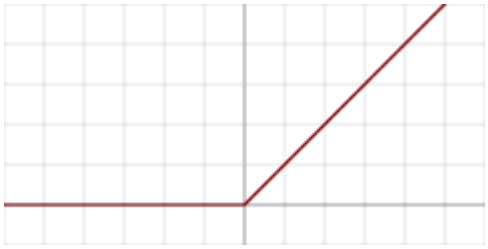


рис.4.5. Приклад опису найпростішої мережі

Дана функція має графік як зображено на рис.4.6., вона є лінійною. Також можна використовувати і інші типи нейронних мереж, для цього необхідно змінити конфігурацію у файлі src/game - доступними варіантами функцій є:

- Sigmoid, класична сигмоїдальна активаційна функція
- ReLU, функція за замовчуванням
- Leaky-ReLU
- Tanh

Нейронна мережа керує поведінкою агента в середовищі сприймаючи 4 вхідних сигнали і видаючи 2 сигнали на вихід. На вхід подаються наступні змінні: поточна швидкість, поточний кут повороту відносно горизонтальної осі, та відсотки кількості їжі і інших агентів, що має у полі зору сам агент.

Сигнали, що отримуються на вихід, це $\boldsymbol{\Delta v}$ та $\boldsymbol{\Delta \gamma}$ - сигнал зміни швидкості та кута повороту агента.

Важливим моментом, при тренуванні, або навіть подачі будь-яких вході до нейронної мережі є нормалізація вхідного вектору даних. Це необхідно для того, щоб дати можливість нейронній мережі оперувати в контексті обмежень активаційних функцій нейронів. В даній моделі, усі входи та виходи є числами, що належать до проміжку [0; 1] і для того щоб конвертувати від'ємні та інші значення за межами діапазону було використано метод *min-max normalization* формула *(4.2)*.

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}} \qquad (4.2)$$

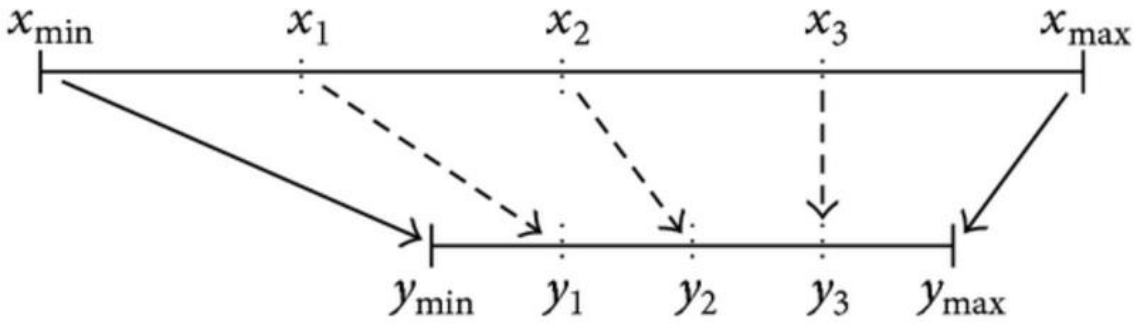


рис.4.6. Ілюстрація алгоритму нормалізації

Нейронна мережа тренується в режимі “Претренування” досить поверхнево, але таким чином, щоб досягти будь-якої працюючої моделі мережі для керування агентами. Закладено поріг 30000 ітерацій для зупинки тренування, якщо метод тренування не дає належних результатів. Після початкового тренування нейронна мережа використовується для створення першої популяції агентів, де вона випадково може включати в себе мутації.

4.3. Архітектура агентів

Агент у розробленій програмні це модель штучного мікроорганізма, основною еволюційною задачею якого є навчитися розпізнавати їжу та “з’їдати” їжу, що розподілена випадковим чином по усій поверхні середовища.

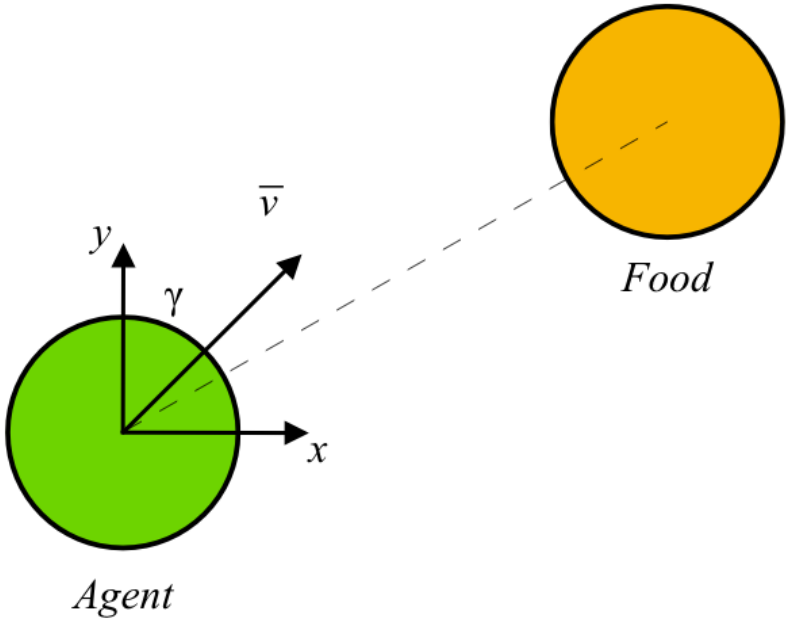


Рис.4.6. Ілюстрація моделі агента

Агент має наступні характеристики:

1) *V* - швидкість, що може бути від 0 до 5
2) *(X,Y)* - вектор позиції на полі
3) *S* - вектор зору (співпадає з вектором швидкості)
4) *network* - нейронна мережа, що виступає генотипом агента
5) eaten - кількість з'їденої їжі

Під час еволюції нейронна мережа отримуватиме в режимі реального часу сигнали від основних параметрів агента - швидкості, куту нахилу відносно горизонтальної осі, і інформації про те, що знаходиться попереду агента.

Коли агент стикається з їжею і "торкається" її, то елемент їжі зникає з ігрового поля, а параметр eaten в екземплярі Agent інкрементується. Таким чином зберігається інформація про те, скільки їжі, кожен агент зміг знайти за час моделювання одного покоління. Детекція колізій агентів і їжі зводиться до

простої задачі рахування відстані між центром кола агента і центром елемента їжі за допомогою наступної формули:

$$distance(O_{food}, O_{agent}) = \sqrt{(x_f - x_a)^2 + (y_f - y_a)^2} \quad (4.3)$$

Після цього разрахунку програма перевіряє чи менше відстань між об'єктами ніж сума радіусів кіл агента і їжі: якщо менше або дорівнює, то значить агент і їжа перетинаються.

Параметр eaten є fitness показником для кожного агента. Після того як вся їжа на ігровому полі закінчується, то програма зупиняє покоління і обирає 2-х найбільш успішних агентів, тобто агентів, що знайшли найбільше їжі. І на основі них генерує нове покоління за допомогою операції схрещування.

4.4. Еволюція агентів

Операція схрещування відбувається над двома агентами через випадкове схрещування вагових коеф. обох нейронних мереж. При чому це схрещування є медіанним і нові вагові коефіцієнти результуючої мережі будуть вираховуватися за формулою:

$$median(a, b) = \frac{max(a,b) - min(a,b)}{2} + min(a, b) \quad (4.4)$$

Результуюча мережа є усередненою версією двох найкращих агентів з покоління. Так як це не гарантує, що така мережа буде ефективною, то алгоритм додає у наступне покоління батьків, щоб переконатися у тому, що показники ефективності отриманої мережі не деградують.

У реалізованій програмі алгоритми мутацій і схрещування не змінюють топологію мережі, а модифікують тільки вагові коефіцієнти і значення баєсів.

Мутації у свою чергу побудовані на алгоритмі з використанням випадковості для прийняття рішення щодо того, чи необхідно робити мутацію конкретного значення в генотипі агента.

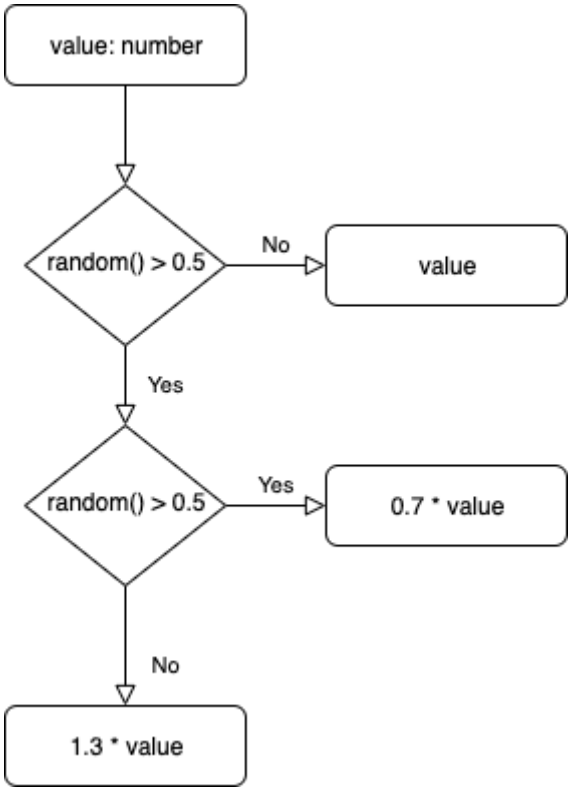


Рис.4.7. Блок-схема алгоритму мутації

Мутації відбуваються через повний перебір вагових коефіцієнтів мережі та випадкового застосування функції мутації на них. Таким чином кожен агент нового покоління генерується на базі отриманої після схрещування мережі і її випадкових мутацій, унікальних для кожного агента.

Для покращення візуальної ідентифікації унікальних генотипів у модельованому середовищі для кожного генотипу вираховується хеш за допомогою односторонньої хеш-функції на основі якого задається колір агента. Отже якщо агенти мають однаковий колір, то вони мають однаковий геном, і, навпаки, якщо колір хоч трохи відрізняється - геноми можуть бути абсолютно різні. Це зв’язано з особливістю хеш-функцій: якщо змінити хоч один біт вхідної

інформації, то її хеш буде повністю відрізнятися від хешу початкового набору вхідних даних.

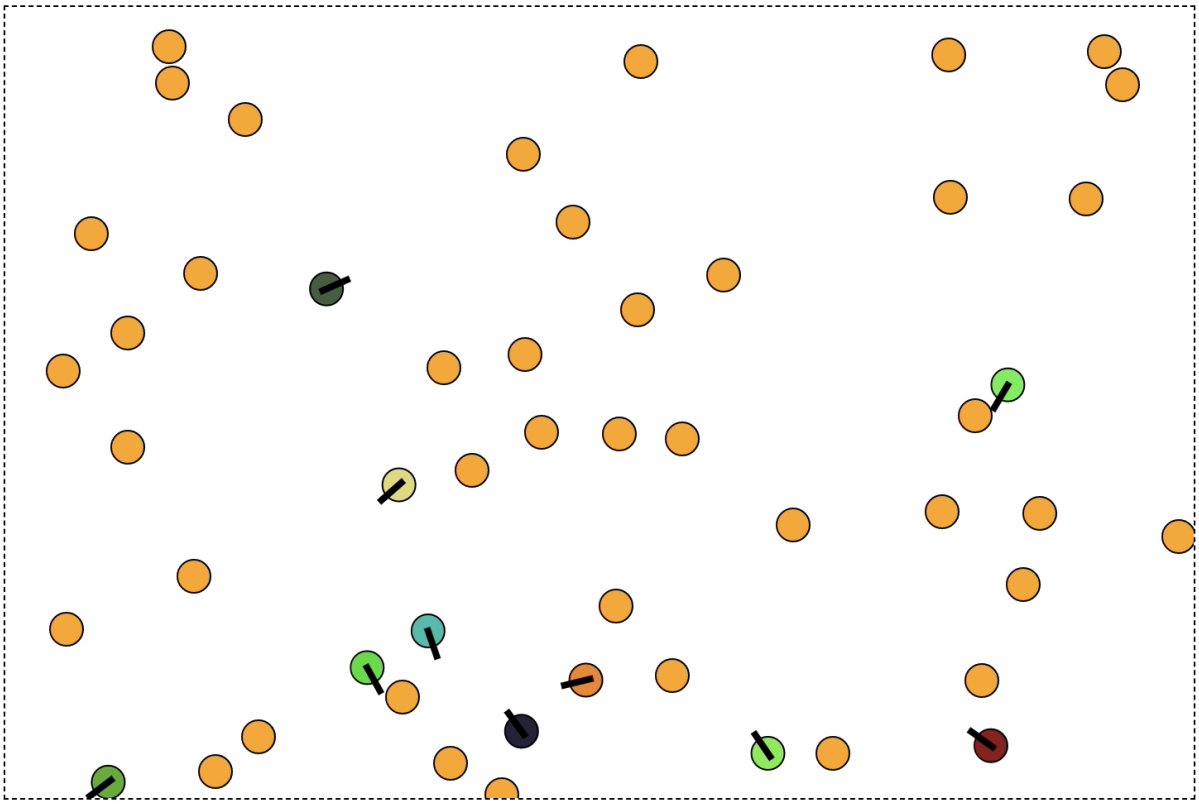


рис.4.8. Агенти з різними генотипами

На рисунку 4.8 видно приклад популяції агентів з різними геномами - вони мають кардинально різні кольори (крім двох зелених).

4.5. Структура розробленої програми

Програма була розроблена за допомогою сучасної мови програмування TypeScript, що має підтримку парадигм об’єктно-орієнтовного, функціонального і імперативного програмування. Особливістю мови є те, що ця мова дуже схожа на останні версії мови JavaScript, але при цьому має статичну типізацію, що дозволяє валідувати написаний код ще під час його написання. Також TypeScript дає

можливість налаштовувати строгість компілятора, що гарно для розробки прототипів.

Структура файлів проекту виглядає наступним чином:

```
├── index.html
├── package-lock.json
├── package.json
├── src
│   ├── agent.ts
│   ├── food.ts
│   ├── game.ts
│   ├── index.ts
│   ├── movement.ts
│   ├── network.ts
│   ├── types.ts
│   └── utils.ts
└── tsconfig.json
```

Головним файлом з якого починається запуск програми є *index.ts*, це канонічна назва ініціючих файлів ще з мови JavaScript.

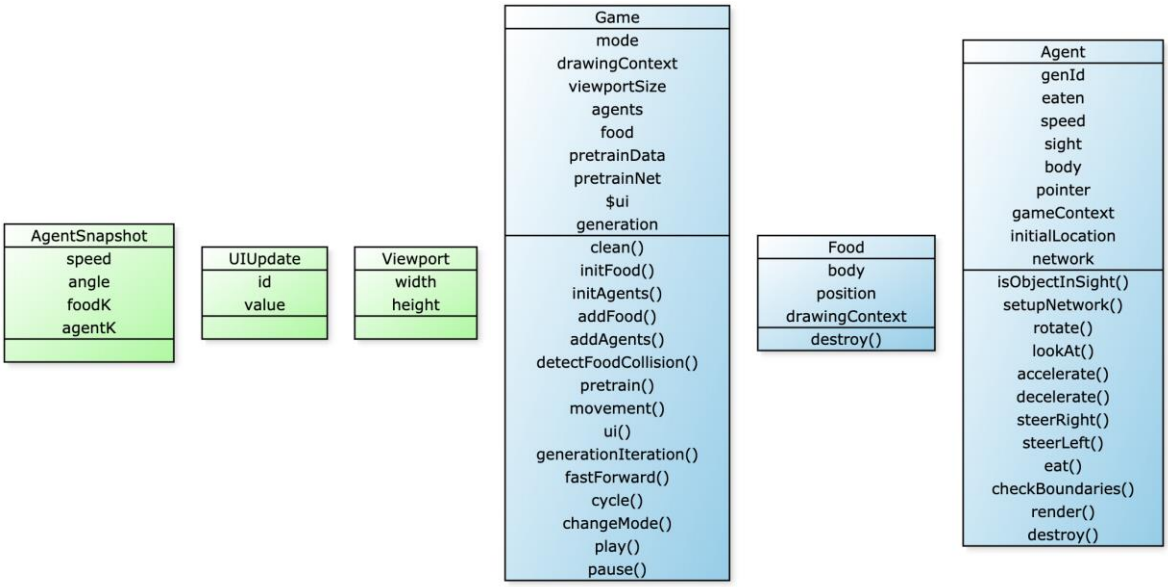


рис.4.9. UML діаграма основних сутностей

Основним классом програми є класс Game. В ньому описується логіка зв'язана з усім процессом моделювання, а також ініцюється контекст для рендеру процесу в svg за допомогою бібліотеки "Two.Js".

Важливі поля, що включає в себе даний класс:

- mode, GameMode режим моделювання
- agents, Set агентів
- food, Set їжі
- generation, Number номер поточного покоління
- $ui, RxJS.Subject для оновлення інтерфейсу

Класс Agent у свою чергу описує роботу агента і включає в себе такі параметри як позиція, швидкість, вектор зору і нейронна мережа. Також описує методи для рендеру і змінення основних параметрів, такі як *accelerate, decelerate, rotate, eat, setupNetwork*.

Класс Food достатньо простий, має тільки позицію і властивості для відмальовки SVG елемента.

Увесь UI або користувацький інтерфейс було вирішено зробити за допомогою стандартного набору елементів, що представляє браузер. Для оновлення інтерфейсу використовується RxJS Observable об'єкт типу Subject під назвую $ui.

В нього передаються відповідно назва UI-елементу і новий стан або значення, після чого listener з *index.js* реагує на зміни і оновлює відповідні елементи у Document Object Model сторінки. Це приклад так званої “реактивної” (від *FRP functional react programming)* реакції користувацького інтерфейсу на зміни стану програми.

Програма запускається і компілюється за допомогою спеціального інструменту під назвою *Parcel.* Цей інструмент потребує мінімум конфігурації для налаштування базового проекту. Для його встановлення в проект необхідно виконати наступну CLI команду:

$ npm i [-g] parcel

Для старту серверу в терміналі необхідно виконати наступну команду:

$ npm run dev

Після чого буде запущено спочатку збирання проекту, а потім буде піднятий сервер для віддачі актуальної версії програми на тестовому TCP порті під номером 1234. Приклад запущеного серверу для розробки знахдиться на рисунку 4.10.

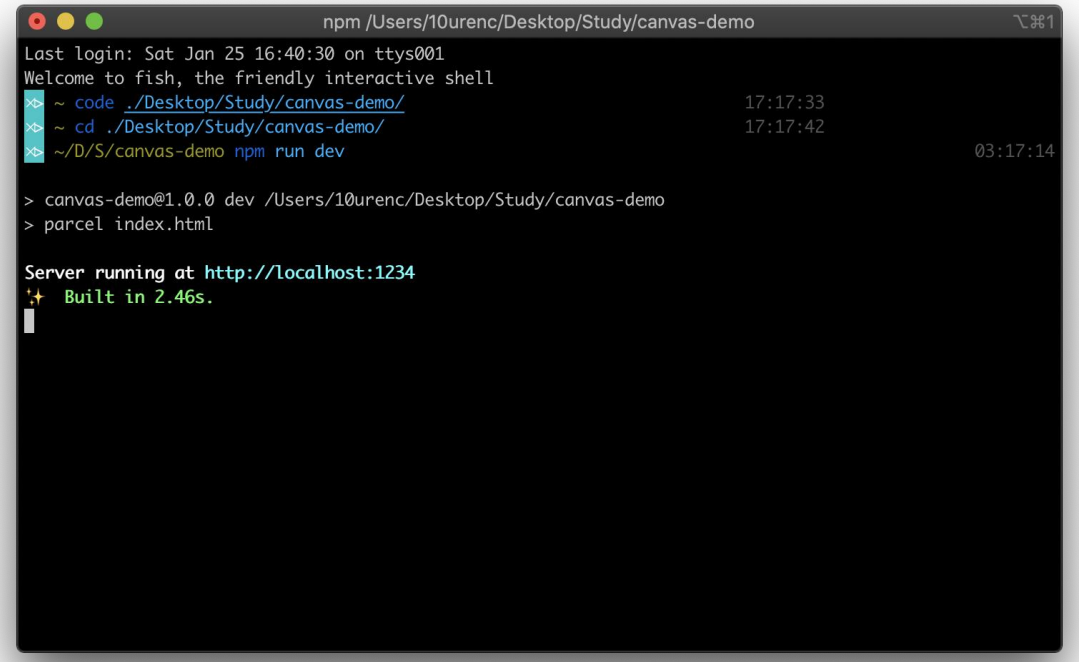


рис. 4.10. Вікно консолі з сервером

4.6. Результати спостережень за моделлю

У результаті моделювання середовища з мікроорганізмами-агентами та елементами їжі була проведена низка експериментів та як результат було знайдено декілька конфігурацій, що задовольняють умови дослідження: знайти оптимальну матрицю ваг нейронної мережі агента, щоб агент якнайефективніше знаходив та “поглинав” їжу в модельованому просторі.

Одна з найкращих варіацій поколінь, що була отримана за допомогою програми повністю знищила всю їжу в середовищі розміром 720 на 480 за 8 секунд, що є достатньо високим результатом відносно перших поколінь, у яких середній час проходження однієї ітерації був близько 2 хвилини 35 секунд - що є значно гіршим показником.

З кожним поколінням агентів видно як зберігається генетична інформація про минулі покоління. Хоча нейронна мережа на має ніякого стану у реальному часі і

немає реалізації пам’яті, то генитична пам’ять що можна спостерігати являє собою вагові коефіцієнти і Bias-значення нейронної мережі.

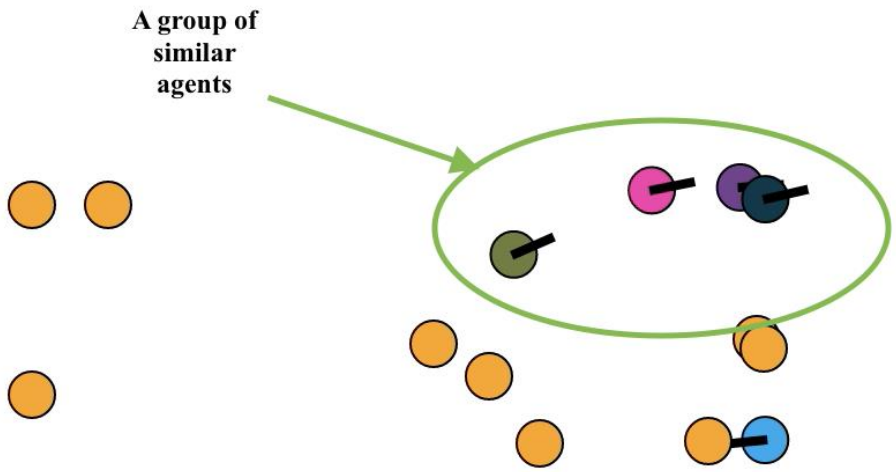


рис. 4.11. Агенти зі схожими генотипами, що об’єднуються в одну групу

Прикладом генетичної пам’яті, що відстежується протягом багатьох симуляцій підряд є об’єднання агентів в групи. Таке трапляється досить часто майже в кожній симуляції, часто в незалежності від коригування конфігурації процесу еволюції. Цей ефект зумовлений тим, що агентам на вхід нейронної мережі подається також інформація про агентів, що знаходяться в їх полі зору і вони мають змогу базувати свої рішення ще на цій інформації.

Іншим прикладом рис, що зберігаються і відстежуються в багатьох симуляціях є те, що багато варіацій агентів під часу руху в просторі роблять коливальні рухи вектором зору таким чином збільшуючи шанси знайти більшу частку їжі на полі. І навіть більше - були знайдені популяції в яких агенти робили повний оберт навколо себе, щоб побачити якомога більше їжі.

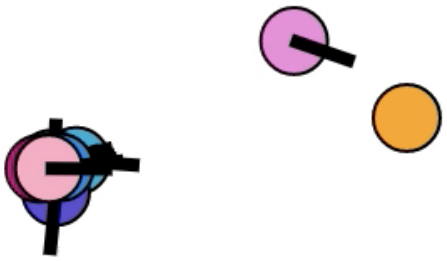


рис.4.12. Приклад успішної мутації

Прикладом вдачної мутації є агент рожевого кольору, що зображений на рисунку (4.11) - мутація призвела до того, що він став швидшим за інших агентів цього покоління. Схожим чином з’являться якісні мутації що стосуються загальної поведінки нових агентів.

Отже, за допомогою даного програмного продукту було розроблено модель мікроорагнізму у вигляді агента, що керується штучною нейронною мережею і який цілком успішно був запрограмований орієнтуватися і взаємодіяти з модельованим середовищем. Поточні розробки можна використовувати для програмування і оптимізації різного роду нейронних мереж для вирішення великого спектру задач. Пропонується використовувати даний алгоритм для тренування і розробки штучного інтелекту, що може використовуватися, наприклад, в комп’ютерних іграх, а також різних наукових моделюваннях таким чином зменшуючи необхідний ресурс для створення комплексних моделей.

ВИСНОВКИ

В даній роботі було досліджено існуючі методи створення штучних нейронних мереж, також була зроблена порівняльна характеристика градієнтних і евристичних алгоритмів навчання таких як нейроеволюційні алгоритми. У ході

аналізу зроблені висновки щодо класу задач для яких краще підходить кожен тип тренування.

Розроблено нейроеволюційний алгоритм для формування найбільш ефективної архітектури та конфігурації вагових коефіцієнтів нейронної мережі, що забезпечує повноцінне моделювання еволюційної системи з агентами-мікроорганізмами та вирішує задачу оптимізації поведінки агентів у середовищі навчаючи їх відповідним реакціями на оточуюче середовище.

На базі даного алгоритму також був створений веб-додаток який дозволяє безпосередньо у браузері користувача відтворити моделювання і зберегти отриману модель для подальшого використання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Козин, Н.Е. Поэтапное обучение радиальных нейронных сетей / Н.Е. Козин, В.А. Фурсов // Компьютерная оптика. – 2004. – № 26. – С. 138-141.
2. LeCun, Y. Gradient Based Learning Applied to Document Recognition / Y. LeCun, L. Bottou, P. Haffner – IEEE Press, 1998. – P.46.
3. Simard, P.Y. Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis / P.Y. Simard, D. Steinkraus, J. Platt // International Conference on Document Analysis and Recognition (ICDAR), IEEE Computer Society. – Los Alamitos. – 2003. – P. 958-962.
4. Хайкин, С. Нейронные сети: полный курс / С. Хайкин. – М.: Вильямс, 2006. – 1104 с
5. Общие понятия фильтрации изображений. Учебное пособие по курсу «Компьютерная обработка изображений» // Университет ИТМО. Конспект лекций. 2016.
6. Y. Lecun, L. Bottou, Y. Bengino, P. Haffner. Gradient-based learning applied to document recognition // Proceedings of the IEEE, November, 1998, Volume 86, Issue: 11, Pp. 2278-2324.
7. Спицын В.Г., Цой Ю.Р. Представление знаний в информационных системах: учебное пособие / Томский политехнический университет – Томск: Изд-во ТПУ, 2007. – 160 с.
8. Гаршин, А.А., Солдатова, О.П. Автоматизированная система распознавания рукописных цифр на основе свёрточной нейронной сети // Свидетельство об официальной регистрации программ для ЭВМ №2010610988 по заявке №2009616812 от 1 декабря 2009 года. Зарегистрировано в Реестре программ для ЭВМ 1 февраля 2010 года.
9. Bishop, C.M. Neural Networks for Pattern Recognition – Oxford University Press, 1995. – 498 p.

10. Gaussian blur – http://en.wikipedia.org/wiki/Gaussian_blur. 11. Duffner, S. An Online Backpropagation Algorithm with Validation Error-Based Adaptive Learning Rate / S. Duffner, C. Garcia // ICANN 2007, Part I, LNCS 4668, 2007. – P. 249-258. References

11. Форсайт Д.А., Понс Ж. Компьютерное зрение. Современный подход. - М.: Вильямс, 2004. – 928 с. 86 2. Уоссермен Ф. Нейрокомпьютерная техника: Теория и практика. - М.: Мир, 1992. – 184 с.

12. Флах П. Машинное обучение. Наука и искусство построения алгоритмов, которые извлекают знания из данных. – М.: ДМК Пресс, 2015. – 400 с.