Міністерство освіти і науки України
Львівський національний університет імені Івана Франка
Факультет прикладної математики та інформатики

Кафедра дискретного аналізу
та інтелектуальних систем

Курсова робота

на тему:

Застосування нейромереж в задачі контролю динамічної системи

Виконали
ст. групи ПМі-33
Накрийко А.В.
Тхір П.С.

Науковий керівник
ас. Годич О.В.

Львів - 2008

Зміст

1 Вступ

Штучні нейронні мережі — це технологія, яка сягає своїм корінням в кілька наукових дисциплін: нейрофізіологію, математику, статистику, фізику, комп'ютерні науки та техніку. Гнучкість нейромереж та їх широкі можливості роблять нейромережі цікавим об'єктом дослідження, який, щоправда, в силу своєї новизни є ще недостатньо добре вивченим, що ускладнює використання його на практиці, оскільки потребує нетривіальної адаптації для кожної конкретної задачі.

Нейромережі успішно застосовуються в різноманітних практичних задачах, таких як: моделювання, аналіз часових рядів, розпізнавання образів, обробка сигналів і управління, завдяки одній дуже важливій властивості — здатності навчатися на основі даних з участю вчителя чи без його втручання[2].

Предметом нашого дослідження є використання нейромереж у теорії керування. Нейромережа є універсальним засобом моделювання довільної функції. В нашому випадку цією функцією є функція керування, яка в якості параметрів приймає стан зовнішнього середовища, а на вихід видає потрібні параметри управління підконтрольним об'єктом. Застосування нейромережі в теорії керування є цікавим з точки зору "природності" поведінки системи. Завдяки нелінійній природі нейромережі можна спробувати домогтися адекватної реакції нейромережі в нових для неї умовах, які не були передані їй в навчальних даних. В силу цього поведінка контрольованої нейромережею системи буде значно ближчою до поведінки системи, контрольованої людиною, ніж при використанні інших засобів контролю.

Щоб перевірити ці припущення, ми поставили собі за мету створити автоматичну систему контролю ("автопілот") автомобіля, використовуючи для цього нейромережу. Основним завдання автопілота автомобіля є самостійний рух в віртуальному середовищі та оминання перешкод, які зустрічаються на шляху авто. Щоб якомога краще наблизити поведінку автопілота до людської поведінки, система керування автомобілем "бачить" лише об'єкти, які потрапляють у задане поле зору, обмежене певним кутом та дальністю огляду. На основі даних про видимі перешкоди система контролю (нейромережа) робить рішення про необхідну зміну прискорення та повороту керма автомобіля.

Для того, щоб домогтися потрібної поведінки, нейромережа проходила навчання на навчальній вибірці, в якій були закладені знання про

бажану реакцію системи в декількох основних ситуаціях. Для ефективного випробовуваня різних конфігурацій нейромереж та наборів навчальних даних, був створений редактор з функціями редагування нейромереж та їх навчання алгоритмом зворотного поширення помилки (back propagation learning).

Оскільки реалізація автопілота на базі фізичної моделі автомобіля доволі проблематична, було прийнято рішення створити віртуальне середовище, в якому можна було б досліджувати поведінку системи керування. Для реалізації віртуального середовища була використана технологія візуалізації 3D-графіки OpenGL. Також був створений редактор віртуального середовища, завдяки якому вдалося перевірити роботу автопілота у різних умовах та ситуаціях.

2 Постановка задачі

Нашею задачею є реалізація системи контролю автомобіля, яка забезпечує безперервний рух у віртуальному середовищі з огинанням перешкод. Це завдання є, фактично, застосуванням теорії керування, або, якщо бути більш точним, то використання апарату нейромереж в теорії керування.

Таким чином, наша робота — це створення функції керування автомобілем. Для її реалізації потрібно насамперед визначити функцію стану, на базі якої функція керування буде приймати рішення, та параметри динамічної системи (автомобіля), якими функція керування буде впливати на поведінку автомобіля для досягнення бажаного результату.

За стан системи ми вибрали інформацію про наявність та дальність видимих об'єктів. Автомобіль має власне поле зору, визначене певними значеннями кута огляду та дальності бачення. Все поле зору ділиться на три рівні частини. Після цього для кожної частини визначається числове значення з діапазону $[0, 1]$, яке відповідає відносній відстані об'єкта, що попадає в цю частину. Таким чином, якщо автомобіль дотикається до певного об'єкта, то значення параметра буде рівне 0, якщо ж об'єкта в заданій частині поля зору немає або об'єкт знаходиться на максимальній видимій відстані, то відповідний параметр буде рівний 1. Таким чином стан системи буде визначатеся трьома дійсними параметрами.

Параметри динамічної системи, які визначає функція керування — це прискорення автомобіля та кут повороту керма. Обидва параметри знаходяться в діапазоні $[-1, 1]$. В випадку прискорення від'ємне значення відповідає гальмуванню або руху автомобіля назад, в залежності від того, в яку сторону рухається автомобіль в даний момент. Для кута повороту значення -1 відповідає максимальному повороту наліво, 0 — руху прямо, -1 — максимальному повороту направо.

Функцію управління ми реалізували на базі багатошарової нейромережі прямого поширення, яка була заздалегідь навчена на вибірці з кількох десятків навчальних наборів даних. На вхід нейромережі подавався певний стан автомобіля і йому співставлялися бажані значення акселерації та повороту. Навчання нейромережі проводилося за алгоритмом *зворотного поширення помилки (back propagation learning)*.

Водночас, для перевірки роботи системи керування автомобілем потрібне віртуальне середовище, яке б давало змогу наочно спостерігати за

поведінкою керованого автомобіля. Це віртуальне середовище було реалізовано з використанням бібліотеки 3D-графіки OpenGL. В силу своєї нетривіальності, реалізація цього середовища становить собою вагому частину всієї роботи. Таким чином, курсова робота складаєтсья з двох взаємопов'язаних частин — реалізація віртуального середовища та системи контролю.

3 Поняття про теорію керування

Теорія керування є наукою про засади та методи управління різноманітними системами, процесами і об'єктами[4]. Основний принцип теорії керування: на базі аналізу *об'єкту управління* (ОУ) синтезується його *математична модель*; потім на основі інформації про бажані характеристики ходу процесу або про мету управління синтезується *алгоритм управління* (АУ). Дана область знань інтенсивно розвивається і знаходить широке застосування в сучасній техніці.

Проблеми керування реальними системами у вік швидкого науково-технічного розвитку набувають значної важливості та актуальності. В цьому напрямку викликає значний інтерес керування системами з позицій оптимізації певних характеристик системи, наприклад, максимізації дальності польоту літального апарату, максимізації прибутку підприємства, мінімізації енергії або витрат, необхідних для досягнення певного цільового стану, мінімізації відхилення траєкторії руху системи від заданої та ін. Можна навести ще багато подібних задач. Знаходження та дослідження керування, за допомогою якого може бути досягнута задана ціль шляхом оптимізації певного критерію, складає фундаментальну основу теорії керування[1].

Для постановки задач оптимального керування необхідно, в першу чергу, визначити цільову функцію оптимізаційного процесу. З цією метою потрібно зробити відповідне формулювання задачі у фізичній формі і переклад цього фізичного опису на формальну мову математичних співвідношень. Для здійснення ефективного керування процесом необхідно знати поточний стан системи, охарактеризувати процес за допомогою адекватної математичної моделі, залежної від дії різноманітних зовнішніх факторів. За умови відомої функції цілі, стану і параметрів системи можна ставити задачу знаходження найкращого керування, яке оптимізує цільову функцію.

Загальна схема системи керування з зворотнім зв'язком[4] подана на рис. 1.

Контролер, базуючись на власних "знаннях", здійснює управління *динамічною системою*, генеруючи на виході *керуючий сигнал*. Згенерований вихідний сигнал проходить через *датчик контролю вихідного сигналу* і на вхід контролера подається певна оцінка сигналу контролера разом з *бажаним вихідним сигналом*, що подається ззовні. Враховуючи оцінку фактичного вихідного сигналу та бажаний вихідний сигнал,

Рис. 1: Схема системи керування з зворотнім зв'язком

контролер корегує власні "знання" таким чином, щоб покращити власну реакцію на поточний стан динамічної системи. Багатократне повторення даної схеми забезпечує вдосконалення реакції контролера, а отже і досягнення мети *системи контролю*.

4 Штучні нейронні мережі

4.1 Що таке нейронні мережі

Дослідження по штучних нейронних мережах (далі — нейромережі) пов'язані з тим, що спосіб обробки інформації людським мозком радикально відрізняється від методів обробки інформації, які використовуються в звичайних цифрових комп'ютерах. Мозок являє собою надзвичайно *складний, нелінійний, паралельний* комп'ютер (систему обробки інформації). Він здатен організовувати свої структурні компоненти, названі *нейронами (neuron)*, так, щоб вони могли виконувати конкретні задачі (такі як розпізнавання образів, обробку сигналів органів чуттів, моторні функції) в багато разів швидше, ніж це можуть зробити сучасні комп'ютери (тут і далі теоретичні дані взято з [2]).

Прикладом такої задачі може бути *локатор (sonar)* кажана, що являє собою систему активної ехолокації. Окрім інформації про віддаль до потрібного об'єкта цей локатор надає інформацію про відносну швидкість, розміри, азимут та висоту руху об'єкта. Для виділення цієї інформації з отримуваного сигналу крихітний мозок кажана проводить складні нейронні обчислення. Ехолокація за своїми характеристиками якості та швидкості роботи значно перевищує найскладніші сучасні прилади, створені інженерами.

Що ж дає змогу мозку кажана чи людини домогтися таких вражаючих результатів? При народженні мозок має досконалу структуру, яка дає змогу створювати власні правила на основі того, що ми називаємо "досвідом". Досвід накопичується з часом і особливо масштабні зміни відбуваються в перші два роки життя людини. В цей час формується кістяк загальної структури. Однак розвиток на цьому не зупиняється — він продовжується аж до смерті людини.

Поняття розвитку нейронів пов'язане з поняттям *пластичності (plasticity)* мозку — здатності налаштування нервової системи відповідно до зовнішнього середовища. Саме пластичність відіграє найважливішу роль в роботі нейронів як одиниць обробки інформації мозку. Аналогічно в штучних нейронних мережах робота проводиться з штучними нейронами. Загалом, *нейронна мережа (neural network)* являє собою машину, яка моделює спосіб обробки мозком конкретної задачі. Для того, щоб домогтися високої продуктивності, нейромережі використовують велику кількість взаємозв'язків між елементарними частинками обчислень —

нейронами. Таким чином, можна дати наступне визначення нейромережам:

> *Нейронна мережа — це величезний розподілений паралельний процесор, що складається з елементарних одиниць обробки інформації, які накопичують експериментальні знання і надають їх для подальшої обробки. Нейромережа схожа з мозком з двох точок зору:*
>
> 1. *Знання поступають в нейромережу з навколишнього середовища і використовуються в процесі навчання.*
> 2. *Для накопичення знань використовуються зв'язки між нейронами, що називаються* ***синаптичними вагами****.*

Процедура, яка використовується для процесу навчання, називається *алгоритмом навчання (learning algorithm)*. Ця процедура вистроює в певному порядку синаптичні ваги нейромережі для забезпечення належної структури взаємозв'язків нейронів. Зміна синаптичних ваг — традиційний метод налаштування нейромережі.

4.2 Переваги нейромереж

Цілком очевидно, що свою силу нейромережі черпають, по-перше, з розпаралелювання обробки інформації, по-друге, зі здатності самонавчатися, тобто створювати узагальнення. Під терміном *узагальнення (generalization)* розуміється здатність отримувати обґрунтований результат на основі даних, які не зустрічалися в процесі навчання. Ці властивості дають змогу вирішувати складні (масштабні) задачі, які на сьогодні вважаються важкорозв'язними. Однак на практиці при автономній роботі нейромережі не можуть забезпечити готових рішень. Їх необхідно інтегровувати в складні системи. Зокрема, комплексну задачу можна розбити на послідовність відносно простих, частину з яких можна розв'язати за допомогою нейромереж.

Використання нейромереж забезпечує наступні корисні властивості систем:

- *Нелінійність (nonlinearity).* Штучні нейрони можуть бути лінійними і нелінійними. Нейромережі, побудовані з нелінійних нейронів, самі являються нелінійними. При цьому ця нелінійність особлива,

оскільки вона розподілена по всій мережі. Нелінійність — дуже важлива властивість, особливо якщо сам фізичний механізм, який відповідає за формування вхідного сигналу, також є нелінійним (наприклад, людська мова).

- *Відображення вхідної інформації на вихідну (input-output mapping).* Одна з популярних парадигм навчання — *навчання з вчителем (supervised learning).* Під цим розуміється зміна синаптичних ваг на основі набору маркованих *навчальних прикладів (training sample).* Кожен приклад складається з вхідного сигналу і відповідного йому *бажаного відклику (desired response).* З цієї множини випадковим чином вибирається приклад, а нейромережа модифікує синаптичні ваги для мінімізації відхилення сформованого нейромережею та бажаного виходу. При цьому власне модифікуються *вільні параметри* мережі. Використані приклади можуть бути застосовані знову, але вже в іншому порядку. Це навчання проводиться до того часу, поки зміни синаптичних ваг не стануть незначними.

- *Адаптивність (adaptivity).* Нейромережі мають властивість *адаптовувати* свої синаптичні ваги до змін навколишнього середовища. Зокрема, нейромережі, навчені діяти в певному середовищі, можуть бути легко перенавчені для роботи в умовах незначних коливань параметрів середовища.

- *Очевидність відповіді (evidential response).* В контексті задачі класифікації образів можна розробити нейромережу, яка збиратиме інформацію не тільки для визначення конкретного класу, але й для збільшення *достовірності (confidence)* прийнятого рішення. Надалі ця інформація може використовуватися для виключення сумнівних рішень, що підвищить продуктивність нейромережі.

- *Контекстна інформація (contextual information).* Знання представляються в самій структурі нейромережі з допомогою її стану активації. Кожен нейрон мережі потенційно може бути підданий впливу всіх інших її нейронів.

- *Стійкість до відмов (fault tolerance).* Нейромережі, реалізовані в електронних пристроях, потенційно стійкі до відмов. Це означає, що при несприятливих умовах їх продуктивність падає незначно. Наприклад, якщо пошкоджений який-небудь нейрон чи його зв'язки, то витягування збереженої інформації утруднюється. Проте, приймаючи до уваги розподілений характер збереження інформації в

нейромережі, можна стверджувати, що лише серйозні пошкодження структури нейромережі суттєво вплинуть на її працездатність.

- *Одноманітність аналізу та проектування (uniformity of analysis and design).* Нейромережі — універсальний механізм обробки інформації. Це означає, що одне і те ж проектне рішення нейронної мережі може використовуватися в багатьох предметних областях. Ця властивість проявляється кількома способами:
  - Нейрони в тій чи іншій формі є стандартними складовими частинами *будь-якої* нейромережі.
  - Ця загальність дає змогу використовувати одні й ті ж теорії і алгоритми навчання в різних нейромережевих програмних рішеннях.
  - Модульні мережі можуть бути побудовані на основі інтеграції цілих модулів.
- *Аналогія з нейробіологією (neurobiological analogy).* Будова нейронних мереж визначається аналогією з людським мозком, який є живим доказом того, що стійкі до відмов паралельні обчислення не тільки фізично реалізовувальні, але й є швидким і потужним інструментом розв'язування задач. Нейробіологи розглядають штучні нейромережі як засіб моделювання фізичних явищ. З іншої сторони, інженери постійно намагаються почерпнути в нейробіологів нові ідеї, які виходять за рамки традиційних електросхем.

4.3 Модель нейрона

Нейрон — одиниця обробки інформації в нейромережі. На блок-схемі (рис. 2) показана модель нейрона, яка лежить в основі штучних нейронних мереж. В цій моделі можна виділити три основні елементи:

1. Набір *синапсів (synapse)* або *зв'язків (connecting link)*, кожен з яких характеризується своєю *вагою (weight)* або *силою (strength)*. Зокрема, сигнал x_j на вході синапса j, зв'язаного з нейроном k, множиться на вагу w_{kj}. Важливо звернути увагу на те, в якому порядку вказані індекси синаптичної ваги w_{kj}. На відміну від синапсів мозку синаптична вага штучного нейрона може мати як додатнє, так і від'ємне значення.

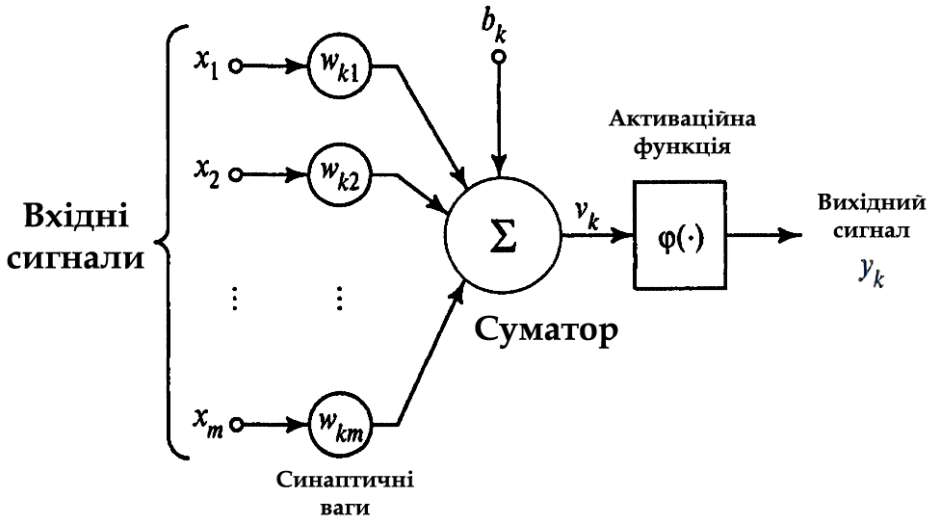

Рис. 2: Схематична модель нейрона

2. *Суматор (adder)* сумує вхідні сигнали, зважені відносно відповідних синапсів нейрона. Цю операцію можна описати як лінійну комбінацію.

3. *Функція активації (activation function)* обмежує амплітуду вихідного сигналу нейрона. Ця функція називається також *функцією стискання (squashing function)*. Зазвичай нормалізований діапазон амплітуд виходу нейрона лежить в інтервалі $[0, 1]$ або $[-1, 1]$.

В модель нейрона, показану на рис. 2, входить також *пороговий елемент (bias)*, який позначений символом b_k. Ця величина вказує на збільшення чи зменшення вхідного сигналу, що подається на функцію активації.

В математичному поданні функціонування нейрона k можна описати наступною парою рівнянь:

$$u_k = \sum_{j=1}^{m} w_{kj} x_j, \tag{1}$$

$$y_k = \varphi(u_k + b_k) \tag{2}$$

де $x_1, x_2, \ldots, x_m$ — вхідні сигнали; $w_{k1}, w_{k2}, \ldots, w_{km}$ — синаптичні ваги нейрона k; u_k — *лінійна комбінація вхідних впливів (linear combiner output)*; b_k — поріг; $\varphi(\cdot)$ — *функція активації (activation function)*; y_k — вихідний

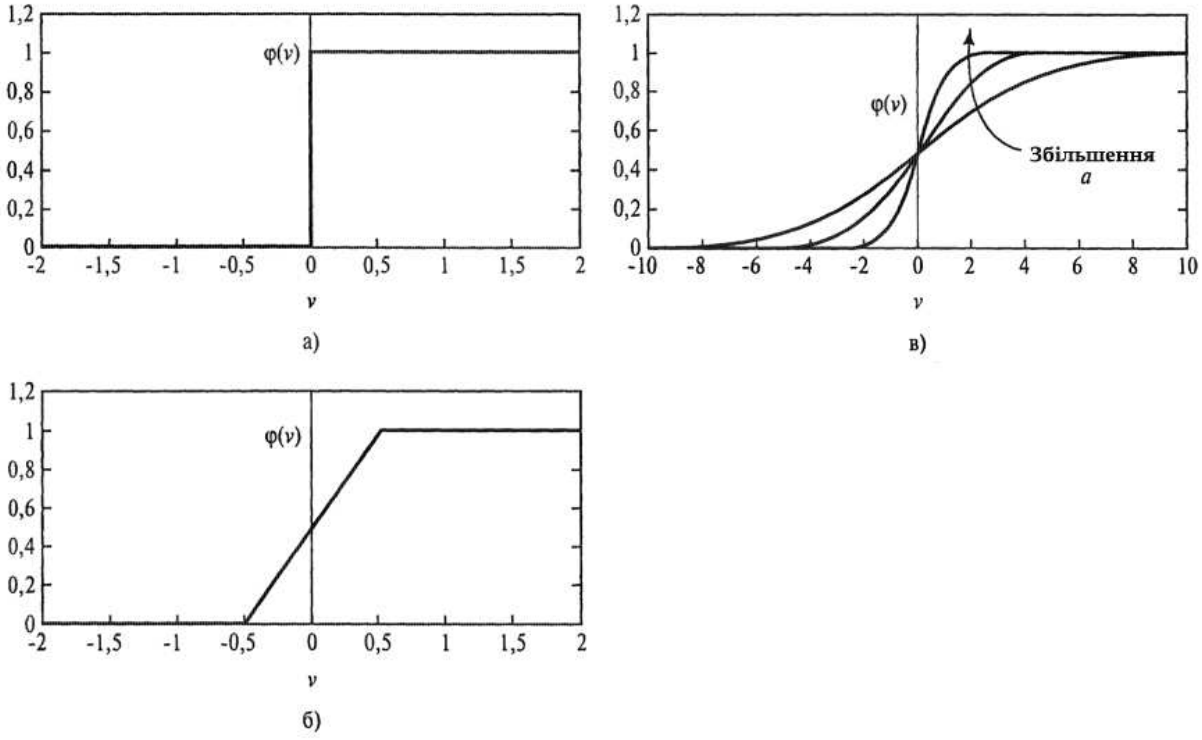

Рис. 3: Основні типи функцій активації: функція одиничного стрибка (а); кусково-лінійна функція (б); сигмоїдальна функція для різних значень параметра а (в)

сигнал нейрона. Використання порога b_k забезпечує ефект *афінного перетворення (affine transformation)* виходу лінійного суматор u_k. В моделі, показаній на рис. 2, постсинаптичний потенціал обчислюється наступним чином:

$$v_k = u_k + b_k. \tag{3}$$

4.4 Типи функцій активації

Функції активації, подані в формулах як $\varphi(v)$, визначають вихідний сигнал нейрона в залежності від індукованого локального поля v. Можна виділити три основні типи функцій активації.

1. *Фукнція одиничного стрибка* або порогова функція (threshold function). Цей тип функції показаний на рис. 3, а і описується наступним чином:

$$\varphi(v) = \begin{cases} 1, & \text{якщо } v \geq 0; \\ 0, & \text{якщо } v < 0; \end{cases} \tag{4}$$

   В технічній літературі ця форма функції одиничного стрибка зазвичай називається *функцією Хевісайда (Heaviside function)*. А мо-

дель нейрона, що базується на цій функції — *моделлю Мак-Каллока-Пітца (McCalloch-Pitts).*

2. *Кусково-лінійна функція (piecewise-linear function.* Кусково-лінійна функція, показана на рис. 3, б, описується наступним виразом:

$$\varphi(v) = \begin{cases} 1, & v \geq +\frac{1}{2}; \\ \frac{1}{2} + v, & -\frac{1}{2} < v < +\frac{1}{2}; \\ 0, & v \leq -\frac{1}{2}, \end{cases} \tag{5}$$

де коефіцієнт підсилення в лінійній області оператора припускається рівним одиниці. Цю функцію активації можна розглядати як *апроксимацію (approximation)* нелінійного підсилювача. Наступні два варіанти можна вважати особливою формою кусково-лінійної функції.

- Якщо лінійна область оператора не досягає порогу насичення, він перетворюється в *лінійний суматор (linear combiner).*
- Якщо коефіцієнт підсилення лінійної області прийняти нескінченно великим, то кусково-лінійна функція вироджується в порогову.

3. *Сигмоїдальна функція (sigmoid function).* Сигмоїдальна функція, графік якої нагадує літеру S є, мабуть, найбільш поширеною функцією, що використовується для створення штучних нейромереж. Це швидкозростаюча функція, яка підтримує баланс між лінійною та нелінійною поведінкою. Прикладом сигмоїдальної функції може бути *логістична функція (logistic function)*, яка задається наступним виразом:

$$\varphi(v) = \frac{1}{1 + exp(-av)}, \tag{6}$$

де a — *параметр нахилу (slope parameter)* сигмоїдальної функції. Змінюючи цей параметр, можна побудувати функції з різною крутизною (див. рис.3, в). Якщо порогова функція може приймати тільки значення 0 і 1, то сигмоїдальна функція може приймати будь-яке значення з діапазону $[0, 1]$. При цьому варто відмітити, що сигмоїдальна функція — диференційовна, в той час як порогова — ні. (Диференційовність активаційної функції відіграє дуже важливу роль в теорії нейронних мереж, зокрема, вона є необхідною для навчання алгоритмом зворотного поширення помилки.)

4.5 Багатошаровий персептрон і його навчання

Зазвичай мережа складається з сукупності сенсорних елементів (вхідних вузлів та вузлів джерела), які утворюють *вхідний прошарок (input layer)*; одного або кількох *прихованих прошарків (hidden layer)* обчислювальних нейронів і одного *вихідного прошарку (output layer)* нейронів. Вхідний сигнал розповсюджується по мережі в прямому напрямку, від прошарку до прошарку. Такі мережі зазвичай називають *багатошаровими персептронами (multilayer perceptron)*.

Багатошарові персептрони мають три характерні ознаки.

1. Кожен нейрон мережі має *нелінійну функцію активації (nonlinear activation function)*. Важливо відмітити, що дана нелінійна функція є гладкою (тобто всюди диференційовною), на відміну від жорсткої порогової функції. Найбільш популярною формою функції, що задовільняє цю умову, є логістична функція:

$$y_j = \frac{1}{1 + exp(-v_j)},$$

де v_j — індуковане локальне поле (тобто зважена сума всіх синаптичних входів плюс порогове значення) нейрона j; y_j — вихід нейрона. Наявність нелінійності відіграє дуже важливу роль, так як в іншому випадку відображення "вхід-вихід" мережі можна звести до звичайного одношарового персептрону. Більше того, використання логістичної функції мотивовано біологічно, оскільки в ній враховується відновлювальна фаза реального нейрона.

2. Мережа містить один або декілька прошарків *прихованих нейронів*, які не є частиною входу або виходу мережі. Ці нейрони дають змогу мережі навчатися вирішенню складних задач, послідовно виокремлюючи найбільш важливі ознаки з вхідного образа (вектора).

3. Мережа має великий рівень *зв'язності (connectivity)*, яка реалізується через синаптичні зв'язки. Зміна рівня зв'язності мережі потребує зміни сукупності синаптичних зв'язків або їх вагових коефіцієнтів.

Комбінація всіх цих властивостей разом зі здатністю до навчання на власному досвіді забезпечує обчислювальну потужність багатошарового персептрона. Однак ці ж якості є також причною неповноти сучасних

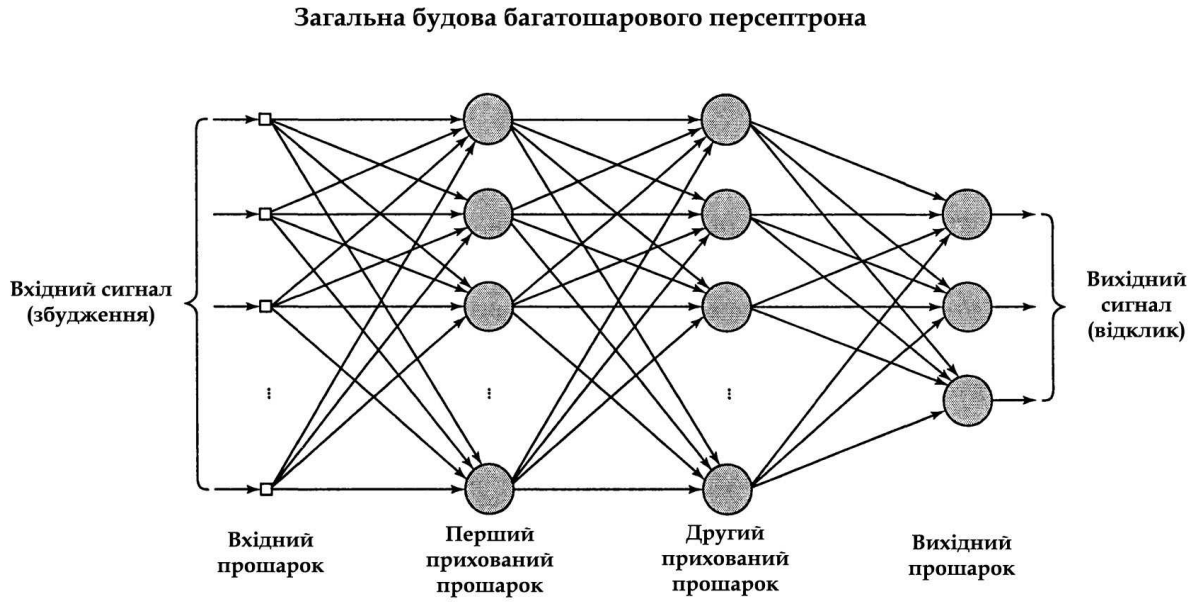

Рис. 4: Архітектурний граф багатошарового персептрона

знань про поведінку мереж такого типу. По-перше, розподілена форма нелінійності і високий рівень зв'язності мережі суттєво ускладнюють теоретичний аналіз багатошарового персептрону. По-друге, наявність прихованих нейронів робить процес навчання більш складним для візуалізації. Саме в процесі навчання необхідно визначити, які ознаки вхідного сигналу треба представляти прихованим нейронам. Тоді процес навчання стає ще більш складним, оскільки пошук повинен виконуватися в дуже широкій області можливих функцій, а вибір повинен проводитися серед альтернативних подань вхідних образів.

На рис. 4 показаний архітектурний граф багатошарового персептрона з двома прихованими прошарками і одним вихідним прошарком.

4.6 Алгоритм зворотного поширення помилки

Багатошарові персептрони успішно застосовуються для розв'язання різноманітних складних задач. При цьому навчання з вчителем відбувається з допомогою такого популярного алгоритму, як *алгоритм зворотного поширення помилки (error back-propagation algorithm)*. Цей алгоритм базується на *корекції помилок (error-correction learning rule)*.

Навчання методом зворотного поширення помилки вимагає два проходи по всіх прошарках мережі: прямого і зворотного. При *прямому проході (forward pass)* образ (вхідний вектор) подається на сенсорні вузли

мережі, після чого поширюється по мережі від прошарку до прошарку. В результаті генерується набір вихідних сигналів, який і є фактичною реакцією мережі на даний вхідний образ. Підчас прямого проходу всі синаптичні ваги фіксовані. Підчас *зворотного проходу (backward pass)* всі синаптичні ваги налаштовуються згідно правила корекції помилок, а саме: фактичний вихід мережі віднімається від бажаного (цільового) відклику, в результаті чого формується сигнал помилки *(error signal)*. Цей сигнал згодом розповсюджується по мережі в напрямку, зворотному напрямку синаптичних зв'язків. Звідси й назва — алгоритм зворотного поширення помилки. Синаптичні ваги налаштовуються з метою максимального наближення вихідного сигналу мережі до бажаного в статистичному сенсі. Процес навчання, який реалізується даним алгоритмом, називається *навчанням на основі зворотного поширення (back-propagation learning)*.

Алгоритм зворотного поширення помилки циклічно обробляє приклади з навчальної множини $\{(x(n), d(n))\}_{n=1}^{N}$, де N — кількість навчальних прикладів, наступним чином:

1. *Ініціалізація (initialization)*. Припускаючи відсутність апріорної інформації, генеруємо синаптичні ваги і порогові значення з допомогою генератора рівномірно розподілених випадкових чисел з середнім значенням 0. Дисперсія вибирається таким чином, щоб стандартне відхилення індукованого локального поля нейронів припадало на лінійну частину сигмоїдальної функції активації (і не досягало області насичення).

2. *Надання прикладів навчання (presentation of training examples)*. В мережу подаються образи з навчальної множини (епохи). Для кожного образу послідовно виконується прямий та зворотній проходи, описані далі в пп. 3 та 4.

3. *Прямий прохід (forward computation)*. Нехай навчальний приклад поданий парою $(\mathbf{x}(n), \mathbf{d}(n))$, де $\mathbf{x}(n)$ — вхідний вектор, який надається вхідному прошарку сенсорних вузлів; $\mathbf{d}(n)$ — бажаний відклик, який надається вихідному прошарку нейронів для формування сигналу помилки. Обчислюємо індуковані локальні поля і функціональні сигнали мережі, проходячи по ній по прошарках в прямому напрямі. Індуковане локальне поле нейрона j прошарку l обчислюється за формулою

$$v_j^{(l)} = \sum_{i=0}^{m_{l-1}} w_{ji}^{(l)}(n) y_i^{(l-1)}(n), \tag{7}$$

де $y_i^{(l-1)}(n)$ — вихідний (функціональний) сигнал нейрона i, розміщеного в попередньому прошарку $l-1$, на ітерації n; $w_{ji}^{(l)}(n)$ — синаптична вага зв'язку нейрона j прошарку l з нейроном i прошарку $l-1$. Для $i=0$ $y_0^{(l)}(n)=+1$, а $w_{j0}^{(l)}(n)=b_j^l(n)$ — поріг, який застосовується до нейрона j прошарку l. Якщо використовується сигмоїдальна функція, то вихідний сигнал нейрона j прошарку l виражається наступним чином:

$$y_j^{(l)}(n)=\varphi_j(v_j^l(n)).$$

Якщо нейрон j знаходиться в першому прихованому прошарку (тобто $l=1$), то

$$y_j^{(0)}(n)=x_j(n),$$

де $x_j(n)$ — j-й елемент вхідного вектора $\mathbf{x}(n)$. Якщо нейрон j знаходиться в вихідному прошарку (тобто $l=L$, де L — глибина мережі), то

$$e_j(n)=d_j(n)-o_j(n), \tag{8}$$

де $d_j(n)$ — j-й елемент вектора бажаного відклику $\mathbf{d}(n)$.

4. *Зворотній прохід (backward computation).* Обчислюємо локальні градієнти вузлів мережі за наступною формулою:

$$\delta_j^{(l)}(n)=\begin{cases} e_j^{(L)}(n)\varphi_j'(v_j^{(L)}(n)), \\ \qquad \text{для нейрона } j \text{ вихідного прошарку } L, \\ \varphi_j'(v_j^{(l)}(n))\sum_{k=0}^{m_{l+1}}\delta_k^{(l+1)}(n)w_{kj}^{(l+1)}(n), \\ \qquad \text{для нейрона } j \text{ прихованого прошарку } l, \end{cases} \tag{9}$$

де штрих в функції $\varphi_j'(\cdot)$ позначає диференціювання по аргументу. Зміна синаптичних ваг прошарку l мережі виконується згідно з узагальненим дельта-правилом

$$w_{ji}^{(l)}(n+1)=w_{ji}^{(l)}(n)+\alpha[w_{ji}^{(l)}(n-1)]+\eta\delta_j^{(l)}(n)y_j^{(l-1)}(n), \tag{10}$$

де η — параметр швидкості навчання (крок навчання); α — стала моменту.

5. *Ітерації (iteration).* Послідовно виконуємо прямий та зворотній проходи (згідно пп. 3, 4), надаючи мережі всі навчальні приклади з епохи, поки не буде досягнуто критерію зупинки.

Зауваження. Порядок надання навчальних прикладів може випадковим чином змінюватися від епохи до епохи. Параметри моменту та швидкості навчання налаштовуються (і зазвичай зменшуються) з ростом кількості ітерацій.

5 Деталі програмної реалізації

5.1 Структура проекту

Структурно програмний проект реалізовано як набір основних об'єктів, які тісно взаємодіють між собою (рис. 5). Такими об'єктами є:

- Об'єкт класу *World (Зовнішнє середовище)*. Він забезпечує завантаження стаціонарних об'єктів типу *WorldObject* віртуального світу з файлу даних, доступ до них всередині програми та візуалізацію їх.
- Об'єкт класу *Car (Автомобіль)*, який відповідає за фізичну модель поведінки автомобіля та завантаження її (моделі) параметрів з файлу конфігурації, а також візуалізацію автомобіля.
- Об'єкт класу *CarController (Контролер)*, який власне і становить собою систему контролю. Він отримує дані про навколишній світ з об'єкту *Зовнішнє середовище*, обробляє їх, передає на вхід об'єкту *Нейромережі* дані про стан автомобіля та, отримавши на виході *Нейромережі* бажані параметри контролю динамічної системи, передає їх об'єкту *Автомобіль*, який змінює власний стан, базуючись на переданих йому нових значеннях прискорення та повороту керма.

Рис. 5: Діаграма взаємодії основних об'єктів проекту

5.2 Фізика автомобіля

5.2.1 Фізика руху по прямій

Спочатку розглянемо автомобіль, що рухається по прямій. Перш за все, на нього діє сили тяги:

$$\vec{F}_{traction} = \vec{u} \cdot EngineForce, \tag{11}$$

де $\vec{u}$ — одиничний вектор напряму руху авто.

Далі введемо сили опору. Перша і найбільш важлива — сила опору повітря, іншими словами — аеродинамічна сила опору. Ця сила пропорційна квадрату швидкості:

$$\vec{F}_{drag} = -C_{drag} \cdot \vec{v} \cdot |\vec{v}|, \tag{12}$$

де C_{drag} — константа, $\vec{v}$ — вектор швидкості.

Також ще є сила тертя:

$$\vec{F}_{rr} = -C_{rr} \cdot \vec{v}, \tag{13}$$

де C_{rr} — константа, $\vec{v}$ — вектор швидкості.

При малих швидкостях сила тертя є основною силою опору,при високих швидкостях $\vec{F}_{drag}$ перевищує по значенню $\vec{F}_{rr}$.

Загальна сила — це векторна сума цих трьох сил.

$$\vec{F}_{long} = \vec{F}_{traction} + \vec{F}_{drag} + \vec{F}_{rr}. \tag{14}$$

Прискорення визначається рівнодійною силою, що діє на авто (в Нютонах), і його масою:

$$\vec{a} = \frac{\vec{F}}{M}. \tag{15}$$

Швидкість автомобіля (в метрах на секунду) визначається як інтеграл прискорення через деякий час dt:

$$\vec{v} = \vec{v} + dt \cdot \vec{a}. \tag{16}$$

В свою чергу, позиція автомобіля визначається за формулою:

$$\vec{p} = \vec{p} + dt \cdot \vec{v}. \tag{17}$$

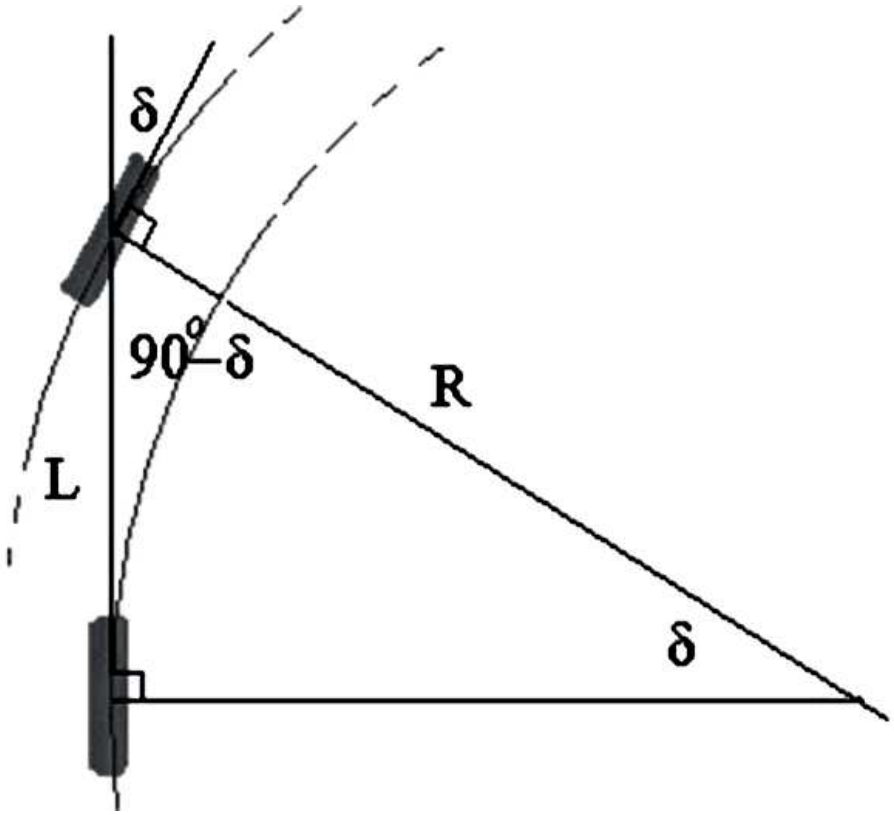

Рис. 6: Радіус кола повороту

5.2.2 Гальмування

При гальмуванні сила тяги замінюється на силу гальмування, направлену в протилежному напрямку:

$$\vec{F}_{long} = \vec{F}_{braking} + \vec{F}_{drag} + \vec{F}_{rr}, \tag{18}$$

$$\vec{F}_{braking} = -\vec{u} \cdot C_{braking}. \tag{19}$$

5.2.3 Рух по кривій

Моделювання фізики поворотів на малих швидкостях відрізняється від моделювання на високих. При малій швидкості колеса рухаються в тому напрямі в якому вони показують.

На рис. 6 зображено радіус кола повороту цілого авто. L — віддаль між передньою та задньою віссю, R — радіус кола, яке описує авто.

Кут δ можна вирахувати за формулою:

$$\sin\delta = \frac{L}{R} \Leftrightarrow R = \frac{L}{\sin\delta}. \tag{20}$$

Якщо кут повороту рівний 0, то радіус кола рівний ∞, тобто ми автомобіль починає рухатись по прямій. Наступним кроком буде розрахунок кутової швидкості за формулою:

$$\omega = \frac{v}{R}. \tag{21}$$

5.3 Графічна оболонка та структури даних

5.3.1 OpenGL

OpenGL (англ. Open Graphics Library — відкрита графічна бібліотека) — специфікація, що визначає незалежний від мови програмування крос-платформовий програмний інтерфейс (API) для написання додатків, що використовують 2D- та 3D-комп'ютерну графіку. Даний інтерфейс містить понад 250 функцій, які можуть використовуватися для малювання складних тривимірних сцен з простих примітивів. OpenGL широко використовується індустрією комп'ютерних ігор та комп'ютерної графіки.

Стандарт OpenGL був розроблений і затверджений у 1992 році передовими компаніями в області розробки апаратного та програмного забезпечення для роботи з комп'ютерною графікою. Метою співпраці було створення апаратно-незалежного інтерфейсу, придатного для реалізації на різних платформах. Основою стандарту стала бібліотека IRIS GL, розроблена компанією Silicon Graphics.

Курт Акелей (Kurt Akeley) і Марк Сіґал (Mark Segal) створили оригінальну специфікацію OpenGL. Кріс Фразаєр (Chris Frazier) редагував версію 1.1, а Джон Ліч (Jon Leech) — версії з 1.2 по 2.0.

На базовому рівні OpenGL — це всього лише специфікація, себто просто документ, який описує набір функцій і їх точну поведінку. На основі цих специфікацій виробники апаратного забезпечення створюють реалізації — бібліотеки функцій, які відповідають заявленій в OpenGL специфікації. Ці реалізації проектуються для того, щоб при можливості використовувати апаратне прискорення функцій OpenGL. Коли апаратне прискорення не підтримується, виконання фукнцій здійснюється за допомогою програмного забезпечення (програмна емуляція). Виробники повинні пройти спеціальні тести на відповідність, перш ніж їхню реалізацію класифікують як реалізацію специфікації OpenGL. Таким чином, розробникам програмного забезпечення необхідно всього лиш навчитися використовувати описані у специфікації функції, а реалізацію залишити розробникам апаратного забезпечення.

Ефективні реалізації OpenGL існують для операційних систем Linux, MacOS X, Microsoft Windows та багатьох UNIX-подібних ОС, а також для таких ігрових приставок, як Sony PlayStation 3. Різні програмні реалізації OpenGL існують для платформ, виробники яких не підтриму-

ють дану специфікацію. Відкрита (open source) бібліотека Mesa — повністю OpenGL-сумісний програмний API. Однак, для того, щоб уникнути витрат на ліцензування, пов'язаних з формалізацією, яка вимагається для офіційного визнання реалізації, Mesa є неофіційною реалізацією специфікації, хоча й повністю з нею сумісна.

OpenGL слугує двом цілям:

- для того, щоб приховувати складнощі встановлення зв'язку комп'ютера з різними 3D акселераторами, надати програмістові один загальноприйнятий API;
- для того, щоб приховати можливості базових інструментальних машин, які відрізняються своїм намаганням виконати підтримку повного набору особливостей OpenGL (використання програмної емуляції, якщо необхідно).

5.3.2 Структура 3DS файлу

Дані про візуальний вигляд об'єктів світу завантажувалися з файлів формату 3DS, які містять інформацію про 3D-модель та текстури об'єкту. Файл 3DS складається із блоків (chunks), кожний із яких складається з певних даних і, можливо, підблоків. Загальний формат блоку:

Зміщення	Довжина	Дані
0	2	Ідентифікатор типу блока (chunk_id)
2	4	Довжина блоку (chunk_len)
4	chunk_len	Дані або підблоки (в залежності від chunk_id)

Табл. 1: Формат блоку.

Конкретні дані містять лише блоки, помічені плюсом, решта блоків складаються з підблоків. (див. табл. 2).
`CHUNK_VERTLIST` — список вершин поточного об'єкта.
`CHUNK_FACELIST` — список граней поточного об'єкта.
`CHUNK_FACEMAT` — назва матеріалу і список всіх граней об'єкта, яким назначений цей матеріал. Таких блоків може бути декілька, а може і взагалі не бути.
`CHUNK_MAPLIST` — текстурні координати вершин поточного об'єкта. Текстурні координати задаються в діапазоні $[0, 1]$. Точка з U=0, V=0 — верхній лівий край текстури; U=1, V=1 — правий нижній край.

CHUNK_MAIN	= 0x4D4D	[–] сцена
CHUNK_OBJMESH	= 0x3D3D	[–] різноманітні об'єкти
CHUNK_OBJBLOCK	= 0x4000	[+] об'єкт
CHUNK_OBJBTRIMESH	= 0x4100	[–] trimesh-об'єкт
CHUNK_VERTLIST	= 0x4110	[+] список вершин
CHUNK_FACELIST	= 0x4120	[+] список граней
CHUNK_FACEMAT	= 0x4130	[+] матеріали граней
CHUNK_MAPLIST	= 0x4140	[+] текстурні координати
CHUNK_TRMATRIX	= 0x4160	[+] матриця переходу
CHUNK_CAMERA	= 0x4700	[+] об'єкт-камера
CHUNK_MATERIAL	= 0xAFFF	[–] матеріал
CHUNK_MATNAME	= 0xA000	[+] назва матеріалу
CHUNK_TEXTURE	= 0xA200	[–] текстура матеріалу
CHUNK_MAPFILE	= 0xA300	[+] ім'я файлу матеріалу
CHUNK_KEYFRAMER	= 0xB000	[–] інформація про анімацію
CHUNK_TRACKINFO	= 0xB002	[–] поведінка об'єкта
CHUNK_TRACKOBJNAME	= 0xB010	[+] назва об'єкту
CHUNK_TRACKPIVOT	= 0xB013	[+] центр повороту об'єкта
CHUNK_TRACKPOS	= 0xB020	[+] траєкторія
CHUNK_TRACKROTATE	= 0xB021	[+] траєкторія повороту
CHUNK_TRACKCAMERA	= 0xB003	[–] поведінка камери
CHUNK_TRACKFOV	= 0xB023	[+] поведінка FOV камери
CHUNK_TRACKROLL	= 0xB024	[+] поведінка ROLL камери
CHUNK_TRACKCAMTGT	= 0xB004	[–] поведінка "цілі"камери

Табл. 2: Список ідентифікаторів типів блоків

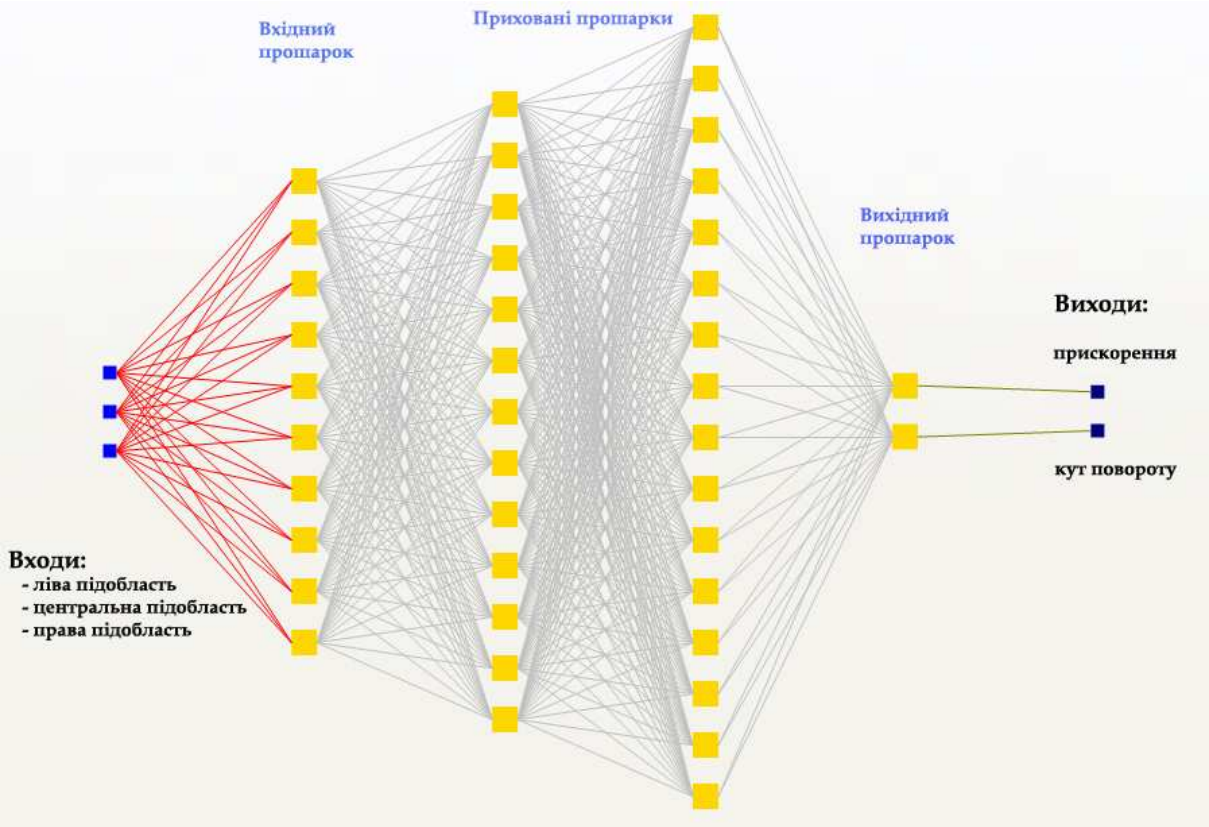

Рис. 7: Конфігурація нейромережі, використаної в проекті (3-10-13-16-2)

5.4 Конфігурація нейромережі

Нейромережа, яка керує автомобілем, повинна мати три входи та два виходи. На входи подаються дані про віддалі об'єктів в межах однієї з трьох підобластей поля зору автомобіля (лівої, центральної та правої), а на виході отримуються дані про необхідну зміну прискорення автомобіля та кута повороту коліс. Обидва параметри знаходяться в діапазоні $[-1, 1]$. В випадку прискорення від'ємне значення відповідає гальмуванню або руху автомобіля назад, в залежності від того, в яку сторону рухається автомобіль в даний момент. Для кута повороту значення -1 відповідає максимальному повороту наліво, 0 — руху прямо, -1 — максимальному повороту направо.

В результаті експериментів з різноманітними конфігураціями нейромережі, було обрано конфігурацію з п'ятьма прошарками. На вході — 3 сенсорних нейрони, в трьох прихованих прошарках міститься 10, 13 та 16 прихованих нейрона, в вихідному прошарку — два вихідних, які відповідають прискоренню та куту повороту. Схематично архітектуру мережі подано на рис. 7.

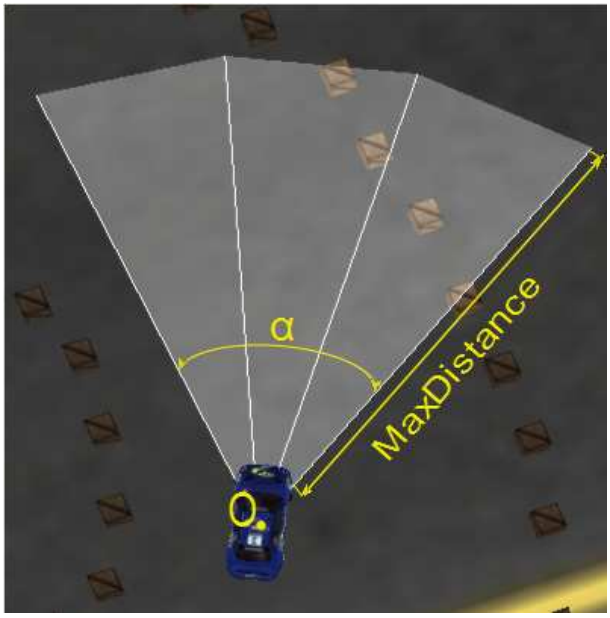

Рис. 8: Поле зору автомобіля

5.5 Поле зору

Поле зору автомобіля має вигляд сектора з початковою точкою O, радіусом $MaxDistance$, рівним максимальній дальності видимості автомобіля, та кутом α, рівним куту огляду автомобіля. Вся область видимості розбивається на три рівні частини з кутом $\frac{\alpha}{3}$. Кожній області — лівій, центральній та правій — відповідає свій вхід нейромережі (рис. 7).

Значення, яке подається на відповідний вхід нейромережі обчислюється наступним чином. Знаходиться об'єкт, який хоча б частково попадає у заданий сектор, обчислюється відстань від точки O до найближчої точки об'єкта, яка знаходиться в заданому секторі. Значенням, яке подається на відповідний вхід нейромережі, буде відношенням знайденої відстані до значення максимальної дальності видимості $MaxDistance$, або 1.0, якщо немає жодного об'єкту, який би попадав в даний сектор:

$$Input = \begin{cases} \frac{|O - ClosestPoint|}{MaxDistance}, & \text{якщо є об'єкт, який попадає у сектор;} \\ 1.0, & \text{якщо жоден об'єкт не перетинає сектор.} \end{cases} \tag{22}$$

5.6 Алгоритм визначення попадання об'єкта в сектор

Для визначення того, чи попадає об'єкт, поданий своїм обмежуючим багатокутником, в заданий сегмент, потрібно здійснити наступні дії:

1. Перевіряємо кожну кутову точку на попадання у заданий сектор (як це здійснити буде пояснено нижче). Якщо точка попадає в сектор, визначаємо відстань від неї, до початку сектора. Якщо ця відстань менша, ніж поточне значення мінімальної відстані $MinDistance$ (яке на початку ініціалізується значенням $MaxDistance$), то присвоюємо нове значення мінімальної відстані.

2. Перевіряємо кожен відрізок обмежуючого багатокутника. Якщо хоча б одна з крайніх точок відрізка попадає в сектор або відрізок перетинає хоча б одну з бокових сторін сектора, то відрізок попадає в сектор. Тоді знаходимо мінімум відстаней серед всіх цих точок і при потребі змінюємо поточний мінімум відстані.

3. Повертаємо відношення $\frac{MinDistance}{MaxDistance}$ як результат.

Розглянемо тепер алгоритм визначення попадання точки в сектор. Але перед тим визначимо *скалярний та векторний добутки векторів*, а також покажемо як здійснити поворот вектора на певний кут.

Озн. *Скалярним добутком* двох двовимірних векторів $\vec{v}_1(x_1; y_1)$ та $\vec{v}_2(x_2; y_2)$ називається дійсне число, яке обчислюється за наступним правилом:

$$(\vec{v}_1, \vec{v}_2) = \vec{v}_1 \cdot \vec{v}_2 = x_1x_2 + y_1y_2. \tag{23}$$

Озн. *Векторним добутком* двох двовимірних векторів $\vec{v}_1(x_1; y_1)$ та $\vec{v}_2(x_2; y_2)$ називається дійсне число, яке обчислюється за наступним правилом:

$$[\vec{v}_1, \vec{v}_2] = \vec{v}_1 \times \vec{v}_2 = \begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix} = x_1y_2 - y_1x_2. \tag{24}$$

Вектор $\vec{v}(x, y)$, повернений на φ радіан проти годинникової стрілки, можна визначити наступним чином:

$$\vec{v}_\varphi = (xcos(\varphi) - ysin(\varphi), \quad xsin(\varphi) + ycos(\varphi)). \tag{25}$$

Визначення попадання точки в сектор. Якщо сектор заданий одиничним напрямним вектором $\vec{v}$, початковою точкою O, центральним кутом α та радіусом R, і потрібно визначити чи попадає точка P в цей сектор, то виконуємо наступні дії:

1. Знаходимо напрямні вектори правої та лівої сторін сегменту, $\vec{v}_L$ та $\vec{v}_R$, повернувши напрямний вектор на $\frac{\alpha}{2}$ та $-\frac{\alpha}{2}$ радіан відповідно.

2. Точка P належить даному сектору тоді і тільки тоді, коли вона лежить між прямими, що задаються напрямними векторами $\vec{v}_L$ та $\vec{v}_R$, вектор $\overrightarrow{OP}$ утворює з вектором $\vec{v}$ гострий кут та відстань від точки P до $O \leq R$. Цьому критерію відповідає наступна умова:

$$((\vec{v}_L \times \overrightarrow{OP})(\vec{v}_R \times \overrightarrow{OP}) < 0) \wedge (\vec{v} \cdot \overrightarrow{OP} > 0) \wedge (|P - O| < R). \quad (26)$$

5.7 Навчальна вибірка

Навчання мережі — один з основних моментів в реалізації проекту. Правильно вибрана множина навчальних прикладів може дати чудові результати на практиці. В той же час, невеликі зміни у навчальних прикладах можуть призвести до суттєвих змін в поведінці нейромережі і, відповідно, керованого автомобіля. Після численних експериментів, ми зупинилися на наступній навчальній вибірці (див. табл. 3).

Кожен рядок табл. 3 відповідає одному навчальному прикладу. Перші три стовпці відповідають трьом входам нейромережі, а саме — відстаням до об'єктів в межах лівої, центральної та правої підобластей відповідно. Наступні два — прискорення та поворот, в заданому порядку.

При створенні навчальної множини за основу була взята навчальна множина з [3], адаптована до особливостей нашої реалізації автомобіля, оскільки фізика автомобіля в [3] суттєво відрізняється від реалізованої в нашому проекті. До кожної ситуації підбиралися параметри прискорення та повороту, які, на нашу думку, найкраще відповідають кожній конкретній ситуації. Наприклад, ситуації, коли перед автомобілем немає жодних перешкод, відповідає вхід $(1.0, 1.0, 1.0)$. Найбільш природнім у цій ситуації буде рух вперед з максимальним прискоренням без зміни напрямку. Відповідно, бажаним виходом був прийнятий вектор $(1.0, 0.0)$, що якраз відповідає максимальному прискоренню та нульовому повороту керма. Ще одним прикладом може слугувати різкий поворот направо, якому відповідає вхід $(0.2, 0.3, 0.4)$. Цілком адекватним у цій ситуації буде невелике значення прискорення та не доволі сильний поворот керма направо. Тому цьому входу поставлено у відповідність вихід $(0.4, 0.7)$.

Шляхом низки експериментів, під час яких змінювалися бажані виходи, після чого автомобіль тестувався на адекватність поведінки на кількох типових трасах, було підібрано найбільш вдалий набір вихідних параметрів.

Табл. 3: Навчальна вибірка

Вхідні нейрони			Вихідні нейрони	
Зліва	По центру	Справа	Прискорення	Поворот
1,0	1,0	1,0	1,0	0,0
0,5	1,0	1,0	0,8	0,4
1,0	1,0	0,5	0,8	-0,4
1,0	0,5	1,0	0,4	-0,4
0,5	1,0	0,5	0,5	0,0
0,0	0,0	0,0	-1,0	-1,0
0,5	0,5	0,5	-0,5	-1,0
0,0	1,0	1,0	0,4	0,6
1,0	1,0	0,0	0,4	-0,6
1,0	0,0	1,0	-0,8	-0,8
0,0	1,0	0,0	1,0	0,0
0,0	0,0	1,0	0,7	1,0
1,0	0,0	0,0	0,7	-1,0
0,3	0,4	0,1	0,4	-0,5
0,1	0,4	0,3	0,4	0,5
0,0	0,1	0,2	0,3	0,8
0,2	0,1	0,0	0,3	-0,8
0,0	0,3	0,6	0,6	0,8
0,6	0,3	0,0	0,6	-0,8
0,2	0,3	0,4	0,4	0,7
0,4	0,3	0,2	0,4	-0,7

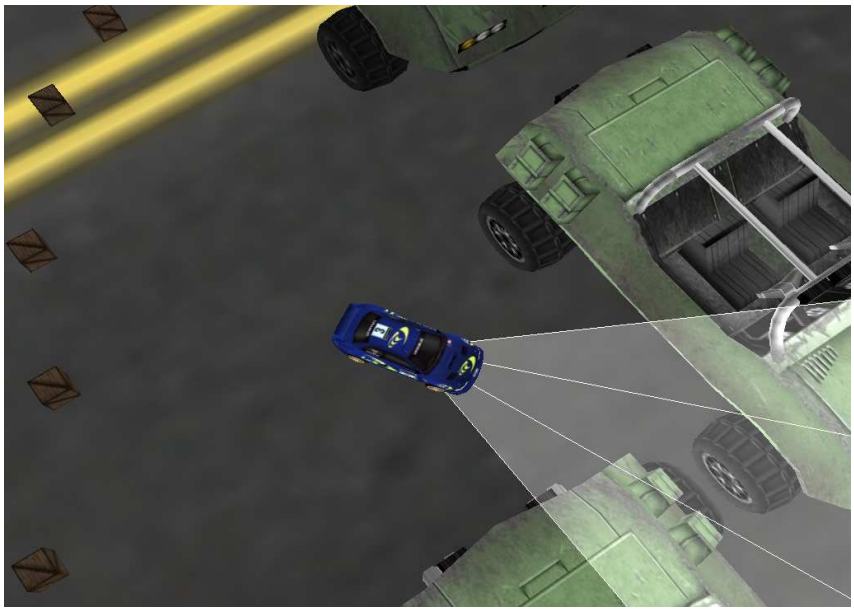

Рис. 9: Автомобіль застряг в куті

Навчена на цій навчальній множині, нейромережа адекватно реагує на перешкоди у абсолютній більшості випадків і забезпечує природну поведінку автомобіля. Щоправда, у випадку попадання у кут автомобілю не завжди вдається вийти з цієї ситуації і після кількох спроб розвернутися автомобіль зупиняється на місці (застрягає) (див. 9). Це пов'язано з тим, що в такій ситуації потрібно робити розворот назад, проте домогтися цього підбором параметрів навчальної множини не вдалося. Можливих виходів з ситуації ми бачимо два:

1. Введення у представлення стану динамічної системи (автомобіля) більшої кількості параметрів, наприклад, поточної швидкості. В цьому випадку, можливо, можна було б домогтися хорошої реакції автомобіля в ситуації, в яких вона на даний час застрягає. Проте збільшення параметрів стану потребує значного збільшення кількості навчальних прикладів, а також їх дуже точний підбір. Була спроба ввести в стан системи параметр поточної швидкості, проте бажаних результатів вона не дала. Тим не менше, це не означає, що неможливо досягти вказаної мети шляхом більш точного підбору навчальних прикладів.

2. "Ручний контроль" над автомобілем. Можна відслідковувати рух автомобіля і, якщо протягом певного часу автомобіль не зрушив з місця,—"перебирати" контроль від нейромережі на певну внутрішню логіку програми, робити розворот і вже тоді вертати контроль над

автомобілем назад нейромережі. Цей підхід пов'язаний зі значним ускладненням логіки роботи програми, оскільки потрібно виконувати контроль за об'єктами, що знаходяться позаду автомобіля. Та й загалом це був би відхід від ідеї контролю за динамічною системою з використанням нейромережі.

Таким чином, на даний момент не було знайдено задовільного рішення вказаної проблеми. В майбутньому можна спробувати вдосконалити управління автомобілем шляхом використання іншої архітектури нейромережі, яка б давала змогу проводити самонавчання без участі "вчителя".

6 Висновки

В даній курсовій роботі ми спробували реалізувати систему контролю автомобіля на базі нейромережі прямого поширення сигналу, дослідити коректність та адекватність управління, реалізованого нею.

Нами була створена інтерактивна програма, яка забезпечує візуалізацію віртуального середовища та руху керованого автомобіля у ньому, а також кілька додаткових можливостей, зокрема наглядна візуалізація поля зору (рис. 10).

Рис. 10: Інтерфейс програми

Результати не можна охарактеризувати однозначно. З однієї сторони, система контролю на базі нейромережі достатньо добре керувала автомобілем, успішно оминаючи перешкоди та забезпечуючи безперервний рух автомобіля в середовищі, про яке вона не мала жодної попередньої інформації. Більше того, для підвищення рівня "природності" керування, системі контролю надавалася лише інформація про об'єкти, які попадали в поле зору автомобіля, що являло собою сектор з визначеним радіусом (дальністю бачення) та центральним кутом (кутом огляду). Базуючись лише на даних про віддаль до найближчих об'єктів в полі зору, система контролю забезпечувала завчасне оминання перешкоди з плавною зміною швидкості та напрямку руху автомобіля. Очевидно, що велику роль

в досягненні вказаної природності та адекватності керування відіграла вдало підібрана навчальна множина.

З іншого боку, найбільшою проблемою, з якою довелося зіткнутися, була наявність ситуацій, в яких автомобіль зупинявся і система контролю не могла вивести його з нерухомого стану. Це, зокрема, ситуація, подана на рис. 9, в якій автомобіль заїхав у глухий кут і застряг, не зумівши здійснити розворот назад. Ми вже частково розглянули цю проблему та можливі варіанти її рішення, які тим не менше, здаються нам не надто хорошими.

Слід зазначити, що ще однією причиною (окрім вказаної обмеженості визначеного стану динамічної системи) такої поведінки є особливість фізичної моделі автомобіля. Мається на увазі природа автомобіля — для того щоб здійснити поворот, чи, тим більше, розворот, автомобіль повинен здійснювати поступальний рух, що в умовах, коли вже відбулося "застрягання" дуже проблематично. Можливим розв'язанням даної проблеми є зміна типу транспортного засобу, керованого системою керування на базі нейромережі. Якщо взяти транспортний засіб, здатний здійснювати поворот без поступального руху, то можна значно зменшити ймовірність застрягання керованого засобу. Таким транспортним засобом, для прикладу, може бути танк, в якому шляхом незалежного обертання гусениць в різні сторони можна домогтися розвороту на будь-який кут, стоячи при цьому на місці.

Ще одним перспективним вдосконаленням нашої системи контролю може бути застосування нейромереж іншого, ніж застосованого нами багатошарового персептрона, типу. Перейшовши до використання інших типів мереж, можна домогтися навчання без вчителя, тобто шляхом "проб та помилок", що, крім потенційного підвищення якості управління динамічною системою, має ще один позитивний момент — позбавлення необхідності створення навчальної множини та трудомісткого і нетривіального підбору параметрів навчальних прикладів. Застосування інших типів нейромереж потребує більш детального дослідження.

Література

[1] Крак Ю.В., Левошич О.Л. Теорія керування. Навчальний посібник для студентів факультету кібернетики спеціальності прикладна математика — Київ, 2001

[2] Саймон Хайкин. Нейронные сети: полный курс, 2-е издание : Пер. с англ. — М. : Издательский дом "Вильямс", 2006

[3] Philippe Kunzle. Vehicle Control with Neural Networks — http://www.gamedev.net/reference/articles/article1988.asp

[4] Wikipedia. Control Theory — http://en.wikipedia.org/wiki/Control_theory

[5] Dave Astle, Kevin Hawkins. Beginning OpenGL Game Programming — Boston, MA 02210, 2004

[6] Jackie Neider, Tom Davis. The Official Guide to Learning OpenGL, Version 1.1 — Addison-Wesley Professional, 1997

[7] Gamedev.ru. Физика автомобиля для игр — http://www.gamedev.ru/articles/?id=70108

[8] Evan Pipho. Focus On 3D Models — Premier Press, a division of Course Technology, 2003