НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
Кафедра комп'ютерних мультимедійних технологій

«Штучний інтелект в засобах мультимедіа»

Методичні рекомендації і завдання
до виконання домашнього завдання з дисципліни
«Штучний інтелект в засобах мультимедіа»
для підготовки фахівців освітнього ступеня «Магістр»

Галузь знань: 18 «Виробництво та технології»
Спеціальність: 186 «Видавництво та поліграфія»
Спеціалізація: «Технології електронних мультимедійних видань»

Укладач: к.т.н., доцент Чаплінський Ю.П.

Тема: Побудова та використання нейронних мереж

Мета: 1. Ознайомитися з особливостями побудови, навчання та використання нейронних мереж .

2. Побудова нейронних мереж в середовищі MATLAB.

Теоретичні відомості

Нейронні мережі (НМ) широко використовуються для розв'язання різноманітних завдань. Основи теорії і технології застосування НС широко представлені в пакеті MATLAB графічним інтерфейсом користувача NNTool.

Щоб запустити NNTool, необхідно виконати команду в командному вікні MATLAB:

>> nntool

після цього з'явиться головне вікно **NNTool, іменоване "Вікном управління мережами і даними" (Network / Data Manager)** (рис. 1).

Рисунок 1. Головне вікно NNTool

Панель "Мережі та дані" (Networks and Data) має функціональні клавіші з наступними призначеннями:

Допомога (Help) - короткий опис керуючих елементів цього вікна;

Нові дані (New Data) - виклик вікна, що дозволяє створювати нові набори даних;

Нова мережа (New Network) - виклик вікна створення нової мережі;

Імпорт (Import) - імпорт даних з робочого простору MATLAB в пространст- у змінних НСTool;

Експорт (Export) - експорт даних з простору змінних NNTool в робочий простір MATLAB;

Вид (View) - графічне відображення архітектури обраної мережі;

Видалити (Delete) - видалення обраного об'єкта.

На панелі **"Тільки мережі" (Networks only)** розташовані клавіші для

роботи виключно з мережами. При виборі покажчиком миші об'єкта будь-якого іншого типу, ці кнопки стають неактивними.

При роботі з NNTool важливо пам'ятати, що клавіші View, Delete, Initialize, Simulate, Train і Adapt (зображені на рис. 1 як неактивні) діють стосовно того об'єкту, який зазначений у даний момент виділенням. Якщо такого об'єкта немає, або над виділеним об'єктом неможливо зробити вказане дію, відповідна клавіша неактивна.

Одним з найбільш важливих властивостей нейронних мереж є здатність апроксимувати та бути універсальними аппроксіматорами. Сказане означає, що за допомогою нейронних ланцюгів можна апроксимувати як завгодно точно безперервні функції багатьох змінних. Розглянемо створення нейронної мережі за допомогою NNTool на прикладі.

Необхідно виконати апроксимацію функції наступного вигляду:

$\sin\left(\frac{5\pi x}{N}+\sin\left(\frac{7\pi x}{N}\right)\right)$, де x € 1÷N, а N - кількість точок функції.

Створення мережі

Перед створенням мережі необхідно підготовити набір навчальних та цільових даних.

Тепер пфдготуємо набір навчальних даних (1, 2, 3, ..., 100), задавши їх наступним виразом: [1: 100].

Скористаємося кнопкою **New Data**. У вікні слід провести зміни, показані на рис. 2, і натиснути клавішу **"Створити" (Create)**.

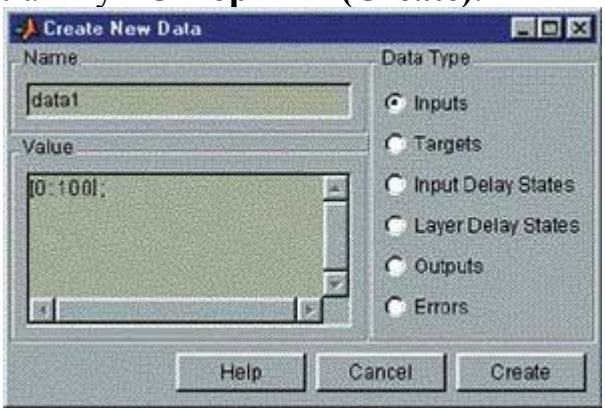

Рисунок 2. Завдання вхідних векторів

Після цього у вікні управління з'явиться вектор **data1** в розділі **Inputs**. Вектор цілей задається схожим чином (рис. 3).

Введемо в поле **"Значення" (Value)** вікна створення нових даних вираз:

sin (5 * pi * [1: 100] / 100 + sin (7 * pi * [1: 100] / 100)).

Ця крива є відрізком періодичного коливання з частотою 5p / N, модульованого за фазою гармонійним коливанням з частотою 7N (рис. 9).

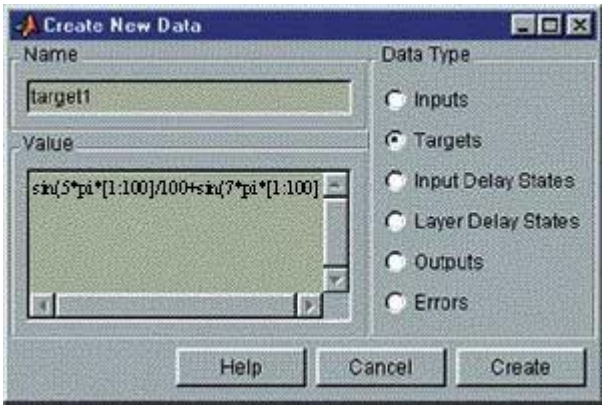

Рисунок 3. Завдання цільового вектора

Після натискання на **Create** в розділі **Targets** з'явиться вектор **target1**.

Тепер слід приступити до створення нейронної мережі. Вибираємо кнопку **New Network** та заповнюємо форму. Виберемо персептрон **(Feed-Forward Back Propagation)** с тринадцятьма сигмоїдними (**TANSIG**) нейронами прихованого шару і одним **лінійним (PURELIN)** нейроном вихідного шару. Навчання будемо проводити, використовуючи **алгоритм Левенберга-Маркардта (Levenberg-Marquardt)**, який реалізує функція **TRAINLM**. Функція помилки - **MSE**.

Поля несуть такі смислові навантаження:

- **Ім'я мережі (Network Name)** - це ім'я об'єкту створюваної мережі.

- **Тип мережі (Network Type)** - визначає тип мережі та в контексті обраного типу являє для введення різні параметри в частині вікна, розташованої нижче цього пункту. Таким чином, для різних типів мереж вікно змінює свій вміст.

- **Вхідні діапазони (Input ranges)** - матриця з числом рядків, що дорівнює кількості входів мережі. Кожен рядок представляє собою вектор з двома елементами: перший - мінімальне значення сигналу, яке буде подано на відповідний вхід мережі при навчанні, другий - максимальне. Для спрощення введення цих значень передбачений випадає "Отримати з входу" (Get from input), що дозволяє автоматично сформувати необхідні дані, вказавши ім'я вхідної змінної.

- **Кількість нейронів (Number of neurons)** - число нейронів в шарі.

- **Передатна функція (Transfer function)** - в цьому пункті вибирається передавальна функція (функція активації) нейронів.

- **Функція навчання (Learning function)** - функція, що відповідає за оновлення ваг та зміщень мережі в процесі навчання.

За допомогою клавіші **"Вид" (View)** можна подивитися архітектуру створюваної мережі (рис. 4).

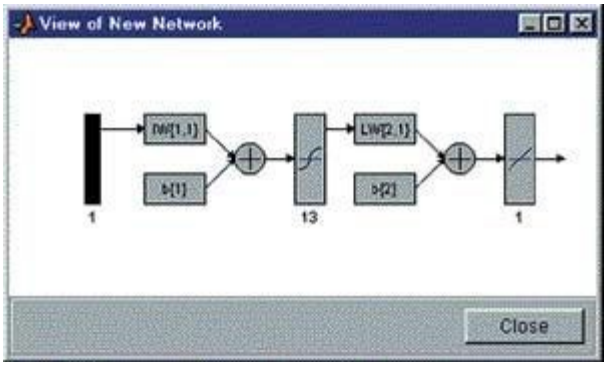

Рисунок 4. Архітектура мережі для розв'язання задачі апроксимації

Вікно попереднього перегляду можна закрити, натиснувши клавішу **"Закрити" (Close)**, і підтвердити намір створити мережу, натиснувши **"Створити" (Create)** в вікні створення мережі.

В результаті виконаних операцій в розділі **"Мережі" (Networks)** головного вікна NNTool з'явиться об'єкт з ім'ям **network1**.

Навчання мережі

Наша мета - побудувати нейронну мережу, яка виконує апроксимацію заданої функції. Очевидно, не можна розраховувати на те, що відразу після етапу створення мережі остання буде забезпечувати правильний результат (правильне співвідношення "вхід / вихід"). Для досягнення мети мережу необхідно належним чином навчити, тобто підібрати відповідні значення параметрів. В MATLAB реалізовано більшість відомих алгоритмів навчання нейронних мереж. Створюючи мережу, ми вказали TRAINLM в якості опції, що реалізує алгоритм навчання.

Повернемося до головного вікна NNTool. На даному етапі інтерес представляє нижня панель "Тільки мережі" (Networks only). Натискання будь-якої з клавіш на цій панелі викличе вікно, на множині вкладок якого представлені параметри мережі, необхідні для її навчання і прогону, а також відображають поточний стан мережі.

Відзначивши покажчиком миші об'єкт мережі network1, викличемо вікно управління мережею натисканням кнопки. Перед нами виникне вкладка "Train" вікна властивостей мережі, що містить, в свою чергу, ще одну панель вкладок (рис. 5). Їх головне призначення - управління процесом навчання. На вкладці "Інформація навчання" (Training info) потрібно вказати набір навчальних даних в поле "Входи" (Inputs) і набір цільових даних в поле "Цілі" (Targets). Поля "Виходи" (Outputs) і "Помилки" (Errors) NNTool заповнює автоматично. При цьому результати навчання, до яких відносяться виходи і помилки, будуть зберігатися в змінних з зазначеними іменами.

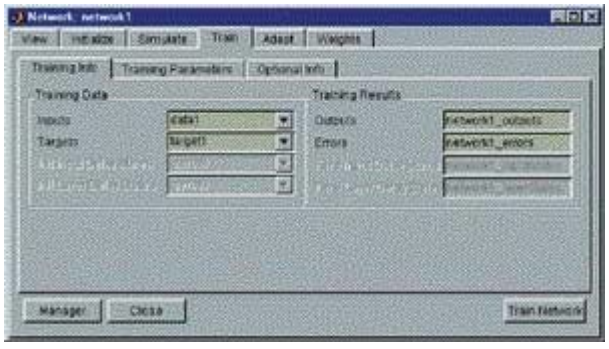

Рисунок 5. Вікно параметрів мережі, відкрите на вкладці "навчання"

(Train)

Завершити процес навчання можна, керуючись різними критеріями. Можливі ситуації, коли можна зупинити навчання, вважаючи достатнім деякий інтервал часу. З іншого боку, об'єктивним критерієм є рівень помилки.

На вкладці **"Параметри навчання" (Training parameters)** для нашої мережі (рис. 6) можна встановити наступні поля:

Кількість епох (epochs) - визначає число епох (інтервал часу), по закінченні яких навчання буде припинено. Епохою називають одноразове подання всіх навчальних вхідних даних на входи мережі.

Досягнення мети або потрапляння (goal) - тут задається абсолютна величина функції помилки, при якій мета буде вважатися досягнутою.

Період оновлення (show) - період оновлення графіка кривої навчання, виражений числом епох.

Час навчання (time) - після закінчення зазначеного в ньому тимчасового інтервалу, вираженого в секундах, навчання припиняється.

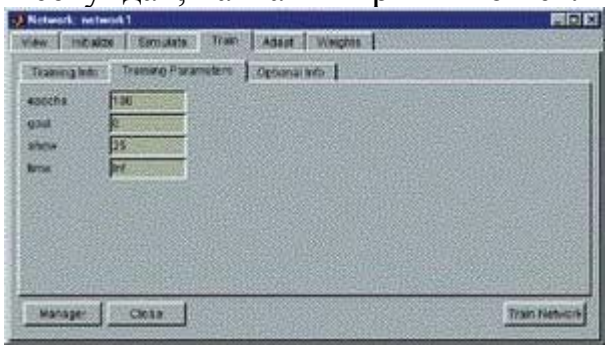

Рисунок 6. Вкладка параметрів навчання

Наступна вкладка **"Необов'язкова інформація" (Optional Info)** показана на рис. 7.

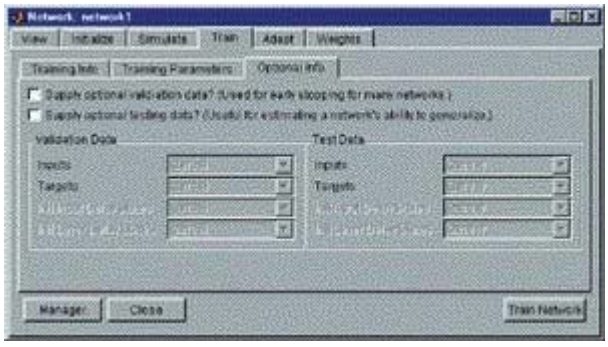

Рисунок 7. Вкладка необов'язковою інформації

Щоб почати навчання, потрібно натиснути кнопку **"Навчити мережу" (Train Network)** на вкладці **"Навчання" (Train)**. Після цього, якщо в поточний момент мережа не задовольняє жодному з умов, зазначених в розділі параметрів **навчання (Training Parameters)**, з'явиться вікно, що ілюструє динаміку цільової функції - *криву навчання*. У нашому випадку графік може виглядати так, як показано на рис. 8. За допомогою кнопки **"Зупинити навчання" (Stop Training)** можна припинити цей процес.

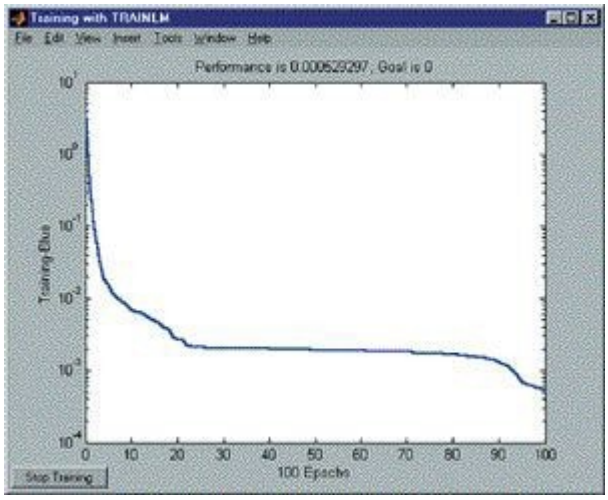

Рисунок 8. Крива навчання в завданню апроксимації

З рисунка 8 видно, наскільки зменшилася помилка апроксимації за 100 епох навчання. Форма кривої навчання на останніх епохах говорить також про те, що точність наближення може бути підвищена.

Відкриємо вкладку "**Прогін" (Simulate)** та виберемо в списку, що випадає **"Входи" (Inputs)** заготовлені дані. У цьому завданню природно використовувати той же набір даних, що і при навчанні **data1**. При бажанні можна встановити прапорець **"Задати мети" (Supply Targets)**. Тоді в результаті прогону додатково будуть розраховані значення помилки. Натискання кнопки **"Прогін мережі" (Simulate Network)** запише результати прогону в змінну, ім'я якої зазначено в поле **"Виходи" (Outputs)**. Тепер можна побудувати два графіка функції в одному вікні: перший графік - задана функція $\sin\left(\frac{5\pi x}{N}+\sin\left(\frac{7\pi x}{N}\right)\right)$, другий - в якості аргументу значення вектора з **"Входи "(Inputs)**, а в якості значень функції значення вектора **"Виходи "(Outputs)**. Отримаємо наступний графік функцій (рис. 9), який ілюструє різницю між цільовими даними і отриманої апроксимуючої кривої.

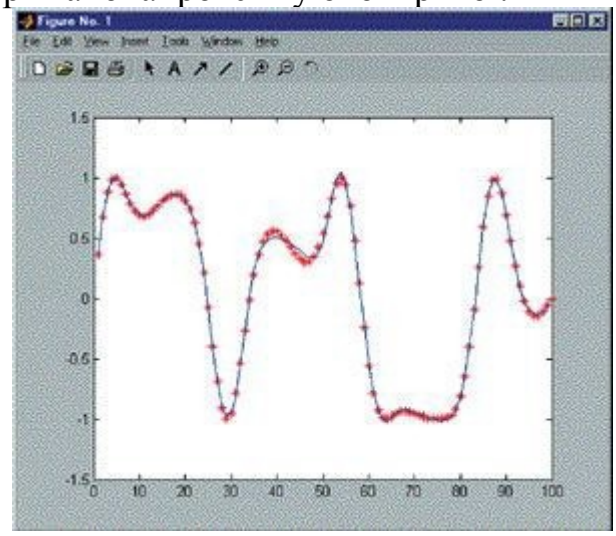

Рисунок 9. Червона крива - цільові дані, синя крива - апроксимуюча функція

Слід зауважити, що мережа створюється ініціалізованою, тобто значення ваг та зміщень задаються певним чином. Перед кожним наступним досвідом навчання зазвичай початкові умови оновлюються, для чого на

вкладці **"Ініціалізація" (Initialize)** передбачена функція ініціалізації. Так, якщо потрібно провести кілька незалежних дослідів навчання, ініціалізація ваг і зміщень перед кожним з них здійснюється натисканням кнопки **"Ініціалізувати ваги" (Initialize Weights)**.

Повернемося до вкладки **"Необов'язкова інформація" (Optional Info)** (рис. 7). Щоб зрозуміти, з якою метою служать представлені тут параметри, необхідно обговорити два поняття: *перенавчання* та *узагальнення*.

При виборі нейронної мережі для розв'язання конкретного завдання важко передбачити її порядок. Якщо вибрати невиправдано великий порядок, мережа може виявитися занадто гнучкою і може уявити просту залежність складним чином. Це явище називається перенавчанням. У разі мережі з недостатньою кількістю нейронів, навпаки, необхідний рівень помилки ніколи не буде досягнутий. Тут у наявності надмірне узагальнення.

Для попередження перенавчання застосовується наступна техніка. Дані поділяються на два множині: **навчальне (Training Data)** та **контрольне (Validation Data)**. Контрольна множина в навчанні не використовується. На початку роботи помилки мережі на навчальному та контрольному множинах будуть однаковими. У міру того, як мережа навчається, помилка навчання убуває, і, поки навчання зменшує дійсну функцію помилки, помилка на контрольній множині також буде спадати. Якщо ж контрольна помилка перестала зменшуватися або навіть стала рости, це вказує на те, що навчання слід закінчити. Зупинка на цьому етапі називається ранньої зупинкою (Early stopping).

Таким чином, необхідно провести серію експериментів з різними мережами, перш ніж буде отримана відповідна. При цьому щоб не бути введеним в оману локальними мінімумами функції помилки, слід кілька разів навчати кожну мережу.

Якщо в результаті послідовних кроків навчання та контролю помилка залишається неприпустимо великий, доцільно змінити модель нейронної мережі (наприклад, ускладнити мережу, збільшивши число нейронів, або використовувати мережу іншого виду). У такій ситуації рекомендується застосовувати ще одна множина - тестова **множина спостережень (Test Data)**, яке представляє собою незалежну вибірку з вхідних даних. Підсумкова модель тестується на цій множині, що дає додаткову можливість переконатися в достовірності отриманих результатів. Очевидно, щоб зіграти свою роль, тестова множина повинна бути використана тільки один раз. Якщо його використовувати для коригування мережі, воно фактично перетвориться на контрольну множину.

Побудова нейронних мереж в середовищі Matlab із застосуванням пакетів NNTool.

Нейромережевий інструментарій середовища MATLAB (Neural Net Toolbox) - універсальна нейронна мережеве середовище. У пакет включено практично всі, що стосується процедур створення нейронних мереж і аналізу отриманих результатів, в частотності, їх інтерпретації.

Всі команди MATLAB приймають наявність мережевого об'єкту, який

називається "мережа" (net). Для того щоб створити такий об'єкт, необхідно виконати команду

net = network;

яка дає чисту мережу, тобто без "особливостей" (характеристик), які визначають специфіку завдання. В якості таких особливостей в першу чергу виступає конфігурація мережі.

Мережа в MATLAB складається з шарів, які діляться на вхідні і приховані. Вхідні шари служать для подачі на вхід мережі вхідних векторів і мають лише одну характеристику розмір, відповідний розмірності вхідних векторів. Приховані шари мають розмір, що відповідає кількості нейронів, і функцію активації. Кожен прихований шар може бути з'єднаний з будь-яким з вхідних шарів, а також з будь-яким іншим прихованим шаром. Крім того кожен прихований шар може бути вихідним або цільовим. Вихідні шари визначають вихід мережі, а виходи цільових верств використовуються при навчанні мережі з учителем. Таким чином, MATLAB дозволяє створювати мережі з досить складною конфігурацією, однак на практиці як правило використовуються багатошарові мережі, в яких кожен прихований шар з'єднаний з одним наступним за ним шаром, перший прихований шар з'єднаний з єдиним вхідним шаром, а останній прихований шар призначається і вихідним , і цільовим.

Для створення нейронної мережі із заданою конфігурацією можна керувати за допомогою розширених синтаксис команди *network*:

net = network(numInputs, numLayers, biasCoHCect, inputCoHCect, layerCoHCect, outputCoHCect, targetCoHCect),

де

- *numInputs* – кількість вхідних шарів.
- *numLayers* – кількість прихованих шарів.
- *biasCoHCect* – вектор *numLayers x 1*, що складається з нулів та одиниць. Якщо і-а компонента даного вектора дорівнює 1, то *i*-ий прихований шар має зсув.
- *inputCoHCect* - матриця *numLayers x numInputs*, також складається з нулів і одиниць. Якщо *inputCoHCect(i, j) = 1*, то *i*- ий прихований шар з'єднаний з *j*-им вхідним шаром.
- *layerCoHCect* – матриця *numLayers x numLayers*, що визначає з'єднання між прихованими шарами. Якщо *layerCoHCect(i, j) = 1*, то є зв'язок з *j*-ого прихованого шару в *i*-ий прихований шар.
- *outputCoHCect* – вектор *1 x numLayers*, що визначає, які приховані шари є вихідними. Якщо *outputCoHCect(i) = 1*, то *i*- ий прихований шар є вихідним.
- *targetCoHCect* - вектор *1 x numLayers*, що визначає, які приховані шари є цільовими. Якщо *targetCoHCect(i) = 1*, то *i*- ий прихований шар є цільовим.

Створений мережевий об'єкт має ряд властивостей. По-перше, це властивості *net.numInputs*, *net.numInputs*, *net.numLayers*, *net.biasCoHCect*, *net.inputCoHCect*, *net.layerCoHCect*, *net.outputCoHCect*, *net.targetCoHCect*, аналогічні відповідним параметрам створення мережевого об'єкту.

По-друге, це властивості, що визначають вагові коефіцієнти:

- *net.IW{i,j}* – матриця вагових коефіцієнтів для зв'язку *j*-го вхідного шару з *i*-им прихованим шаром.
- *net.LW{i,j}* - матриця вагових коефіцієнтів для зв'язку *j*-го прихованого шару з *i*-им прихованим шаром.
- *net.b{i}* – величини зсувів для *i*-го прихованого шару.

По-третє, це властивості, що містять под'об'єкти, що визначають деякі додаткові параметри:

- *net.inputs{i}* – структура, яка містить параметри *i*-го вхідного шару.

Для кожного вхідного шару можна задати розмірність векторів, що надходять на його вхід, через властивість *net.inputs{i}.size*, а також вхідний діапазон через властивість *net.inputs{i}.range*. Для завдання вхідного діапазону використовується матриця *net.inputs{i}.size x 2*. В *k*-му рядку цієї матриці вказується мінімальне і максимальне значення, які може приймати *k*-а компонента вхідного вектора.

- *net.layers{i}* – структура, яка містить параметри *i*-го прихованого шару.

Для кожного прихованого шару можна задати кількість нейронів в ньому через властивість *net.layers{i}.size*, а також функцію активації через властивість *net.layers{i}.transferFcn*, в якості якої можна вказати одну з наступних функцій:

- *logsig* – униполярная сигмоїда: *logsig(x) = 1 / (1 + exp(-x))*.
- *tansig* – біполярна сигмоїда: *tansig(x) = 2/(1+exp(-2*x))-1*.
- *purelin* – лінійна функція: *purelin(x) = x*.
- *satlin* – асиметрична лінійна функція з насиченням:

$$satlin(x)=\begin{cases}0, & x\le 0\\ x, & 0\le x\le 1\\ 1, & 1\le x\end{cases}$$

- *satlins* – симетрична лінійна функція з насиченням:

$$satlins(x)=\begin{cases}-1, & x\le -1\\ x, & -1\le x\le 1\\ 1, & 1\le x\end{cases}$$

- *hardlim* – асиметричний стрибок:

$$hardlim(x)=\begin{cases}1, & x\ge 0\\ 0, & x\le 0\end{cases}$$

- *hardlims* – симетричний стрибок:

$$hardlims(x)=\begin{cases}1, & x\ge 0\\ -1, & x\le 0\end{cases}$$

- *net.biases{i}* – структура, яка містить параметри зміщень *i*-го прихованого шару.
- *net.inputWeights{i,j}* – структура, яка містить параметри зв'язку з j-го вхідного шару в *i*-ий прихований шар. Найбільш важливим

параметром тут є вагова функція *net.inputWeights{i,j}. weightFcn*, визначальна яким чином використовується матриця ваг при обчисленні виходів нейронів. Як ваговій функції може виступати:

- *dotprod* –множення.
- *normprod* – нормалізоване множення
- *dist* – евклідова відстань.
- *negdist* – евклідова відстань, взяте з протилежним знаком.
- *mandist* – Манхетеновська відстань

- *net.layerWeights{i,j}* – структура, яка містить параметри зв'язку з j-го прихованого шару в і-ий прихований шар. За допомогою даної структури також є можливість задати вагову функцію за допомогою властивості *net.layerWeights{i,j}.weightFcn*.
- *net.outputs{i}* – структура, яка містить параметри і-го вихідного шару.
- *net.targets{i}* – структура, яка містить параметри і-го цільового шару.

Для перегляду значень простих властивостей або складу складних властивостей досить ввести їх в командний рядок MATLAB і натиснути ENTER. Крім того в MATLAB є такий засіб як Simulink, здатне побудувати візуальне уявлення нейронної мережі.

Для того щоб скористатися цією можливістю досить виконати команду

gensim(net);

Нижче наведено код, який створює мережу з одним вхідним шаром і двома прихованими шарами:

net = network; net.numInputs = 1;

net.numLayers = 2;

Те ж саме можна зробити наступним чином:

net = network(1,2);

Нижче наведено код, який створює мережу, в якій вхідний шар з'єднаний з першим прихованим шаром, перший прихований шар з'єднаний з другим прихованим шаром, який є вихідним і цільовим:

net = network(1,2,[1;0],[1; 0],[0 0; 1 0],[0 1],[0 1]);

При етов в створеній мережі перший прихований шар має зсув, а другий прихований шар - немає.

Ініціалізація

В MATLAB є дві можливості для навчання нейронних мереж: тренування і адаптація. При тренуванні ваги мережі оновлюються після пред'явлення їй всієї множини вхідних векторів. Після цього отримані значення помилок усереднюються відповідно до обраної функцією виконання - net.performFcn. Як її значення можна вказати одну з наступних стандартних функцій:

• *mse* – середньоквадратична помилка;

• *mae* – середньоабсолютна помилка.

Далі ваги мережі оновлюються відповідно до обраної тренувальної функцією- net.trainFcn.

У середовищі MATLAB є великий вибір таких функцій:

• *traingd* – навчання за алгоритмом найшвидшого спуску;

• *traingdm* – навчання за алгоритмом найшвидшого спуску з урахуванням

моменту;

- *trainlm* - навчання за алгоритмом Левенберга-Марквардта;
- *trainrp* – навчання за алгоритмом RPROP і ін.

Один такий цикл навчання називається Епоха. Розширені возможности поиска навчання встановлюються через структуру net.trainParam. Склад цієї Структури Залежить від обраної тренувальної Функції. Спільнімі и найбільш часто вікорістовуванімі є следующие Параметри:

- *net.trainParam.lr* – коефіцієнт навчання;
- *net.trainParam.goal* – максимально допустиме значення помилки (мета навчання);
- *net.trainParam.epochs* – максимальну кількість епох навчання;
- *net.trainParam.show* – визначає проміжок часу в епохах між повідомленнями про статус процесу навчання.

Сам процес тренування викликається за допомогою функції *train*:

```
P = [0.3 0.2 0.54 0.6;
1.2 2.0 1.4 1.5];
T = [0 1 1 0];
net = train(net, P, T);
```

У цьому прикладі *P* - матриця, що представляє велику кількість вхідних даних, кожен стовпець якої інтерпретується як один з вхідних векторів. А *T* - це матриця цільових значень, кожен стовпець якої інтерпретується як цільової вектор. При цьому кількість стовпців в матріцач *P* і *T* має збігатися.

У разі навчання без учителя матриця *T* не вказується, тому що навчання відбувається лише на основі вхідних векторів.

```
net = train(net, P);
```

Адаптація викликається за допомогою функції adapt. При адаптації є можливість контролювати порції вхідних векторів, після подачі яких буде відбуватися оновлення ваг:

```
P = {[0.3; 1.2] [0.2; 2.0] [0.54; 1.4] [0.6; 1.5]}
T = {[0] [1] [1] [0]};
net = adapt(net, P, T);
```

У наведеному прикладі оновлення ваг буде відбуватися після подачі кожного з вхідних векторів. Адаптація ваг відбувається відповідно до методу адаптації - net.adaptFcn. На даний момент передбачений лише один метод адаптації - adaptwb, відповідно до якого використовуються функції навчання, що встановлюються для всіх ваг та зсувів окремо через властивості *net.inputWeights{i,j}.learnFcn*, *net.layerWeights{i,j}.learnFcn* та *net.biases{i}.learnFcn*. Як навчальної функції можна вказати

- *learnp* – навчання за правилом персептрона;
- *learngd* – за алгоритмом найшвидшого спуску;
- *learngdm* – за алгоритмом найшвидшого спуску з урахуванням моменту інерції;
- *learnh* – за правилом Хебба;
- *learnlv1* – методом квантування векторів;
- *learnis* – за правилом навчання інстара;
- *learnos* – за правилом навчання аутстара і ін.

Кількість проходів по всій множині навчальних даних встановлюється у властивості *net.adaptParam.passes*.

Після навчання мережі можна подати на її вхід множину вхідних векторів і отримати множину вихідних векторів за допомогою функції sim:

Y = sim(net, P);

У цьому випадку, як і при навчанні, кожен стовпець матриці P буде відповідати одному вхідному вектору, а кожен стовпець матриці Y - одному вихідному вектору.

Таким чином загальна схема роботи з нейронними мережами в середовищі MATLAB зводиться до наступної схеми

1. Створення нейросетевого об'єкта і його ініціалізація.
2. Навчання нейронної мережі.
3. Отримання значень виходів нейронної мережі і їх інтерпретація.

Слід зазначити, що в середовищі MATLAB є можливість використовувати для створення "типового" нейромережового об'єкту одну зі стандартних функцій. Далі будуть розглянуті деякі найбільш часто використовуваних з них.

Мережа прямого поширення (newff).

Така мережа є звичайною багатошаровою мережею з одним вхідним шаром та декількома прихованими шарами, послідовно з'єднаними один з одним. Останній прихований шар призначається вихідним і цільовим. Для створення даної мережі використовується функція *newff*:

net = newff(PR,[S1 S2...SNl],{TF1 TF2...TFNl},BTF,BLF,PF);

де *PR* - матриця R x 2 мінімальних і максимальних векторів для *R* вхідних елементів; *Si* - размір *i*-ого прихованого шару; *TFi* – функція активації i-ого прихованого шару; *BTF* - тренувальна функція; *BLF* - навчальна функція, яка використовується при адаптації; *PF* - функція виконання. Тільки перші два параметри є обов'язковими.

Нижче наведено приклад використання даної мережі для апроксимації модельної функції MATLAB *humps*:

```
x = -1:0.1:3; y = humps(x);
net = newff(minmax(x), [5 1], ...
{'tansig' 'purelin'}, ...
'trainlm', 'learngdm', 'mse'); net.trainParam.epochs = 100;
net = train(net, x, y); a = sim(net, x); plot(x, a, x, a-y);
```

Нижче наведено приклад використання даної мережі для апроксимації поверхні *z = cos(x)*sin(y)*:

```
x = -2:0.1:2; y = -2:0.1:2; z = cos(x)'*sin(y); P = [x; y]; T = z;
net = newff(minmax(P), [25 length(x)], ...
{'tansig' 'purelin'}, ...
'trainrp', 'learngdm', 'mse'); net.trainParam.epochs = 1000;
net = train(net, P, T); Y = sim(net, P);
mesh(x, y, Y);
```

Радіальна мережа (newrb).

Така мережа складається з двох шарів. Перший шар складається з нейронів, що обчислюють радіальну функцію *RADBAS* як функцію від відстані *DIST* між позицією нейрона та вхідним вектором. Позиції нейронів зберігаються в рядках матриці ваг *net.IW {1,1}*. Другий шар складається зі звичайних нейронів з лінійною функцією активації *PURELIN*. При створенні даної мережі підбирається кількість нейронів в першому шарі. Спочатку береться один нейрон і обчислюються ваги другого шару. Якщо вийшла помилка більше допустимої, то додаються нейрони в перший шар на позиції тих вхідних векторів, для яких помилка досягає свого максимального значення, та процес повторюється.

Для створення даної мережі використовується функція *newrb*:

net = newrb(P,T,GOAL,SPREAD,MN,DF)

де *P* – матриця *R x Q* з *Q* вхідних векторів розмірністю R, *T* - матриця S x Q з Q цільових векторів розмірністю S, *GOAL* - максимально допустима помилка, *SPREAD* - параметр сглаженности радіальних функцій, *MN* - максимально допустима кількість нейронів в першому шарі, *DF* - кількість нейронів, що додаються на кожному кроці. Лише перші два параметри є обов'язковими.

Нижче наведено приклад використання даної мережі для апроксимації поверхні *z = cos(x*sin(y))*:

x = -2:0.1:2; y = -2:0.1:2; z = cos(x'*sin(y)); P = [x; y]; T = z;

net = newrb(P, T); Y = sim(net, P);

mesh(x, y, Y);

Імовірнісна нейронна мережа (newpHC).

Така мережа є однією з різновидів радіальних нейронних мереж, що будуються на основі теореми Ковера. Вона використовується для класифікації, причому допускає тільки навчання з учителем. Кількість нейронів у першому шарі даної мережі встановлюється рівною кількості вхідних векторів. Ваги першого шару встановлюються рівними транспонованою матриці вхідних даних, а ваги другого шару встановлюються такими, щоб забезпечити бажаний вихід мережі.

Для створення даної нейронної мережі використовується функція *newpHC*:

net = newpHC(P,T,SPREAD)

де *P* – матриця, стовпці якої складають множину вхідних векторів, T - матриця, стовпці якої складають бажані вихідні вектора, SPREAD - параметр сглаженности радіальних базисних функцій (як такої тут використовується функція MATLAB RADBAS).

Нижче наведено приклад використання даної нейронної мережі:

% Вхідні значення

P = [1 2 3 4 5 6 7];

% відповідні класи

Tc = [1 2 3 2 2 3 1];

% Перетворення номерів класів в вектора

T = ind2vec(Tc)

```
% Створення нейронної мережі і її навчання
net = newpHC(P,T);
% Перевірка роботи нейронної мережі
Y = sim(net,P)
% Перетворення отриманих векторних даних в номери класів
Yc = vec2ind(Y)
```

Мережа змагального шару (newc).

Така мережа складається з одного шару нейронів, яких навчають за правилом Кохонена зі стратегією WTA (WiHCer Takes All). Позиції нейронів зберігаються в рядках матриці net.IW {1,1}. На виході такої мережі буде набір векторів, що складаються з нулів і одиниць. У кожному з них присутній лише одна одиниця, що показує, який з нейронів переміг.

Для створення даної мережі використовується функція *newc*:

```
net = newc(PR,S,KLR,CLR)
```

де *PR* - матриця R x 2 мінімальних та максимальних значень для R компонентів вхідних векторів; S - кількість нейронів, KLR - коефіцієнт навчання за правилом Кохонена, CLR - коефіцієнт "совісті", який використовується для підвищення шансів на перемогу тих нейронів, які перемагають рідше інших, таким чином, щоб всі нейрони перемагали з приблизно однаковою частотою. Лише перші два параметри є обов'язковими.

Розглянемо приклад навчання такої мережі на деякій множині точок:

```
n = 100;
P = [[ 9+5*rand(1,n); 9+5*rand(1,n)]
[19+5*rand(1,n); 19+5*rand(1,n)]]
plot(P(1,:), P(2,:), '.')

net = newc(minmax(P), 2);
net = train(net,P);

Y = sim(net,P); Yc = vec2ind(Y);

figure; hold;
for i=1:length(P) if Yc(i) == 1
c = '.r';
else
c = '.b';
end
plot(P(1,i), P(2,i), c); end
```

У наведеному прикладі було створено масив точок, що складається з двох кластерів. На цих точках була навчена мережа змагального шару з двома нейронами. Далі за допомогою даної мережі множину точок було розділено на два класи, а отримані результати були виведені на графіку: точки, ближчі до першого нейрона, були виведені червоним кольором, а точки, ближчі до другого нейрона - синім кольором. Самі позиції нейронів можна вивести за допомогою команди

```
plotsom(net.IW{1,1}');
```

Самоорганізована карта (newsom).

Така мережа є подальшим розвитком мережі змагального шару. Вона також складається з одного шару нейронів, але в ній на кожному кроці навчання піддається не тільки переміг нейрон, а й кілька найближчих до нього нейронів. Процес навчання складається з двох етапів: фази упорядкування і фази настройки. На першому етапі нейрони повинні приблизно рівномірно розподілитися по всій множині вхідних точок. Для цього на першому кроці навчання піддаються всі нейрони, в процесі навчання кількість учнів нейронів зменшується, на останньому кроці навчання піддається лише один переміг нейрон. На другому етапі нейрони повинні визначити всі кластери, складові навчальної множини. На цьому етапі навчання піддається переміг нейрон і нейрони, що знаходяться до нього ближче заданої відстані. Крім того є можливість управляти початковим розташуванням нейронів в просторі.

Для створення даної мережі використовується функція *newsom*:

```
net = newsom(PR,[D1,D2,...,DNl],TFCN,DFCN,OLR,OSTEPS,TLR,TND)
```

де *PR* - матриця R x 2 мінімальних і максимальних значень для R компонентів вхідних векторів; нейрони мережі будуть рівномірно розподілені в Nl-вимірному просторі по Di шарів в кожному вимірі (наприклад, [5 8] задає розташування 40 нейронів в двовимірному просторі в вузлах решітки 5 x 8); TFCN - функція топології, яка визначає, яким саме чином будуть розташовані нейрони (gridtop - в вузлах прямокутної решітки, hextop - в вузлах решітки, що складається з шестикутників); DFCN - використовувана функція відстані (dist - евклідова відстань, mandist - манхетеновська відстань); OLR - коефіцієнт навчання в фазі упорядкування, OSTEPS - кількість кроків, що відводяться по фазу упорядкування; TLR - коефіцієнт навчання в фазі настройки (зазвичай TLR << OLR); TND - відстань від переміг нейрона, в межах якого піддаються навчанню інші нейрони під час другої фази.

Нижче наведено приклад використання даної мережі:

```
P = [];
for i = 1:100
P = [P [9+5*mod(i,4)+5*rand; 9+5*mod(i,4)+5*rand]]; end
plot(P(1,:), P(2,:), '.'); hold;

net = newsom(minmax(P), [2 2], 'hextop', 'dist', 1, 100, 0.1, 0); plotsom(net.layers{1}.positions);

net.trainParam.epochs = 500; net = train(net,P);

Y = sim(net,P); Yc = vec2ind(Y);

figure; hold; colors = ['b' 'g' 'r' 'c']; for i=1:length(P)
plot(P(1,i), P(2,i), ['.' colors(Yc(i))]); end
plotsom(net.iw{1,1},net.layers{1}.distances);
```

Мережа квантування векторів (newlvq).

Дана мережа також є подальшим розвитком мережі змагального шару. Вона використовується для поділу вхідних векторів на класи, але на відміну від змагального шару може навчатися только з учителем. Основною перевагою даної мережі є те, що одному класу може відповідати не один, а декілька нейронів. Вона складається з двох шарів: перший шар є звичайним змагальним шаром, а другий складається зі звичайних лінійніх нейронів та використовується для об'єднання сігналів від всіх нейронів, що відповідають одному класу. Вихід цієї мережі, як і вихід змагального шару, складається з нулів та однієї одиниці, яка б показала, до якого класу належить вхідній вектор.

Для створення даної мережі використовується функція *newlvq*:

net = newlvq(PR,S1,PC,LR,LF)

де *PR* - матриця *R x 2* мінімальних та максимальних значень для R компонентів вхідних векторів; *S1* – кількість нейронів в першому шарі; PC - вектор [p1 p2 ... pn] процентних співвідношень, що показують, яка частина нейронів першого шару відповідає даного класу (сума всіх pi повинна бути дорівнює 1); LR - коефіцієнт навчання, LF - навчальна функція (повинна дорівнювати learnlv1, ще один алгоритм - learnlv2 - застосовується для "тонкої" настройки мереж, вже навчених за алгоритмом learnlv1).

Нижче наведено приклад застосування даної мережі для поділу двох класів точок, які не можна розділити за допомогою мережі змагального шару:

```
P = []; Tc = []; colors = ['r' 'b']; figure; hold;
for i = 1:100
P = [P [9+5*mod(i,4)+5*rand; 9+5*mod(i,4)+5*rand]]; Tc = [Tc 2-mod(i,2)];
plot(P(1,i), P(2,i), ['.' colors(Tc(i))]); end
T = ind2vec(Tc);
net = newlvq(minmax(P), 4, [0.5 0.5]);
plotsom(net.iw{1,1}');

net.trainParam.epochs = 500; net = train(net,P,T);

Y = sim(net,P); Yc = vec2ind(Y);

figure; hold;
for i=1:length(P)
plot(P(1,i), P(2,i), ['.' colors(Yc(i))]); end
plotsom(net.iw{1,1}');
```

Мережа Хопфілда (newhop).

Мережа Хопфілда використовується для створення автоасоціативної пам'яті. За її допомогою можна запам'ятати набір векторів, що складаються з +1 та -1. При подачі на її вхід деякого вектора на виході мережі буде отримано один з її векторів, який мережа визначила найбільш схожим на вхідний. В MATLAB реалізована модифікована мережа Хопфілда, що володіє

меншою ємністю, але краще запам'ятовує вектора. Крім того, при відновленні векторів вона допускає подачу на вхід векторів з довільних значень (а не тільки +1 та -1).

Для створення даної мережі використовується функція *newhop*:

net = newhop(T)

де *T* – матриця, кожен стовпець якої інтерпретується як запам'ятовується вектор.

Нижче наведено приклад використання даної мережі для запам'ятовування двох точок в тривимірному просторі:

```
T = [-1 -1 1; 1 -1 1]';
net = newhop(T);
```

Далі перевіримо роботу даної мережі:

```
Y = sim(net, 2, [], T);
```

Зверніть увагу на незвичайний синтаксис команди sim. Це пов'язано з тим, що у мережі Хопфілда фактично немає вхідного шару. Замість цього задається початковий стан єдиного прихованого шару нейронів. У цьому випадку вказується кількість векторів і матриця, складена з них.

Перевіримо роботу мережі на перекручених векторах:

```
T = [0.7 0.6 -0.25]';
Y = sim(net, 1, [], T);
```

У цьому випадку на виході мережі буде отримано вектор, відмінний від збережених. Це пов'язано з тим, що для відновлення значень векторів мережі Хопфілда потрібно кілька циклів роботи, а в даному випадку проведено лише один цикл. Для того щоб провести кілька циклів (в даному випадку 5) використовується наступний синтаксис:

```
T = {[0.7; 0.6; -0.25]};
Y = sim(net, {1 5}, {}, T);
```

Після цього Y {i} буде містити вихідний вектор мережі Хопфілда після *i*-ого циклу роботи.

Лінійний шар (newlin).

Лінійний шар складається з одного шару звичайних нейронів з лінійною функцією активації.

Для створення даної мережі використовується функція newlin:

net = newlin (PR, S, ID, LR)

де PR - матриця R x 2 мінімальних та максимальних значень для R компонентів вхідних векторів; S - кількість нейронів; ID - вектор зростаючих невід'ємних тимчасових затримок, що задає від яких вхідних векторів надходить сигнал вхід лінійного шару (наприклад, [0] - сигнал надходить від вхідного вектора, [0 1] - сигнал надходить від вхідного вектора і від попереднього вхідного вектора, [0 3 5] - сигнал надходить від вхідних векторів поточного і надходили 3 і 5 кроків назад; в поточній реалізації наявність 0 в першій позиції є обов'язковим); LR - коефіцієнт навчання. Лише перші два параметри є обов'язковими.

Нижче наведено приклад мережі для відновлення синусоїдального

сигналу тривалістю 5 секунд, вимірювання якого виконані за вибіркою по нормі 40 зразків в секунду, по 5 попереднім значенням:

```
time = 0:0.025:5; y = sin(time*4*pi); Q = length(y); P = {}; T = {}; P1 = [];
for i = 1:Q-5
P = [P {[y(i) y(i+1) y(i+2) y(i+3) y(i+4)]'}]; P1 = [P1 P{i}];
T = [T {y(i+5)}];
end;

net = newlin([-ones(5,1) ones(5,1)], 1); net.inputWeights{1.1}.learnFcn = 'learngdm'; net.adaptParam.passes = 10;
net = adapt(net, P, T); a = sim(net, P1); plot(a);
```

Зверніть увагу, що тут використовувалася функція adapt, а не train, як в інших прикладах. Це пов'язано з тим, що в разі використання тренування помилка буде спочатку усереднюватися по всім вхідним значенням, а потім використовуватися для навчання, що не принесе користь в силу періодичності значень синусоїди (позитивні значення будуть компенсувати негативні).

Той же приклад з використанням тимчасових затримок буде виглядати наступним чином:

```
time = 0:0.025:5; y = sin(time*4*pi); Q = length(y); T1 = {}; T2 = {};
for i = 1:Q-1
T1 = [T1 {y(i)}];
T2 = [T2 {y(i+1)}];
end;

net = newlin([-1 1], 1, [0 1 2 3 4]); net.inputWeights{1.1}.learnFcn = 'learngdm'; net.adaptParam.passes = 10;
net = adapt(net, T1, T2); a = sim(net, y);
```

Заваження

Для роботи з зображеннями застосовуються такі функції:

- **A = imread(filename);**

В результаті в матрицю A буде занесено значення кольорів пікселів.

Розміри завантаженого зображення можна дізнатися наступним чином:

[height width] = size(A)**imshow(A);** В В результаті в матрицю A буде занесено значення результаті з'явиться вікно із зображенням, що зберігається в матриці A.

- **imwrite(A, filename);**

В результаті зображення, яке зберігається в матриці A, буде збережено в файлі з вказаним ім'ям.

Варианти завдань:

Завдання 1.

Реалізувати нейронну мережу прямого поширення з одним нейроном в вхідному шарі, N нейронами у прихованому шарі з сигмоидальной функцією активації (logsig) та одним нейроном у вихідному шарі з лінійною функцією активації. Навчити її апроксимації заданої функції за алгоритмом зворотного

поширення. Виконати порівняння якості апроксимації для поліноміальної, синусоїдальної та humps функцій. Для однієї з функцій виконати порівняння якості апроксимації для різних N (варіант – номер в списку групи).

1. Алгоритм найшвидшого спуску
2. Алгоритм градієнтного спуску з рестарту Пауелла-Біля
3. Алгоритм градієнтного спуску з оновленням по Флетчеру-Рівсу
4. Алгоритм градієнтного спуску з оновленням по Полак-Рібьер
5. Адаптивний алгоритм градієнтного спуску
6. Алгоритм градієнтного спуску з моментом інерції
7. Алгоритм градієнтного спуску з однокрокової січною
8. Алгоритм змінної метрики з обчисленням гессіан за формулою Бройдена- Флетчера-Гольдфарба-шенно (BFGS)
9. Алгоритм змінної метрики з обчисленням гессіан за формулою Девідона- Флетчера-Пауелла (DFP)
10. Алгоритм Левенберга-Марквардта
11. Алгоритм Quickprop
12. Модифікований алгоритм Quickprop
13. Алгоритм RPROP

Завдання 2:

Розв’язати завдання згідно з Вашим варіантом в середовищі Matlab із застосуванням пакету Neural Network Toolbox (варіант – номер в списку групи).

1.Застосувати радіальну мережу (Radial basis, newrb) для апроксимації поверхні peaks (рис. 1) при різних кількостях шарів (2-4) і нейронів в шарах (1-30). У звіті привести поверхню залежності максимальної помилки апроксимації від параметрів мережі та усереднену поверхню помилки апроксимації по всіх проведених експериментів (поверхню помилки для кожного експерименту поділити на величину максимальної помилки, скласти поверхні для всіх експериментів і поділити на їх кількість). Яка архітектура мережі буде оптимальною?

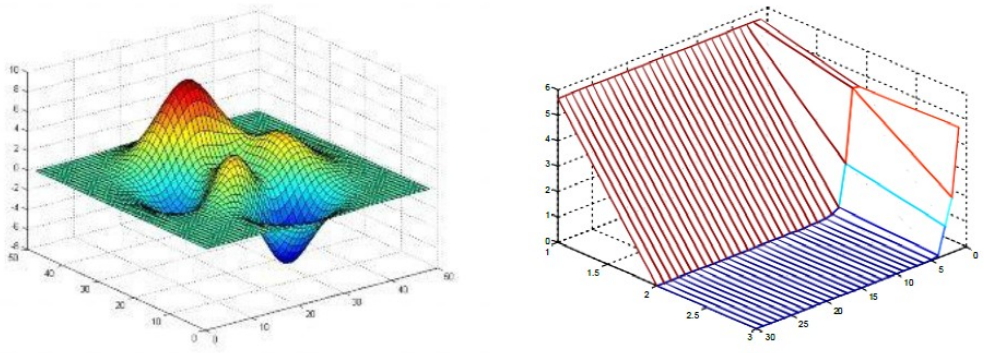

Рис. 1. Поверхня функції двох аргументів peaks (зліва) і приклад поверхні помилки апроксимації в залежності від параметрів мережі (праворуч).

2.Сформувати два класи точок за формулами

```
t = 0:0.01:(2.25*pi); n = length(t);
delta = 70/length(t);
r = 30:delta:(100-delta);
r = exp(r/20)+20;
d = 10;
t1 = t+d1; t2 = t+d2;
x1 = 100+r.*cos(t1)+rand(1,n)*d;
y1 = 100+r.*sin(t1)+rand(1,n)*d;
x2 = 110+r.*cos(t2)+rand(1,n)*d;
y2 = 90+r.*sin(t2)+rand(1,n)*d;
plot(x1, y1, '.r'); hold; plot(x2, y2, '.b');
```

Приклад цих класів для **d1 = -pi/4** и **d2 = 3*pi/4** наведено на рис. 2.

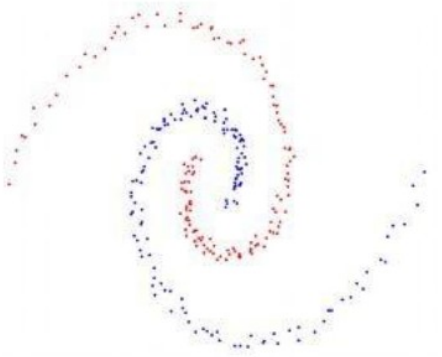

Рис. 2. Приклад взаємного розташування двох класів точок - синього та червоного.

Побудувати мережу квантування векторів (Learning vector quantization, newlvq) для поділу цих класів. Скільки нейронів необхідно для цього і як вони будуть розташовуватися? Показати траєкторії нейронів, прохідні ними в процесі навчання.

3. Застосувати мережу квантування векторів (Learning Vector Quantization, newlvq) для розпізнавання зображень цифр від 1 до 3 (16 градацій сірого, 16x16, див. Рис. 3). Застосуйте ймовірностну нейронну мережу для розв'язання тієї ж проблеми (Probabilistic neural networks, newpHC). Порівняйте якість роботи обох мереж на зашумлених зразках (для накладення шуму додайте до кожного пікселя випадкове число).

Рис. 3. Зображення цифр для розпізнавання.

4. Побудувати мережу лінійного шару (Linear layer, newlin) для відновлення послідовності $f(n) = n \times 2^n + 3^n$ за вибіркою з 3-х попередніх значень без використання тимчасових затримок. Порівняйте коефіцієнти рекурентной формули, отримані під час навчання мережі, з їх точними значеннями. Побудуйте мережу лінійного шару з використанням тимчасових затримок (3-й аргумент у функції newlin) для відновлення тієї ж

послідовності.

5. Побудувати множину точек

```
t = 0:0.05:(6*pi);
n  =length(t);
P = zeros(2,n);
r = exp(2:(3/n):(5-3/n))/exp(5)*50;
P(1,:) = r.*cos(t)+rand(1,n);
P(2,:) = r.*sin(t)+rand(1,n);
```

Отримана множина повинна виглядати як на рис. 4.

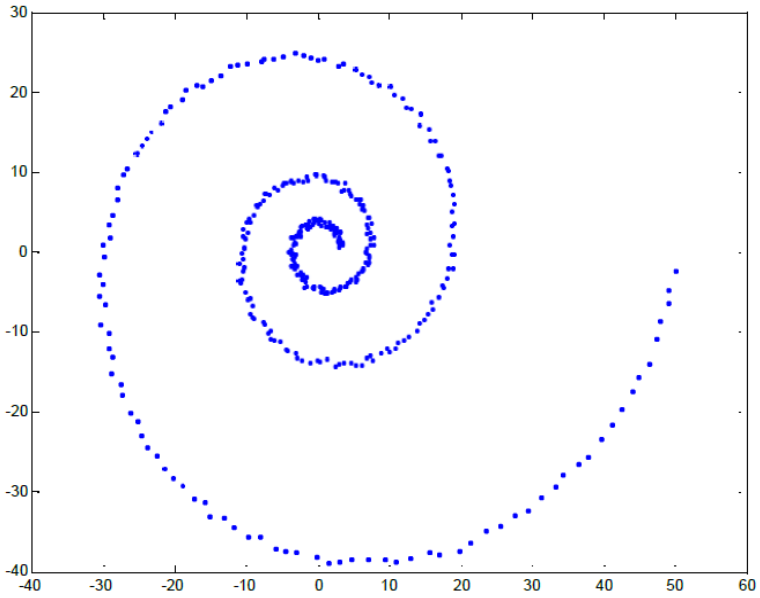

Рис. 4. Множина точок для навчання мережі за правилом Кохонена.

Побудувати змагальний шар (Competitive layer, newc) з 25 нейронів та навчити його на даній множині. Показати траєкторії руху нейронів при навчанні і як будуть розташовуватися нейрони після його закінчення. Навчити самоорганізуючу карту (Self-organizing map, newsom) (к-ть нейронів той же) на тій же множині. Порівняти розташування нейронів в першому та другому випадку. Пояснити отримані результати.

6. Порівняти застосування мережі прямого поширення (Feed-forward, newff) та мережі Хопфілда (Hopfield, newhop) для розпізнавання зображень букв (чорно-білі зображення, 20x20, див. рис. 5). Оцініть якість роботи мереж на зашумлених зразках (для накладення шуму випадковим чином інвертуйте кілька пікселів).

Рис. 5. Зображення букв для розпізнавання.

7. Застосування самоорганізуючої карти (Self-organizing map (SOM), newsom) для стиснення зображення. Як зображення взяти bmp-файл (256

градацій сірого) і розбити його на кадри nxn. Перетворити послідовність кадрів зображення в послідовність n^2- елементних векторів і використовувати для навчання мережі SOM (бажано взяти 256 нейронів у мережі). Процес стиснення полягає в перетворенні послідовності кадрів зображення в послідовність чисел (номер класу, до якого був віднесений даний кадр). Відновлення зображення полягає в заміні кожного числа вектором, що представляє "центр ваги" даного класу (він зберігається у вхідних вагах мережі). Приклад роботи даного механізму наведено на рис. 6. Привести приклад роботи побудованої мережі для іншого зображення. Оцінити ступінь стиснення даних за допомогою такого механізму.

Рис. 6. Приклад застосування мережі SOM для стиснення зображення (вихідне зображення зліва, а відновлене - праворуч).

8. Стиснення зображення за допомогою виділення головних компонент. Для виділення компонент застосувати двошарову мережу прямого поширення (Feed-forward, newff), в якій розмір першого шару збігається з кількістю компонент K, а другого (вихідного) шару - з вхідним шаром. Розбити зображення на кадри 10x10, в якості вхідних векторів мережі можна використовувати або рядки, або стовпці матриці пікселей одного кадру. Процес стиснення полягає в отриманні для кожного кадру матриць розкладання і реконструкції (зберігаються в вагах мережі вхідного і другого шарів відповідно) і подальшої заміни кожного стовпця (рядка) $\boldsymbol{x}$ вектором $\boldsymbol{y}$ з **K** головних компонент. Таким чином кожен кадр буде замінений на матрицю рекострукції та послідовність векторів **y**. Приклад роботи даного механізму наведено на рис. 7. Оцінити ступінь стиснення даних за допомогою такого механізму.

Рис. 7. Приклад стиснення зображення за допомогою виділення

головних компонент (зліва-направо: вихідне зображення, відновлене при використанні 1-ої головної компоненти, при використанні 4-х головних компонент).

9. Застосування мережу прямого поширення (Feed-forward, newff) для розпізнавання характерного ділянки на графіку реєстрації деякого процесу за допомогою переміщуваного тимчасового вікна (рис. 8, розпізнається момент відзначений пунктирною лінією). Як буде вести себе мережу при аналізі спотвореного сигналу (зміна постійної складової сигналу (зрушення вгору-вниз), його стиснення - розтягнення по вертикалі, накладення шуму)? Додаткове завдання - спробувати застосувати мережу Ельмана (newelm) для розв'язання того ж завдання.

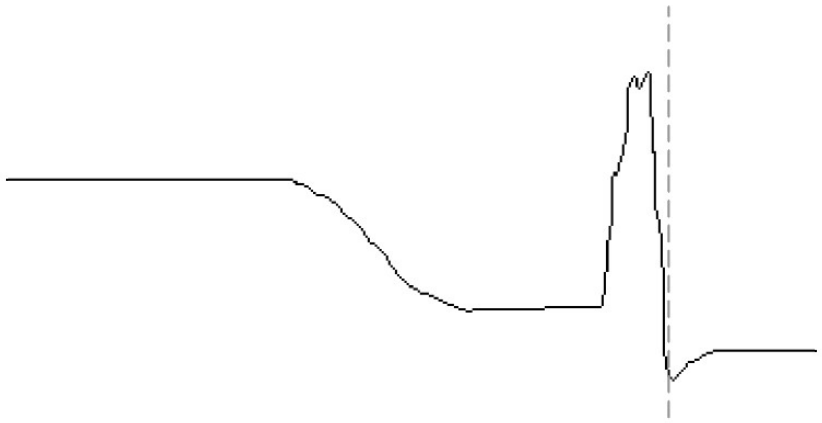

Рис. 8. Приклад графіку процесу реєстрації деякого процесу.

10. Застосувати мережу прямого поширення (Feed-forward, newff) для прогнозування значень часового ряду методом занурення в N-мірний простір. Часовий ряд заданий формулами:

t = 0:0.1:8;

n = length(t);

y = sin(t/5)+rand(1,n)/10-0.05;

Даний часовий ряд наведено на рис. 9. Мережа повинна мати N входів, M нейронів в першому шарі з тангенсальною функцією активації (tansig) і одним нейроном у вихідному шарі з лінійною функцією активації (purelin). Побудувати поверхню помилки прогнозу в залежності від N і від M, а також графік залежності середньої величини помилки від періоду прогнозування.

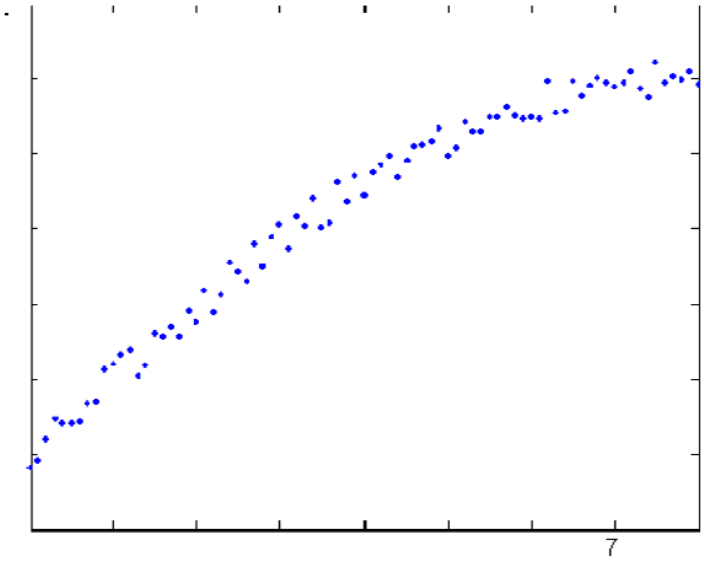

Рис. 9. Часовий ряд для побудови прогнозу.

Зміст звіту:

1. мета роботи
2. завдання
3. для завдання 1: опис алгоритму.
4. для завдання 1: порівняння роботи алгоритму для 3-х функцій різного типу.
5. для завдання 2: опис способу вирішення поставленого завдання.
6. для завдання 2: опис використаних команд та функцій.
7. для завдання 2: графіки, що ілюструють якість апроксимації для різних N.
8. висновки по роботі.

Список літератури

1. Руденко О.Г., Бодянський Є.В. Штучні нейронні мережі: Навч. посібник. – Харків: ТОВ “Компанія СМІТ”, 2006. – 404 с.

2. Круглов В.В., Борисов В.В. Искусственные нейронные сети: теория и практика.-М.: Горячая линия - Телеком, 2001.- 382 с.

3. Дубровін В.І., Субботін С.О. Методи оптимізації та їх застосування в задачах навчання нейронних мереж: Навчальний посібник.-Запоріжжя: ЗНТУ, 2003.- 136 с.

4. Дьяконов В.П., Круглов В.В. MATLAB 6.5 SP1/7/7 SP1/7 SP2 + Simulink 5/6. Инструменты искусственного интеллекта и биоинформатики. – М.: СОЛОН-ПРЕСС, 2006. – 456 с.

2.