МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТЕХНОЛОГІЙ

С.В.ТКАЛІЧЕНКО

ШТУЧНІ НЕЙРОННІ МЕРЕЖІ

НАВЧАЛЬНИЙ ПОСІБНИК

КРИВИЙ РІГ 2023

Рецензенти:

Євген АФАНАСЬЄВ – доктор економічних наук, професор професор кафедри менеджменту і адміністрування.

Ігор ДОВГАЛЬ – кандидат економічних наук, доцент,Директор представництва ПрАТ ВНЗ МАУП "Центр дистанційного навчання "Криворізький інститут"".

Володимир ЛИСЕНКО, к.е.н., доцент кафедри інформатики і прикладного програмного забезпечення, Державний університет економіки і технологій

Рекомендовано до друку Вченою радою Державного університету економіки і технологій протокол № 10 від 30.03.2023р.

С.В.Ткаліченко. Штучні нейронні мережі: Навчальний посібник. – Кривий Ріг: Державний університет економіки і технологій, 2023. –150 с.

Навчальний посібник містить практичні рекомендації для допомоги здобувачів вищої освіти у вивченні освітньої компоненти “Нейромережі”, зокрема передбачають дослідження моделей архітектур, методологій побудови моделей з використанням штучних нейронних мереж, використаня методів практико-орієнтованого навчання, зокрема: процес аналізу предметної області; розробки моделі варіантів використання із застосуванням шаблонів; програмну реалізацію у вигляді Matlab та C# коду.

Розглянуто сучасні моделі та підходи до використання штучних нейронних мереж.

ЗМІСТ

ВСТУП

Інформатизація суспільства, розвиток інноваційних технологій і, як наслідок, розширення можливостей комп'ютерної обробки інформації сприяють появі нових підходів, серед яких дедалі більше значення набуває інтелектуальний аналіз даних, заснований на використанні методів штучного інтелекту у пошуку нових знань, прихованих у масивах вихідної інформації.

Під інтелектуалізацією інформаційних технологій розуміється процес впровадження елементів штучного інтелекту в технології пошуку, збору, обробки, аналізу, зберігання та передачі інформації, технологій роботи з інформацією, яка є одним з основних імперативів поточного століття, які сприяють подоланню наслідків інформаційного вибуху за рахунок зміни пріоритетів інформаційної діяльності та перенесення акценту з нарощування обсягів інформації на способи здобуття актуальних знань з безмежних інформаційних масивів.

Штучні нейронні мережі, які є основою штучного інтелекту, це математичні моделі, а також їх програмні чи апаратні реалізації, побудовані за принципом організації та функціонування біологічних нейронних мереж – мереж нервових клітин живого організму.

Штучні нейронні мережі являють собою систему сполучених та взаємодіючих між собою простих процесорів (штучних нейронів). Такі процесори зазвичай досить прості, особливо в порівнянні з процесорами, які використовуються в персональних комп'ютерах. Кожен такий процесор ає справу тільки з сигналами, які він періодично отримує, та сигналами, які він періодично надсилає іншим процесорам. Будучи з'єднаними в досить велику мережу з керованим взаємодією, такі локально прості процесори разом здатні виконувати доволі складні завдання.

З точки зору машинного навчання, нейронна мережа є окремим випадком методів розпізнавання образів, дискримінантного аналізу, методів кластеризації тощо. З математичної точки зору, навчання нейронних мереж –

це багатопараметричне завдання нелінійної оптимізації. З погляду кібернетики, нейронна мережа використовується в задачах адаптивного управління та як алгоритми для робототехніки. З точки зору розвитку обчислювальної техніки та програмування, нейронна мережа – спосіб вирішення проблеми ефективного паралелізму. А з погляду штучного інтелекту, нейронна мережа є основою філософської течії коннективізму та основним напрямом у структурному підході з вивчення можливості побудови (моделювання) природного інтелекту за допомогою комп'ютерних алгоритмів.

Нейронні мережі не програмуються у звичному значенні цього слова, вони навчаються. Можливість навчання – одна з головних переваг нейронних мереж перед традиційними алгоритмами. Технічно навчання полягає у знаходженні коефіцієнтів зв'язків між нейронами. У процесі навчання нейронна мережа здатна виявляти складні залежності між вхідними даними та вихідними, а також виконувати узагальнення.

Нейронні мережі застосовуються для багатьох цікавих проблем різних галузях науки, медицини та техніки, а в деяких випадках вони забезпечують високотехнологічні рішення.

Метою навчального посібника є ознайомлення студентів з основами штучних нейронних мереж, використанням сучасних підходів та технологій.

Завдання: вивчення основ конструювання, типів нейронних мереж для якісної розробки програмного забезпечення.

Посібник розглядає на основі прикладів Matlab та C# штучні нейронні мережі таких типів: мережі прямого розповсюдження, радіально-базисні мережі, згорткові мережі. Такі знання забезпечують якісну підготовку фахівця.

Навчальний посібник буде використовуватися у навчальному процесі для підготовки фахівців зі спеціальностей Інженерія програмного забезпечення.

Розділ 1. Біологічний та штучний нейрон.

Задачі теорії штучного інтелекту (ШІ)

Приблизно сорок-п'ятдесят років тому, коли теорія штучного інтелекту почала розвиватися, люди сподівалися змоделювати людський мозок та навчити комп'ютер вирішувати такі ж складні задачі, які вирішує людський мозок. В даний час ця глобальна мета виглядає не дуже досяжною, однак у низці важливих конкретних задач ми просунулися. Розглянемо основні задачі з елементами штучного інтелекту, які реалізуються за допомогою нейромереж.

Класифікація образів. Завдання полягає у визначенні приналежності вхідного образа (мовного сигналу або рукописного символу), представленого вектором ознак, одному або декільком попередньо визначеним класам. Використовується для розпізнавання букв, розпізнавання мови, класифікація сигналу електрокардіограм, класифікації клітин крові, забезпечення діяльності біометричних сканерів і т. п.

Кластеризація / категоризація. При вирішенні задачі кластеризації, яка відома також як класифікація образів без вчителя, відсутня навчальна вибірка з мітками класів. Алгоритм кластеризації заснований на подобі образів і розміщує близькі образи в один кластер. Використовується для стиснення даних і дослідження їх властивостей.

Апроксимація функцій. Якщо є навчальна вибірка пар даних вхід-вихід, яка генерується невідомою функцією, аргумент якої спотворено шумом. Завдання апроксимації полягає в знаходженні та оцінці цієї функції. Типовим прикладом є очищення від шуму при прийомі сигналів різної природи. Апроксимація функцій необхідна при вирішенні численних інженерних і наукових задач моделювання.

Інший приклад: курси валют, акцій зазвичай сильно вагаються. Ці коливання пов'язані з тим, що на ринку є багато агентів, які діють, як правило, неузгоджено. Трейдер вирішує зокрема таке завдання — виділити серед цих коливань так званий основний тренд, щоб зрозуміти, чи варто вкладати свої гроші в акції. Основний тренд є якоюсь усередненою кривою, де дрібні коливання (шум) повинні бути усунені в результаті фільтрації.

Прогнозування (екстраполяція). Нехай задані n дискретних значень деякого процесу $\{y(t_1), y(t_2), y(t_3),..., y(t_n)\}$ в послідовні моменти часу $\{t_1, t_2, t_3,..., t_n\}$. Завдання полягає в передбаченні майбутнього значення $y(t_{n+p})$, де p – кількість періодів прогнозування. Прогноз має значний вплив на прийняття рішень в бізнесі, науці і техніці. Типовий приклад це прогнозування цін на фондовій біржі, прогноз погоди тощо.

Оптимізація. Численні проблеми в математиці, статистиці, техніці, науці, медицині та економіці можуть розглядатися як проблеми оптимізації. Завданням алгоритму оптимізації є знаходження такого рішення, яке задовольняє систему обмежень і максимізує або мінімізує цільову функцію. Завдання комівояжера, оптимальне завантаження літака, призначення працівників на посади є класичними прикладами завдань оптимізації.

Управління. Розглядаються динамічні системи, задані сукупністю векторів вхідного керуючого впливу, поточного стану, вихідного вектора системи. Класичною постановкою завдання є розрахунок керуючого впливу для перебування системи в конкретний момент часу в певному бажаному стані. Прикладом є оптимальне управління двигуном, рульове управління на кораблях, літаках і т.п..

Асоціативна пам'ять. Асоціативна пам'ять є особливим видом машинної пам'яті, що використовується для дуже швидкого пошуку. Відома також як пам'ять, що адресується за вмістом, або асоціативний масив, хоча останній термін частіше використовується в програмуванні для позначення структури даних.

У моделі обчислень фон Неймана (архітектура фон Неймана це широко відомий принцип спільного зберігання команд і даних в пам'яті комп'ютера. Обчислювальні машини такого роду часто називають «машина фон Неймана», однак відповідність цих понять не завжди однозначна. У загальному випадку, коли говорять про архітектуру фон Неймана, мають на увазі принцип зберігання даних і інструкцій в одній пам'яті) звернення до пам'яті доступно тільки за допомогою адреси, який не залежить від змісту пам'яті. Асоціативна пам'ять доступна за вказівкою заданого змісту. Асоціативна пам'ять може знаходить застосування при створенні мультимедійних інформаційних баз даних.

Ще кілька важливих задач – це розпізнавання спаму в Інтернеті, автоматичний переклад, розпізнавання почерку тощо. Завдання розпізнавання мови також належить до класу задач ШІ.

Популярне нині і вкрай важливе завдання щодо виділення основного тренду — це завдання аналізу кліматичних даних з метою з'ясування, є глобальне потепління чи ні.

Теорія систем штучного інтелекту використовує такі розділи математики:

• теорія багатовимірних векторних просторів;

•алгоритми, оцінка швидкодії алгоритмів; жадібні алгоритми, алгоритм градієнтного спуску;

• динамічне програмування;

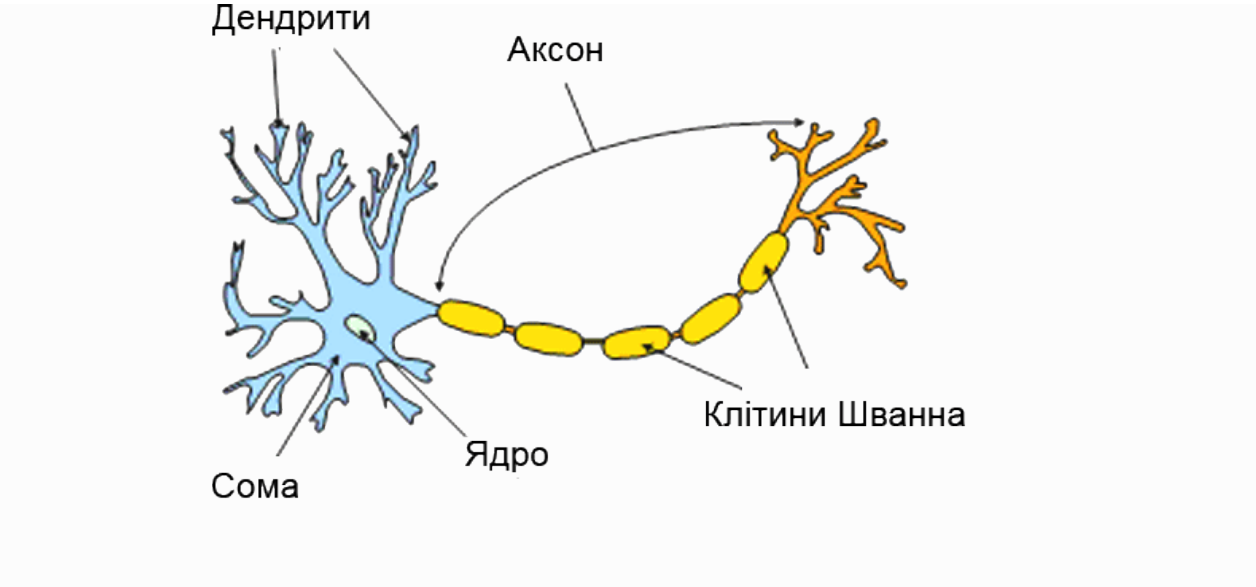
• динамічні системи та атрактори;

• функції Ляпунова.

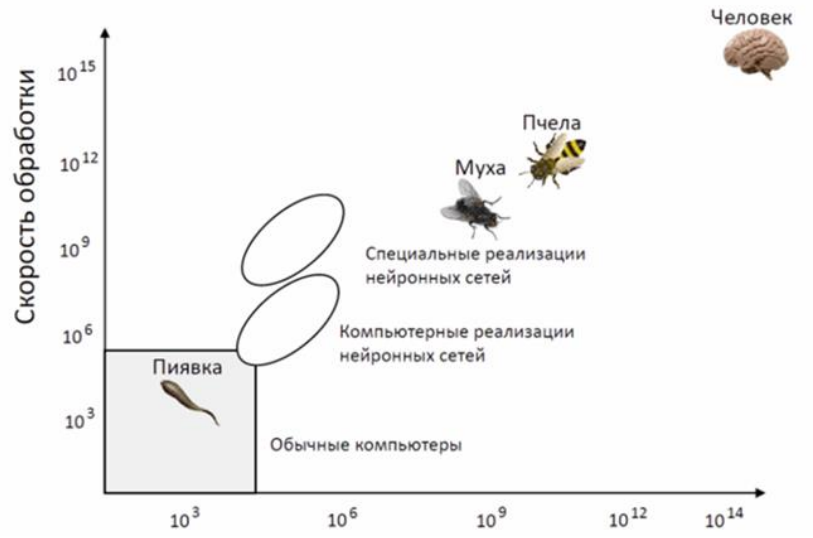
Знати все це не обов'язково, хоч і корисно.

Біологічний нейрон.

Нейрони - взаємопов'язані нервові клітини мозку людини. Мозок людини складається з білої і сірої речовини: біле - це тіла нейронів, а сіре – це нервові волокна що з'єднують їх. Кожен нейрон складається з трьох частин: тіла клітини, дендритів і аксона.



Нейрон отримує інформацію через свої дендрити, а передає її далі через аксон, що розгалужується на кінці на тисячі синапсів - нервових ниток, що з'єднують нейрони між собою. Не всі тварини мають нейрони - пластинчасті і губки взагалі не мають нервових клітин.

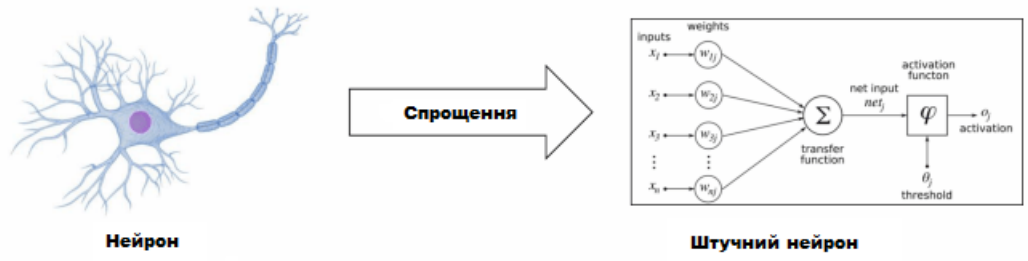


Весь людський мозок містить 86 мільярдів нейронів; приблизно 16 мільярдів нейронів знаходяться в корі великих півкуль. Нейрони взаємодіють між собою за допомогою коротких серій імпульсів тривалістю кілька мікросекунд. Частота імпульсів становить від декількох одиниць до сотень герц, що в мільйон разів повільніше, ніж в сучасних електронних схемах.

Проте, такі складні операції, як розпізнавання образу, людина виконує за кілька сотень мікросекунд. Якщо врахувати, що швидкість виконання операцій нейронами становить одиниці мікросекунд, то вся операція розпізнавання вимагає близько 100 послідовних нейронних операцій. Це означає, що при розпізнаванні образів людський мозок запускає паралельні програми, кожна з яких має не більше ста кроків. Зроблений висновок відомий під назвою «правило ста кроків».

Приблизно однакова кількість нейронів містять мозок вченого, політичного діяча і спортсмена. Відмінність полягає в силі синаптичних зв'язків, себто величині електропровідності нервових волокон, що з'єднують нейрони. На цій підставі була висловлена гіпотеза про те, що всі наші думки, емоції, знання, вся інформація, що зберігається в людському мозку, закодована у вигляді сил синаптичних зв'язків.

Якщо врахувати, що таких зв'язків в людському мозку 10^{14} ... 10^{15}, то виходить, що саме такий розмір має матриця кодів, що зберігається в мозку. А процес навчання людини який триває все його життя, полягає в безперервному коригування вмісту цієї матриці.



Нейрон Штучний нейрон

Отже, що ми знаємо про мозок? Дуже коротко:

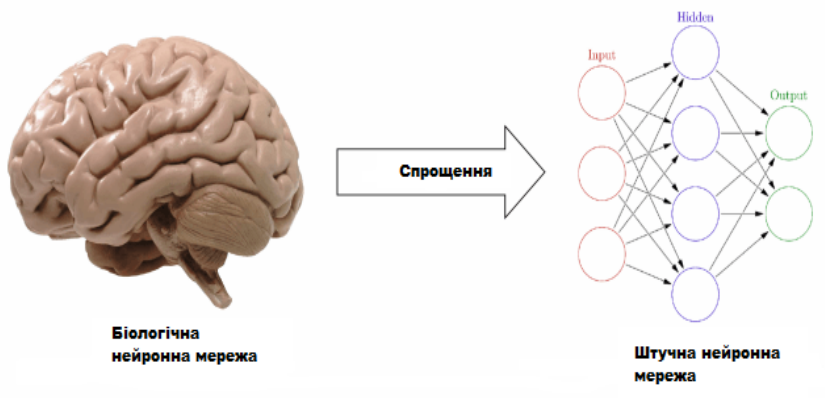
• мозок складається з нейронів;

• нейрони пов'язані один з одним;

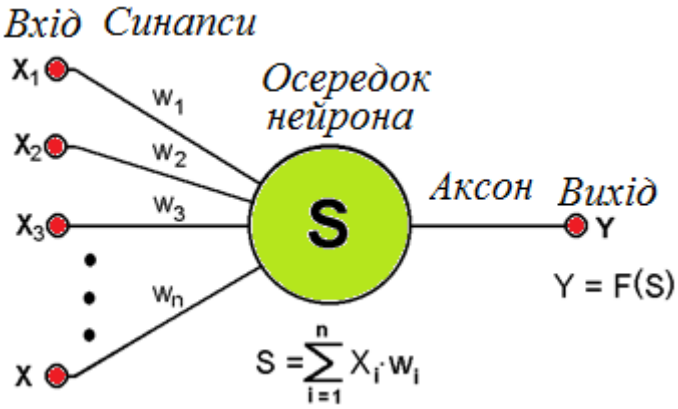
• нейрони обмінюються сигналами;

• сигнали мають булеву природу;

• система зв'язаних нейронів – це стохастична динамічна система.



Штучний нейрон



Алгоритм функціонування штучного нейрона

- множина вхідних сигналів, позначених $x_1, x_2, \ldots, x_n$ поступає на штучний нейрон
- сигнал множиться на відповідну вагу (силу зв'язку) $w_1, w_2, \ldots, w_n$ і передається на сумуючий блок
- сумуючий блок, який відповідає тілу біологічного нейрона,
- складає зважені входи алгебраїчно
- значення аксона нейрона за формулою $Y = F(S)$

де F – деяка функція, яка називається активаційною

ШНМ значною мірою запозичують принципи роботи головного мозку. Знання в них не відокремлені від процесора, а рівномірно розподілені і існують неявно у вигляді сил синаптичних зв'язків. Такі знання не закладаються спочатку, а продукуються в процесі навчання.

Приклад:

Сигнали на входах:

x1=0.35; x2=0.12; x3=0.6;

w1=0.1; w2=0.3; w3=0.2.

Тоді на вхід функції активації буде передано

(0.35 * 0.1) + (0.12 * 0.3) + (0.6 * 0.2) = 0.191

Спрощено, нейрон - це порогова система, яка отримує вхідні сигнали від інших нейронів, підсумовує їх і, якщо ця сума перевищує якийсь поріг, генерує вихідний сигнал.

Функції активації

Ідеалізована модель такої граничної системи може бути побудована за допомогою так званих **сигмоїдальних функцій**. Типовий графік такої функції має вигляд, поданий на рис. 1.

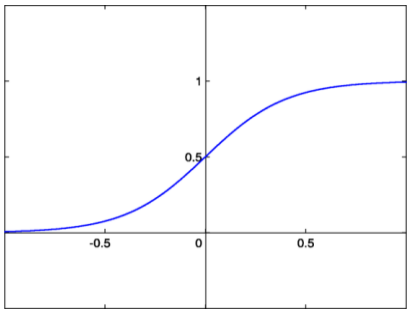


Рис. 1. Графік сигмоїдальної функції

Наведемо як приклад кілька функцій подібного роду:

$$\frac{1}{1+e^{-ax}}$$

де x — амплітуда вхідного сигналу, який отримує нейрон з інших нейронів; σ - вихідний сигнал нейрона (рис. 1);

- функція стрибка, або функція Хевісайда (рис. 2);

$$\sigma = H(x) = \begin{cases} 1, & \text{якщо x} > 0 \\ 0, & \text{якщо x} \leq 0 \end{cases}$$

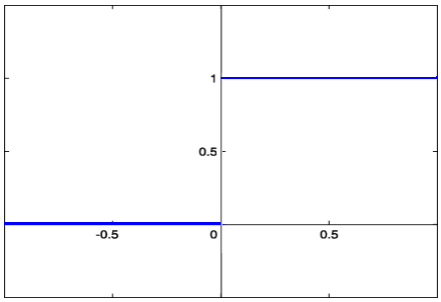


Рис. 2. Функція Хевісайда

При розпізнаванні образів у мозку відбуваються складні процеси. Спрощено систему розпізнавання образів можна як багатошаровий персептрон.

Лінійна функція з насиченням

$$Y = \begin{cases} 1, S \geq T \\ S, 0 \leq S < T \\ 0, S > T \end{cases}$$

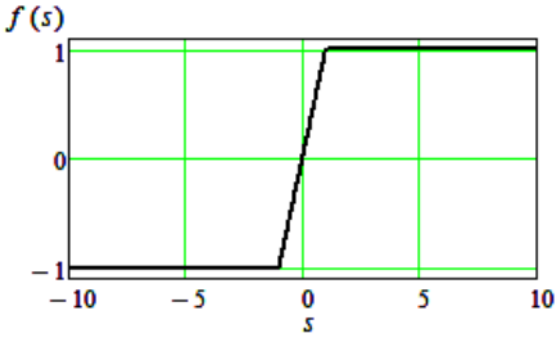


Рис. 3. Функція активації за лінійним законом.

Гіперболічний тангенс

$$Y = th(C * S) = \frac{e^{C*S} - e^{-C*S}}{e^{C*S} + e^{-C*S}} (4)$$

де C > 0 – коефіцієнт ширини сигмоїди по осі абсцис (зазвичай C=1).

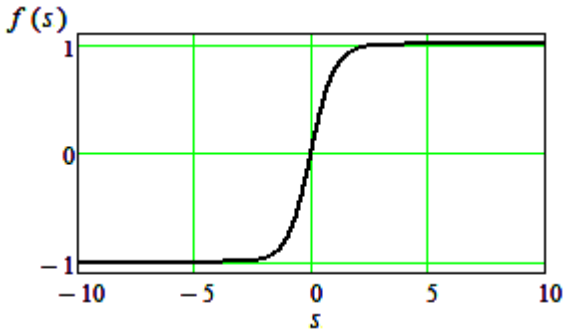
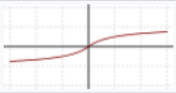
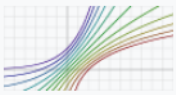
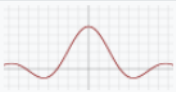
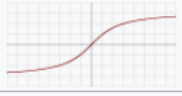


Рис. 4. Функція активації гіперболічний тангенс.

Має сенс використання тільки тоді, коли вхідні значення можуть бути і негативними, і позитивними, як правило діапазон [-1,1]. Використовувати цю функцію тільки з позитивними значеннями недоцільно.

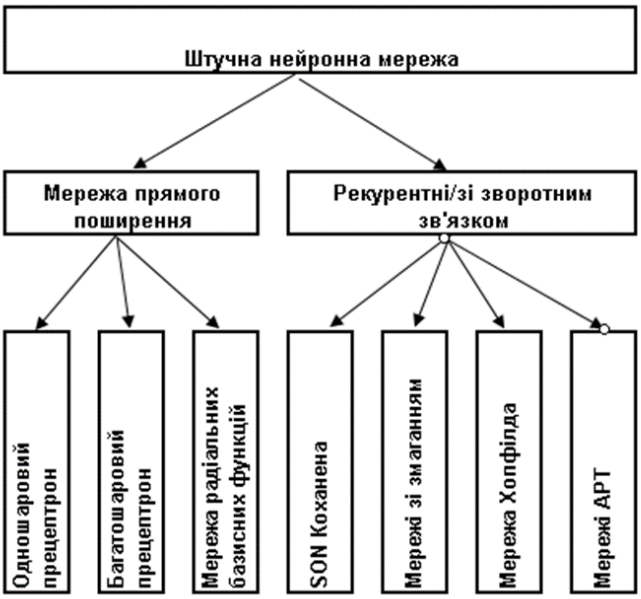
Для різних задач використовуються різні типи функції активації. Деякі з них представлені на рисунку.

th		$f(x) = \mathrm{th}\,(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$	$f'(x) = 1 - f(x)^2$
Softsign[9][10]		$f(x) = \frac{x}{1 + \lvert x \rvert}$	$f'(x) = \frac{1}{(1 + \lvert x \rvert)^2}$
SoftPlus[21]		$f(x) = \ln(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$
SoftExponential[23]		$f(\alpha, x) = \begin{cases} -\frac{\ln(1-\alpha(x+\alpha))}{\alpha} & \alpha < 0 \\ x & \alpha = 0 \\ \frac{e^{\alpha x}-1}{\alpha} + \alpha & \alpha > 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} \frac{1}{1-\alpha(\alpha+x)} & \alpha < 0 \\ e^{\alpha x} & \alpha \geqslant 0 \end{cases}$
Sinc		$f(x) = \begin{cases} 1 & x = 0 \\ \frac{\sin(x)}{x} & x \neq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & x = 0 \\ \frac{\cos(x)}{x} - \frac{\sin(x)}{x^2} & x \neq 0 \end{cases}$
Сигмоидно-взвешенная линейная функция (англ. *Sigmoid-weighted linear unit*, SiLU)[22]		$f(x) = x \cdot \sigma(x)$[5]	$f'(x) = f(x) + \sigma(x)(1 - f(x))$[6]
arctg		$f(x) = \mathrm{tg}^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$

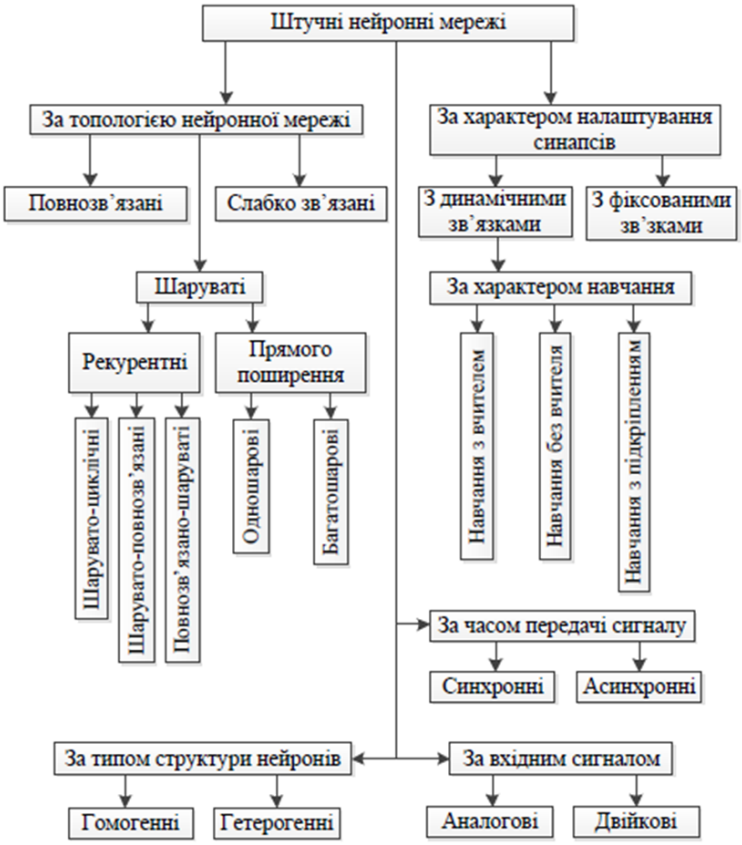
Розділ 2 Штучні нейронні мережі.

Класифікація нейронних мереж

Поділ ШНМ за структурою та завданнями, які вони виконують



Більш повна класифікація нейронних мереж наведено на наступному рисунку.



Вибір параметрів ШНМ.

Якими б досконалими не були методи та алгоритми, що використовуються для класифікації, вони не дадуть коректних результатів, якщо застосовуються до брудних «даних». Тому першим кроком побудови класифікаційної моделі на основі ШНМ є попередня підготовка даних.

Першим кроком у напрямі є добір ознак, значимих з погляду відмінності класів. Наприклад, якщо об'єкти різних класів мають приблизно однаковий розмір, то використання «габаритних» ознак не має сенсу. Зменшення кількості ознак, що використовуються, погіршують роздільність класів. Наприклад, може скластися ситуація, коли в об'єктів різних класів виявляться однакові значення ознак і виникне суперечність.

Наприклад, в завданні класифікації позичальників на «поганих» і «хороших» можна залишити лише дві ознаки «Дохід» і «Вік». Тоді дуже ймовірно, що два позичальники з тим самим віком і доходом виявляться в різних класах. Щоб зробити позичальників помітними потрібно додати ще одну ознаку, наприклад, число утриманців. Таким чином, відбір ознак для навчання класифікатора на основі ШНМ є пошуком компромісу.

Ще одним важливим видом попередньої обробки навчальних даних є нормалізація значень ознак до діапазону 0..1. Нормалізація необхідна, оскільки класифікуючі ознаки мають різну фізичну природу і їх значення можуть відрізнятися на кілька порядків (наприклад "Дохід" і "Вік").

Принциповою відмінністю завдання класифікації від завдання чисельного передбачення є те, що вихідна змінна дискретна (мітка класу чи його числовий код). Оскільки ШНМ є моделями, які використовують навчання з учителем, змінна класу має бути задана для кожного навчального прикладу.

У найпростішому випадку, якщо класифікація бінарна, завдання може бути вирішена за допомогою ШНМ з єдиним нейроном вихідного шару, на виході якого формується два можливі стани (наприклад, 0 і 1). Якщо класів кілька, необхідно вирішувати проблему їх валідації на виході мережі. Насправді

зазвичай використовується вихідний вектор, елементами якого є мітки чи номери класів.

В іншому способі представлення номер класу кодується в двійковій формі у вихідному векторі мережі. Тоді якщо число класів дорівнює 5, то для їх уявлення буде достатньо трьох вихідних нейронів, а код, що відповідає, скажімо, 3-му класу буде 011. Недоліком підходу є відсутність можливості використання показника впевненості, оскільки різниця між будь-якими елементами вихідного вектору завжди дорівнює 0 або 1. Отже, зміна будь-якого елемента вихідного вектору неминуче призведе до помилки. Тому для збільшення відстані між класами зручно використовувати код Хемінга, який дозволить точність класифікації.

Вибір розміру мережі

Для побудови класифікатора, що ефективно працює, дуже важливо правильно вибрати розмір мережі, а саме кількість зв'язків між нейронами, які налаштовуються в процесі навчання і обробляють вхідні дані при її роботі. З одного боку, якщо їх в мережі буде мало, то вона не зможе реалізовувати складні функції поділу класів. З іншого боку, збільшення кількості зв'язків призводить до зростання інформаційної ємності моделі (ваги працюють як елементи пам'яті).

В результаті, коли кількість зв'язків у мережі перевищить кількість прикладів навчальної вибірки, мережа не апроксимуватиме залежності в даних, а просто запам'ятає і відтворюватиме комбінації вхід-вихід з навчальних прикладів. Такий класифікатор чудово працюватиме на навчальних даних і видаватиме довільні відповіді на нових даних, які не брали участь у процесі навчання. Іншими словами, мережа не придбає узагальнюючу здатність і використовувати практично побудований на її основі класифікатор буде безглуздо.

Вибір розміру мережі

Перший спосіб полягає в виборі мережі мінімального розміру, який потім нарощують в залежності від якості роботи та величини похибки. Недоліком

такого підходу є те, що після кожного нарощування мережу необхідно перенавчати.

Альтернативний метод полягає в виборі мережі підвищеного розміру, з її поступовим зменшенням до отримання наперед заданої точності та результатів.

Крім того існує правило: число прикладів у навчальній множині повинно бути більше числа ваг мережі, що настроюються. В іншому випадку мережа не набуде узагальнюючої здатності і видаватиме на нових даних довільні значення.

Для контролю узагальнюючої здатності мережі, на основі якої будується класифікатор, корисно використовувати тестову множину, що формується з прикладів які випадково відбираються, з навчального набору даних. Приклади тестової множини не беруть участь у процесі навчання мережі (тобто. не впливають на підстроювання її ваг), а просто подаються на її вхід разом із навчальними прикладами.

Якщо мережа показує високу точність як на навчальній, так і тестовій множині, можна говорити, що мережа придбала узагальнюючу здатність. Якщо мережа видає хороші результати лише на навчальних даних і погані на тестових, то узагальнююча здатність не надбана.

Часто помилку мережі на навчальній множині називають помилкою навчання, а на тестовій — помилкою узагальнення. Співвідношення розмірів навчальної та тестової множин, в принципі, може бути будь-яким. Головне, щоб у навчальній множині залишалося достатньо прикладів для якісного навчання моделі.

Очевидним способом поліпшення узагальнюючої здатності мережі є збільшення кількості навчальних прикладів або скорочення кількості зв'язків. Перше не завжди можливе через обмежений обсяг набору даних та зростання обчислювальних витрат. Скорочення числа зв'язків призводить до погіршення точності мережі. Тому вибір розміру моделі часто виявляється досить складним завданням, що вимагає багаторазових експериментів.

Вибір архітектури мережі

Як зазначалося вище, жодних спеціальних архітектур нейромереж для вирішення завдань класифікації не використовується. Типовим рішенням тут є мережі прямого розповсюдження із послідовними зв'язками (персептрони). Зазвичай випробовується кілька конфігурацій мережі з різною кількістю нейронів і методів організації в шарах.

При цьому основним показником для вибору є обсяг навчальної множини та досягнення узагальнюючої здатності мережі.

Алгоритм побудови класифікатора

Підготовка данних

- Скласти базу даних із прикладів, характерних для даної задачі
- Розбити всю сукупність даних на дві множини: навчальну та тестову (можливе розбиття на 3 множини: навчальна, тестова та валідаційна)

Попередня обробка даних

- Провести вибір ознак, значимих з погляду завдання класифікації.
- Виконати трансформацію та при необхідності очищення даних (нормалізацію, виключення дублікатів та протиріч, придушення викидів тощо). В результаті бажано отримати лінійно розділяється за класами простір безлічі прикладів.
- Вибрати систему кодування вихідних значень (класичне кодування, «2 на 2»-кодування тощо)

Конструювання, навчання та оцінка якості мережі

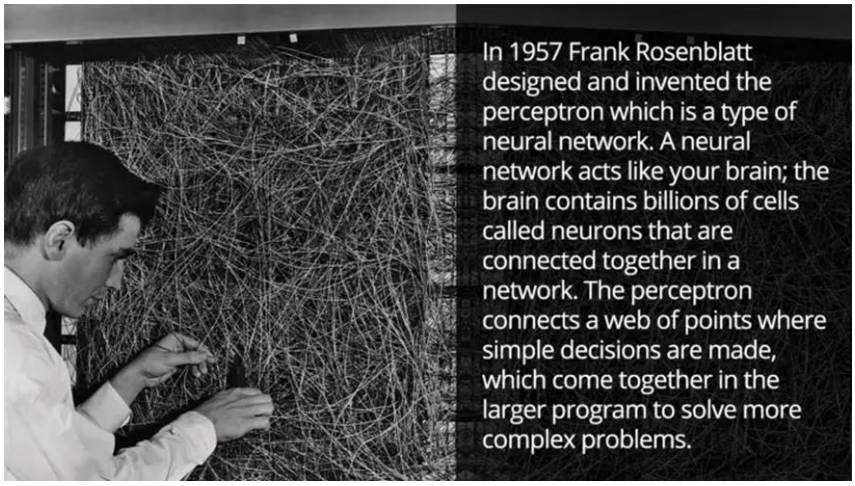
- Вибрати топологію мережі: кількість шарів, число нейронів у шарах тощо.
- Вибрати активаційну функцію нейронів
- Вибрати алгоритм навчання мережі
- Оцінити якість роботи мережі на основі валідаційної множини, або іншого критерію, оптимізувати архітектуру (зменшення ваг, проріджування простору ознак)
- упиниться на варіанті мережі, який забезпечує найкращу здатність до узагальнення та оцінити якість роботи

Використання та діагностика

- З'ясувати ступінь впливу різних факторів на рішення (евристичний підхід).
- Переконатись, що мережа забезпечує необхідну точність класифікації (число неправильно розпізнаних прикладів мале)
- При необхідності повернутися на етап 2, змінивши спосіб подання прикладів або змінивши базу даних
- Практично використовувати мережу для вирішення задачі

Одношарова нейронна мережа (персептрон).

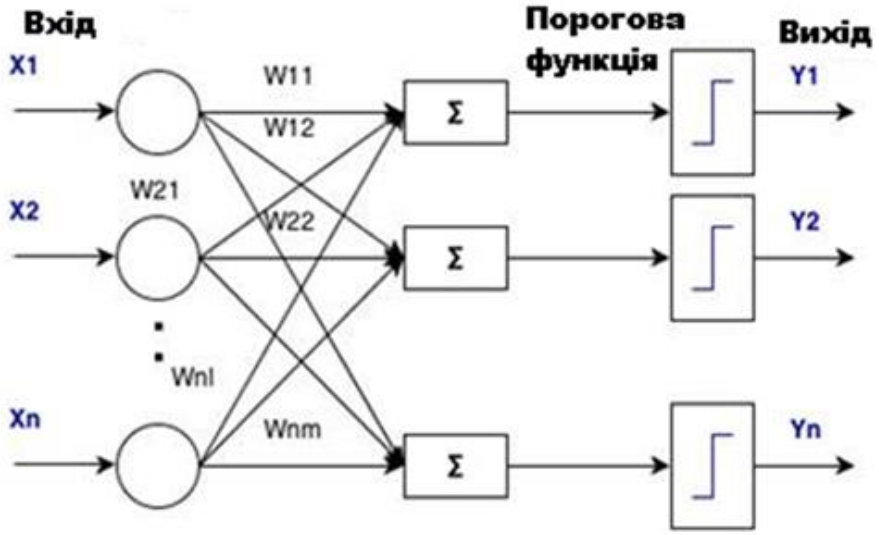
Поняття штучного нейрона і штучної нейронної мережі з'явилися досить давно, ще в 1943 році. Це була чи не перша спроба змоделювати роботу мозку. Її автором був Уоррен Мак-Каллок. Ці ідеї продовжив нейрофізіолог Френк Розенблатт. Він назвав його «персептрон» (від латинського perceptio - сприйняття).



Френк Розенблатт та його персептрон

Персептрони стали дуже активно досліджувати. На них покладали великі надії. Однак, як виявилося, вони мали серйозні обмеження.

Модель одношарового персептрона



Персептрон може бути використаний для вирішення задач класифікації двох класів. Узагальнену форму алгоритму можна записати у вигляді:

$$y(x) = sign(w \cdot x + b)$$

Нелінійна активація підписати функція:

$$sign(k) = \begin{cases} +1, & k \geq 0 \\ -1, & k < 0 \end{cases}$$

У той час, як логістична регресія націлена на ймовірність того, що події відбудуться чи ні, діапазон цільового значення становить $[0, 1]$. Персептрон використовує зручніші цільові значення $t = +1$ для першого класу та $t = -1$ для другого класу. Інше обмеження пов'язане з тим, що алгоритм може обробляти лише лінійні комбінації фіксованої базисної функції.

Стохастичний градієнтний спуск для персептрону

Згідно з попередніми двома формулами, якщо запис класифіковано правильно, то $y \cdot (w \cdot x + b) > 0$ або $y \cdot (w \cdot x + b) < 0$.

Тому мінімізація значення функції вартості персептрона:

$$minL(w, b) = -\sum_{x_i \in M} y_i \cdot (w \cdot x_i + b)$$

M означає набір помилково класифікованих елементів.

Взявши приватну похідну, ми можемо отримати градієнт функції вартості:

$$\nabla_w L(w, b) = -\sum_{x_i \in M} y_i \cdot x_i \qquad \nabla_b L(w, b) = -\sum_{x_i \in M} y_i$$

Персептрон може використовувати лише стохастичний градієнтний спуск. Нам потрібно ініціалізувати параметри ваг також, а потім випадковим чином вибрати один помилково класифікований запис і використовувати метод для ітеративного оновлення параметрів, ваг і записів класифікованих неправильно: $w_{i+1} = w_i + a \cdot y_i \cdot x_i \;\; b_{i+1} = b_i + a \cdot y_i$.

Зверніть увагу, що швидкість навчання коливається від 0 до 1.

Наприклад, у нас є 3 записи, Y1 = (3, 3), Y2 = (4, 3), Y3 = (1, 1), Y1 а також У2 позначені як +1 і Y3 позначений як -1. Враховуючи, що всі початкові параметри дорівнюють 0. Тому всі точки класифікуватимуться як клас 1.

Стохастичний градієнтний метод циклічно перебирає всі тренувальні дані. У першому раунді, застосовуючи перші дві формули, Y1 а також У2 можна класифікувати правильно. Але Y3 буде неправильно класифіковано.

Припускаючи, що швидкість навчання дорівнює 1, застосовуючи градієнтний спуск, показаний вище, ми можемо отримати:

$$w = w + y_3 \cdot x_3 = (0,0) + -(1,1) = (-1,-1)$$

$$b = b + 1 \cdot y_3 = -1$$

Тоді лінійний класифікатор можна записати у вигляді:

$$-x_1 - x_2 - 1 = 0$$

Це один раунд ітерації градієнтного спуску.

Round	Misclassified Record	w1	w2	b	y1	y2	y3
0	y3	0	0	0	0	0	0
1	y1, y2	-1	-1	-1	-7	-8	-3
2	y3	2	2	0	12	14	4
3	y3	1	1	-1	5	6	1
4	y1, y2	0	0	-2	-2	-2	-2
5	y3	3	3	-1	17	20	5
6	x3	2	2	-2	10	12	2
7		1	1	-3	3	4	-1

Якщо ми виконуємо градієнтний спуск знову і знову, у 7 раунді всі 3 записи будуть позначені правильно. Тоді алгоритм зупиниться. Кінцева формула для лінійного класифікатора:

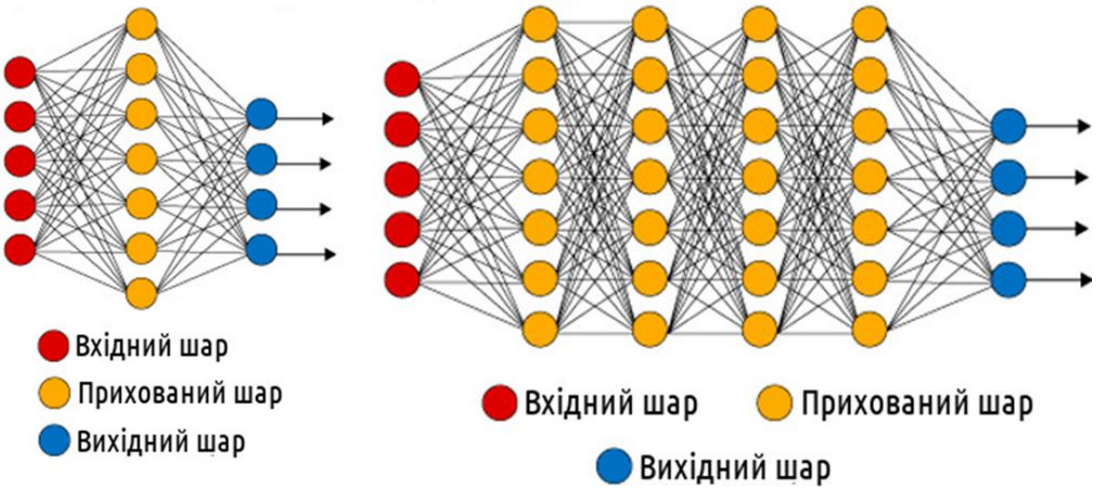
$$x_1 + x_2 - 3 = 0$$

Багатошарові нейронні мережі.

Для рішення більш складних задач можливостей одношарового пресептрона невистачає. Тому використовуються більш складні мережі-багатошарові.

Спочатку йде шар, який називають вхідним чи розподільним. Його нейрони (які також називають вхідними) приймають елементи вектору ознак і розподіляють їх за нейронами наступного шару. При цьому обробка даних у вхідному шарі не провадиться.

Останній шар називається вихідним. На виходах його нейронів (вони називаються вихідними) формується результат роботи мережі елементи вихідного вектору.



Між вхідним та вихідним шаром розташовуються один або кілька проміжних або прихованих шарів. Прихованими вони називаються тому, що їх входи та виходи невідомі для зовнішніх по відношенню до нейронної мережі програм і користувача.

В даний час багатошарова архітектура нейронних мереж є найбільш популярною та добре розробленою. Багатошарова нейронна мережа може моделювати функцію практично будь-якого ступеня складності, причому кількість шарів та число нейронів у кожному шарі визначають складність функції.

Крім одношарового та багатошарового персептрона, які називають мережами прямого розповсюдження, в класифікації наведено інші види ШНМ. Перерахуємо скорочено їх особливості.

Мережі зі зворотними зв'язками.

У мережах такого типу (рис.9) сигнал може йти і в зворотний бік, виходи нейронів можуть повертатися на входи. Це означає, що вихід якогось нейрона визначається не тільки його вагами і вхідним сигналом, але ще і попередніми виходами (так як вони знову повернулися на входи).

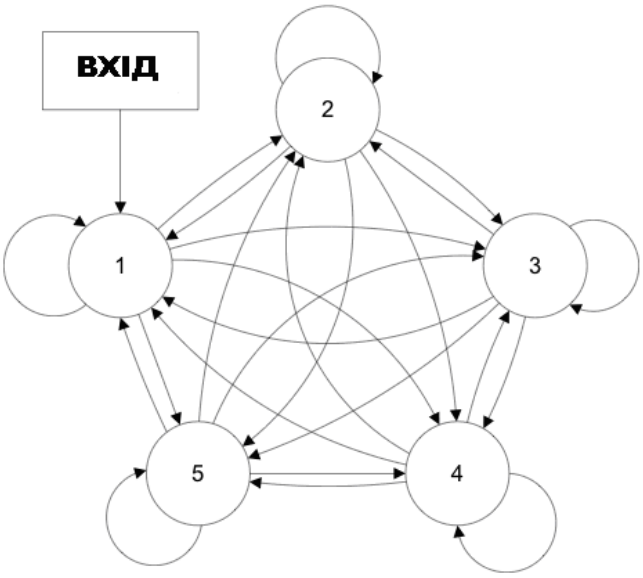


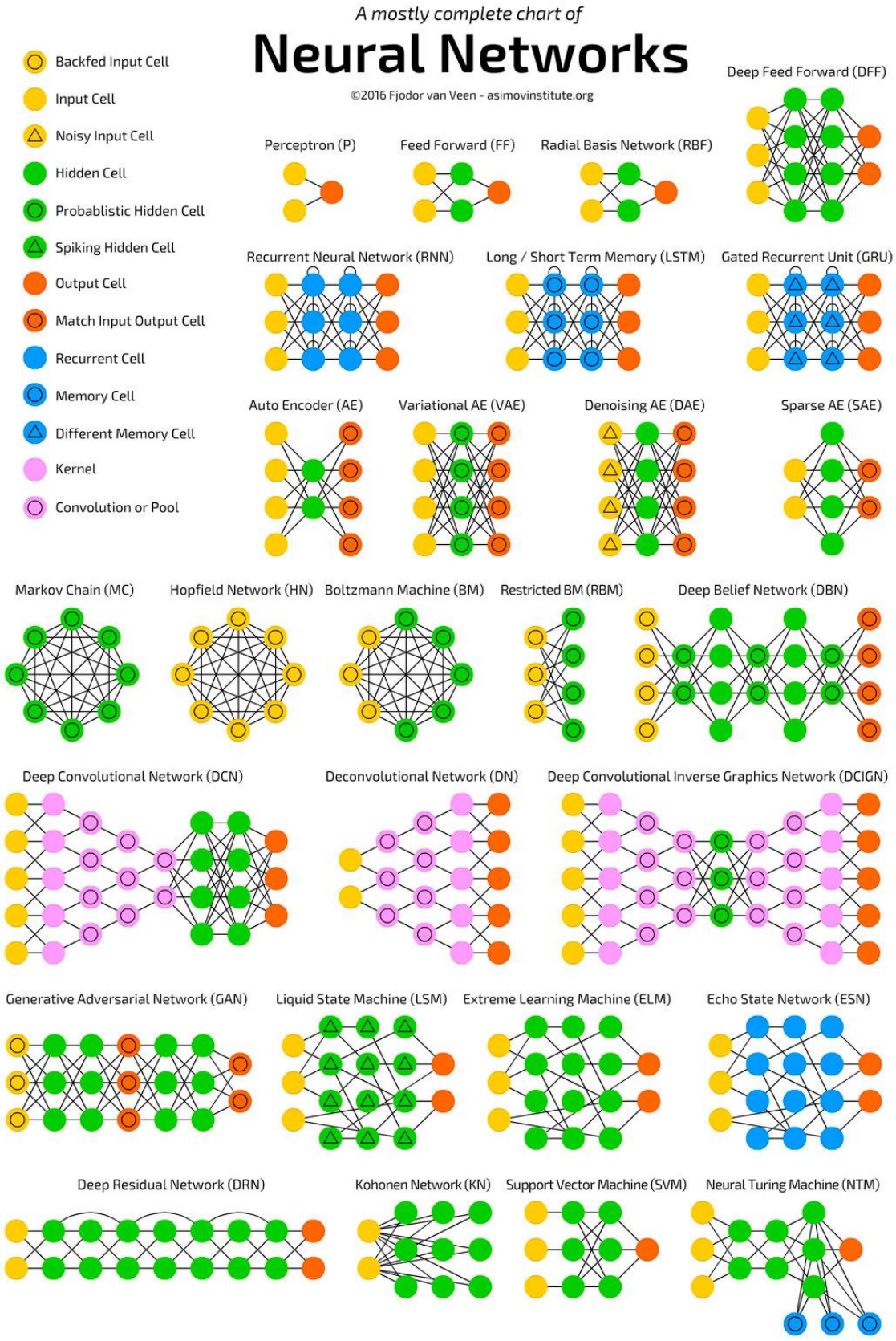
Схема нейтронної мережі зі зворотними зв'язками.

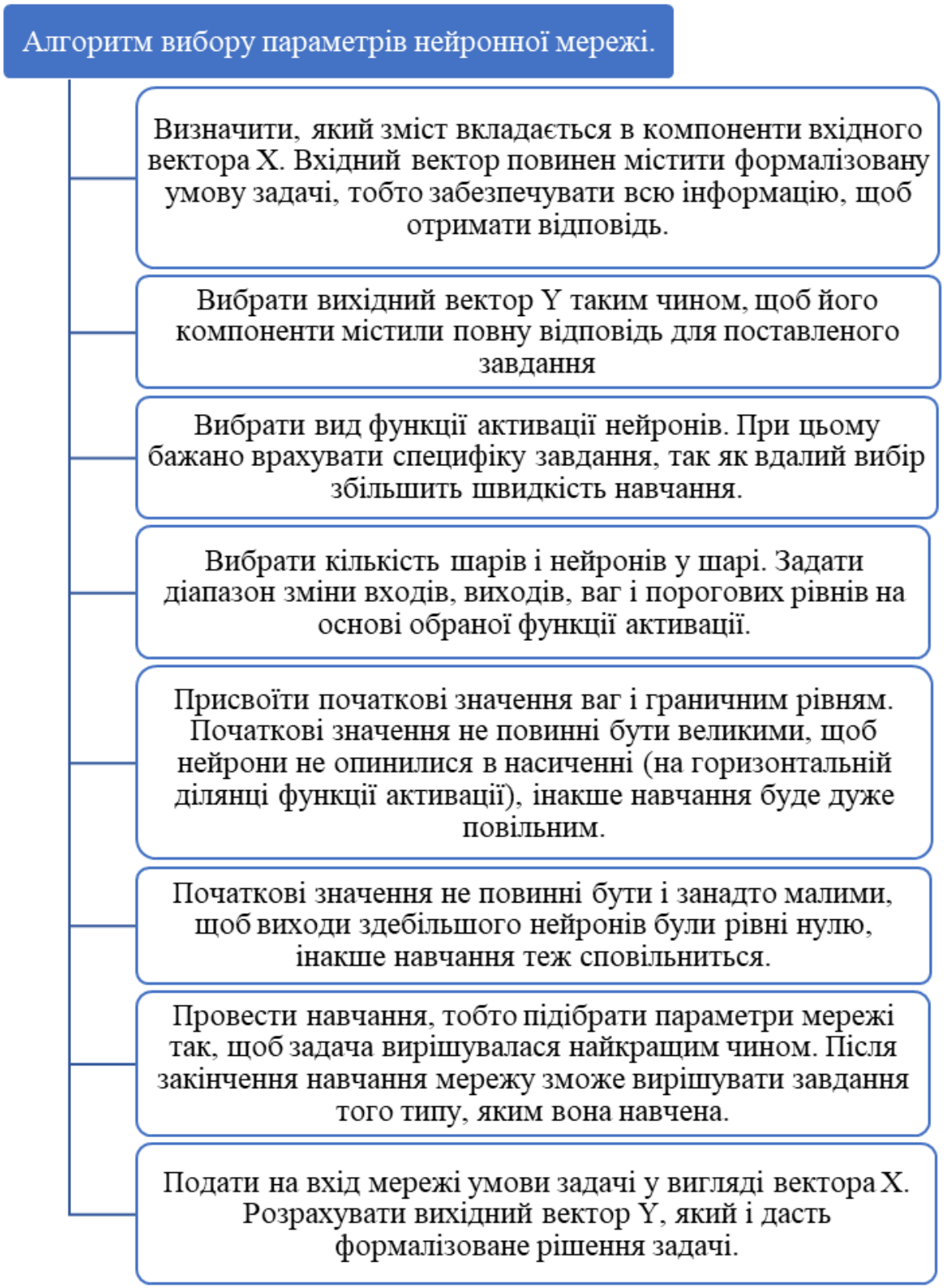
Мережа радіально базисних функцій у математичному моделюванні — це штучна нейронна мережа, яка використовує радіальні базисні функції у якості функції активації. Виходом мережі є лінійна комбінація радіальних базисних функцій входу та параметрів нейрона.

Нейронні мережі Кохонена - клас нейронних мереж, основним елементом яких є шар Кохонена. Шар Кохонена складається з адаптивних лінійних суматорів («лінійних формальних нейронів»). Як правило, вихідні сигнали шару Кохонена обробляються за правилом «Переможець отримує все»: найбільший сигнал перетворюється в одиничний, інші звертаються в нуль.

Нейронна мережа Гопфілда це тип рекурентної, повнозв'язної, штучної нейронної мережі з симетричною матрицею зв'язків. У процесі роботи динаміка таких мереж зводиться до одного з положень рівноваги. Ці положення рівноваги є локальними мінімумами функціоналу, який називається енергією мережі.

Існує багато інших варіантів архітектури ШНМ. Основні з них наводяться в таблиці та на рисунках.




Алгоритм вибору параметрів нейронної мережі.
Визначити, який зміст вкладається в компоненти вхідного вектора X. Вхідний вектор повинен містити формалізовану умову задачі, тобто забезпечувати всю інформацію, щоб отримати відповідь.
Вибрати вихідний вектор Y таким чином, щоб його компоненти містили повну відповідь для поставленого завдання
Вибрати вид функції активації нейронів. При цьому бажано врахувати специфіку завдання, так як вдалий вибір збільшить швидкість навчання.
Вибрати кількість шарів і нейронів у шарі. Задати діапазон зміни входів, виходів, ваг і порогових рівнів на основі обраної функції активації.
Присвоїти початкові значення ваг і граничним рівням. Початкові значення не повинні бути великими, щоб нейрони не опинилися в насиченні (на горизонтальній ділянці функції активації), інакше навчання буде дуже повільним.
Початкові значення не повинні бути і занадто малими, щоб виходи здебільшого нейронів були рівні нулю, інакше навчання теж сповільниться.
Провести навчання, тобто підібрати параметри мережі так, щоб задача вирішувалася найкращим чином. Після закінчення навчання мережу зможе вирішувати завдання того типу, яким вона навчена.
Подати на вхід мережі умови задачі у вигляді вектора X. Розрахувати вихідний вектор Y, який і дасть формалізоване рішення задачі.

Характеристики основних ШНМ

Архітектура	Алгоритм навчання	Задача
Одношаровий та багатошаровий прецептрон	Зворотне розповсюдження. Adaline та Madaline.	Класифікація образів. Апроксимація функцій. Прогнозування. Управління.
Нейронна мережа Гопфілда	Навчання асоціативної пам'яті	Асоціативна пам'ять.
Нейронні мережі Кохонена	SON Кохонена	Категоризація, аналіз даних.
Мережі АРТ	ART Map	Класифікація образів.
Мережа радіально базисних функцій	Алгоритм навчання радіально базисних функцій	Класифікація образів. Апроксимація функцій. Прогнозування. Управління.
Змагання	Векторне квантування	Категоризація всередині класу. Стиснення даних

Задачі, які виконуються за допомогою нейромереж.

Силу зв'язку між нейронами визначає матриця синаптичних зв'язків (W). Відомо, що вся довготривала фундаментальна інформація зберігається у цій матриці. Матриця синаптичних зв'язків повільно змінюється згодом у процесі навчання мозку. Отже, у мозку дві динаміки — швидка динаміка нейронів, що описується динамічною системою, та повільна, пов'язана із зміною сили зв'язку між нейронами. Повільна динаміка спрощено описується так званим правилом Хебба.

Дональд Хебб - канадський фізіолог, який експериментально показав, що якщо два нейрони обидва часто одночасно активні, то сила зв'язку між ними зростає. Аналогічні дослідження проводив Є. Конорський. Можливі різні варіанти математичного запису такого правила.

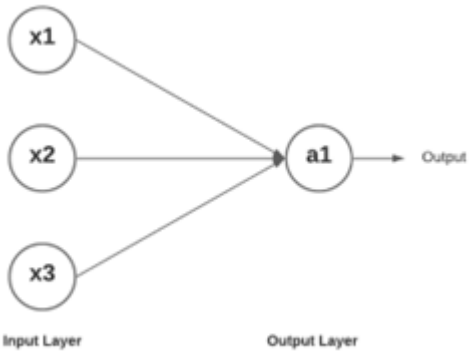
Якщо скористатися найпростішим варіантом правила Хебба, запропонованим С. Фузі, Н. Брюнелем та ін., можна показати, що це правило

в сукупності з шестишаровим персептроном реалізує все найважливіші алгоритми, наприклад, перетворення Фур'є.

Математичне подання нейронних мереж

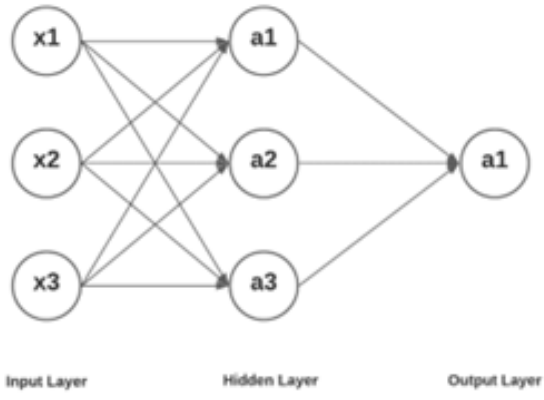
Одношарова нейронна мережа (перцептрон)

Ось як могло б виглядати математичне рівняння для отримання значення a_1 (вихідний вузол) в якості функції вхідних x_1 , x_2 , x_3.



$$a_1^{(2)} = g(\theta_{10}^{(1)} + \theta_{11}^{(1)} + \theta_{12}^{(1)} + \theta_{13}^{(1)})$$

У наведеному вище рівнянні верхній індекс ваги представляє шар, а нижній індекс ваг являє вагу з'єднання між вхідним і вихідним вузлами. Таким чином $\theta_{12}^{(1)}$ являє собою вагу першого рівня між вузлом 1 в наступному шарі і вузлом 2 в поточному шарі.



Нейронна мережа з одним прихованим шаром

Нижче нейронна мережа з одним прихованим шаром, що має три нейрона, прихованим шаром з трьома вхідними нейронами і вихідним шаром з одним нейроном.

Ось так може виглядати математичне рівняння для отримання значеннa_1, a_2, a_3я в шарі 2 в якості опції вхідних x_1, x_2, x_3. Крім того, значення a_1 в шарі 3 представлено як функція значень a_1, a_2, a_3 в шарі 2.

Спочатку давайте уявимо вихідні значення, оброблені в трьох прихованих нейронах прихованого шару. Вхідний шар представлений як шар 1, прихований шар як шар 2 і вихідний шар - це шар 3.

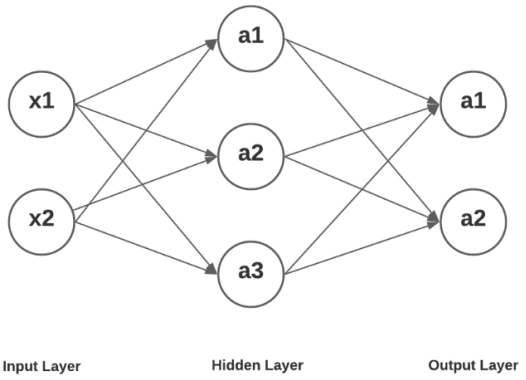
$$a_1^{(2)} = g(\theta_{10}^{(1)} x_0 + \theta_{11}^{(1)} x_1 + \theta_{12}^{(1)} x_2 + \theta_{13}^{(1)} x_3)$$

$$a_2^{(2)} = g(\theta_{20}^{(1)} x_0 + \theta_{21}^{(1)} x_1 + \theta_{22}^{(1)} x_2 + \theta_{23}^{(1)} x_3)$$

$$a_3^{(2)} = g(\theta_{30}^{(1)} x_0 + \theta_{31}^{(1)} x_1 + \theta_{32}^{(1)} x_2 + \theta_{33}^{(1)} x_3)$$

Визначимо вихідний значення вузла в вихідному шарі. Значення представляється як функція від a_1, a_2, a_3 в попередніх вузлах, які можуть бути представлені як значення x_1, x_2, x_3 у вхідному шарі.

$$a_1^{(3)} = g(\theta_{10}^{(2)} a_0^{(2)} + \theta_{11}^{(2)} a_1^{(2)} + \theta_{12}^{(2)} a_2^{(2)} + \theta_{13}^{(2)} a_3^{(2)})$$



Нейронна мережа з одним прихованим шаром (3 нейрона) і вихідним шаром (2 нейрона)

Спочатку давайте уявимо вихідні значення, оброблені в трьох прихованих нейронах прихованого шару. Вхідний шар представлений як шар 1, прихований шар як шар 2 і вихідний шар - це шар 3.

Визначимо вихідний значення вузлів у вихідному шарі. Значення представляється як функція від a_1, a_2, a_3 в попередніх вузлах, які можуть бути представлені як значення x_1, x_2, x_3 у вхідному шарі.

$$a_1^{(2)} = g(\theta_{10}^{(1)} x_0 + \theta_{11}^{(1)} x_1 + \theta_{12}^{(1)} x_2)$$

$$a_2^{(2)} = g(\theta_{20}^{(1)} x_0 + \theta_{21}^{(1)} x_1 + \theta_{22}^{(1)} x_2)$$

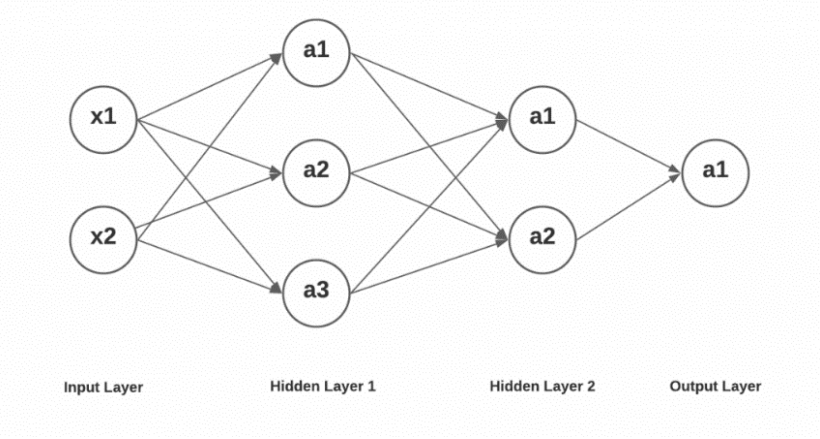
$$a_3^{(2)} = g(\theta_{30}^{(1)} x_0 + \theta_{31}^{(1)} x_1 + \theta_{32}^{(1)} x_2)$$

$$a_1^{(3)} = g(\theta_{10}^{(2)} a_0^{(2)} + \theta_{11}^{(2)} a_1^{(2)} + \theta_{12}^{(2)} a_2^{(2)} + \theta_{13}^{(2)} a_3^{(2)})$$

$$a_2^{(3)} = g(\theta_{20}^{(2)} a_0^{(2)} + \theta_{21}^{(2)} a_1^{(2)} + \theta_{22}^{(2)} a_2^{(2)} + \theta_{23}^{(2)} a_3^{(2)})$$

Мережі глибокого навчання з двома прихованими шарами

Нарешті, давайте подивимося, як вихідні значення вузлів a1 в вихідному шарі можуть бути виражені у вигляді математичних обчислень як функція вхідних сигналів x_1, x_2. Ось схема мережі глибокого навчання, що має два приховані прошарки, один з яких має три вузли, а інший - два вузла. Потім є вхідний рівень з двома вхідними вузлами, і вихідний шар з одним вихідним вузлом. Ось схема спрощеної мережі глибокого навчання.



Значення в шарі 2 (a_1, a_2, a_3) і шарі 3 (a_1, a_2) залишаться такими ж, як у попередньому прикладі. Давайте уявимо значення 1 в вихідному шарі як функцію значень a_1, a_2 в попередньому шарі (шар 3).

$$a_1^{(4)} = g(\theta_{10}^{(3)} a_0^{(3)} + \theta_{11}^{(3)} a_1^{(3)} + \theta_{12}^{(3)} a_2^{(3)})$$

Важливо розуміти позначення, в яких ви будете представляти нейронну мережу як рівняння. Першому або вхідного шару можна призначити номер 1, прихованого номер 2, а вихідного 3. Вагам між вхідним вузлом в одному шарі і вузлом в наступному шарі призначається верхній індекс - значення шару, що складається з вхідного вузла. Нижній індекс ваги складається з двох чисел - числа, що представляє вузол в наступному шарі і номера вхідного вузла.

Навчання штучних нейронних мереж (ШНМ).

Здатність штучних нейронних мереж до навчання є їх найбільш цікавою властивістю. Подібно біологічним системам, які вони моделюють, ці нейронні мережі самі вдосконалюють себе в результаті спроб створити кращу модель поведінки.

Нейронна мережа, в принципі, здатна пристосовуватися до будь-яких даних, щоб звести до мінімуму загальну квадратичну помилку. Щоб уникнути цього, у процесі навчання штучних нейронних мереж застосовують різні

методи керування мережею. Наприклад, для цього навчальна вибірка перед початком процесу навчання випадковим чином ділиться на дві підвибірки: навчальна вибірка та тестова.

Навчальна вибірка використовується в процесі навчання, при цьому вагові коефіцієнти нейронів змінюються. Тестова вибірка використовується у процесі навчання для перевірки на ній сумарної квадратичної помилки, під час цього зміни вагових коефіцієнтів немає.

Якщо нейронна мережа показує поліпшення в апроксимації навчальної та тестової вибірок, то нейронна мережа навчається у правильному напрямку. В іншому випадку помилка в навчальній вибірці може зменшитися, але при цьому збільшитись у тестовій вибірці. Останнє означає, що мережа «перенавчився» і більше не може використовуватися для прогнозування чи класифікації.

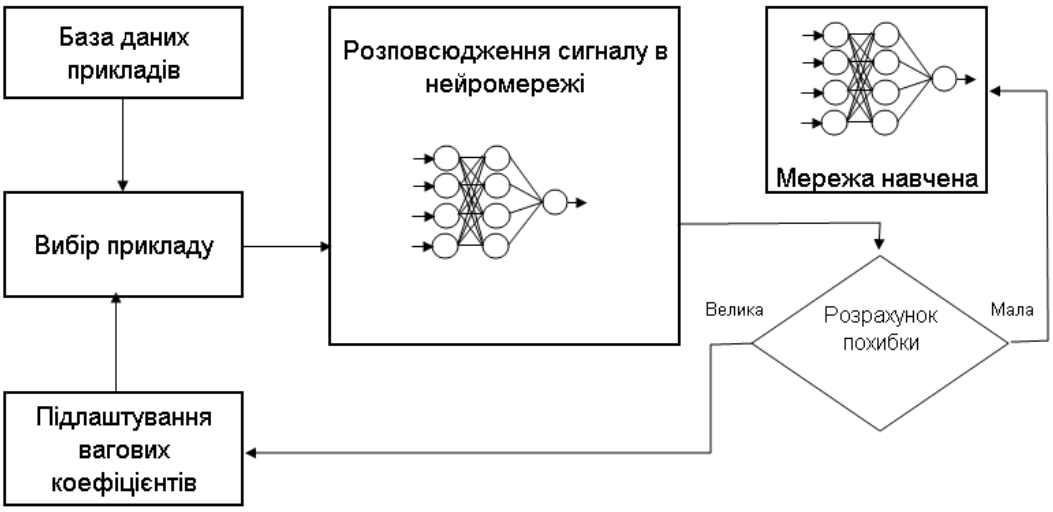
Парадигми навчання ШНМ

Навчання з вчителем (supervised learning) — передбачає, що для кожного вхідного вектора (xi) існує вектор вихідних значень (di). Разом ці два вектора називають навчальною парою $(xi\,, di)$, а множину навчальних пар — навчальною вибіркою. Процес навчання зводиться до почергового подавання на вхід нейронної мережі навчальних пар, вираховування похибки між дійсним і бажаним значенням нейронної $\delta = y - d$ та корегування параметрів мережі в бік зменшення цієї похибки

Навчання без вчителя (unsupervised)— один зі способів машинного навчання, при вирішенні яких випробовувана система спонтанно навчається виконувати поставлене завдання, без втручання з боку експериментатора. З точки зору кібернетики, є одним з видів кібернетичного експерименту. Як правило, це підходить тільки для задач, в яких відомий опис множини об'єктів (навчальна вибірка), і необхідно виявити внутрішні взаємозв'язки, залежності, закономірності, що існують між об'єктами.

Навчання з підкріпленням є проміжним варіантом двох попередніх парадигм . Замість «вчителя» в схему навчання вводиться блок «критика» , який відслідковує реакцію середовища на вхідний сигнал і опираючись на неї визначає евристичну похибку, яку покладено в процес навчання мережі.

Навчена нейромережа далі може узагальнювати (інтерполювати і екстраполювати) отриманий навик рішення і видавати прогноз для нових ситуацій - тих поєднань значень вхідних сигналів, які не входили в "підручник".



Процес навчання штучних нейромереж.

ШНМ формує вихідний сигнал Y, реалізуючи при цьому деяку функцію $Y = G(X)$, функцію вагових коефіцієнтів. Відома функція $Y = F(X)$, яка задається парами (X^1, Y^1) , (X^2, Y^2),…,(X^n, Y^n) вхідних та вихідних даних, для яких $Y^k = F(X^k)(k = 1,2,\ldots,N)$.

Процесом навчання називається синтез функції G близької до функції F з врахуванням похибки E.

$$|\boldsymbol{G(X) - E(X)}| \rightarrow \boldsymbol{F(X)}$$

В залежності від кількості вибраних пар (X^n, Y^n) та способу обчислення похибки задача навчання нейронної мережі зводиться до задачі багатомірної оптимізації з дуже великою розмірністю.

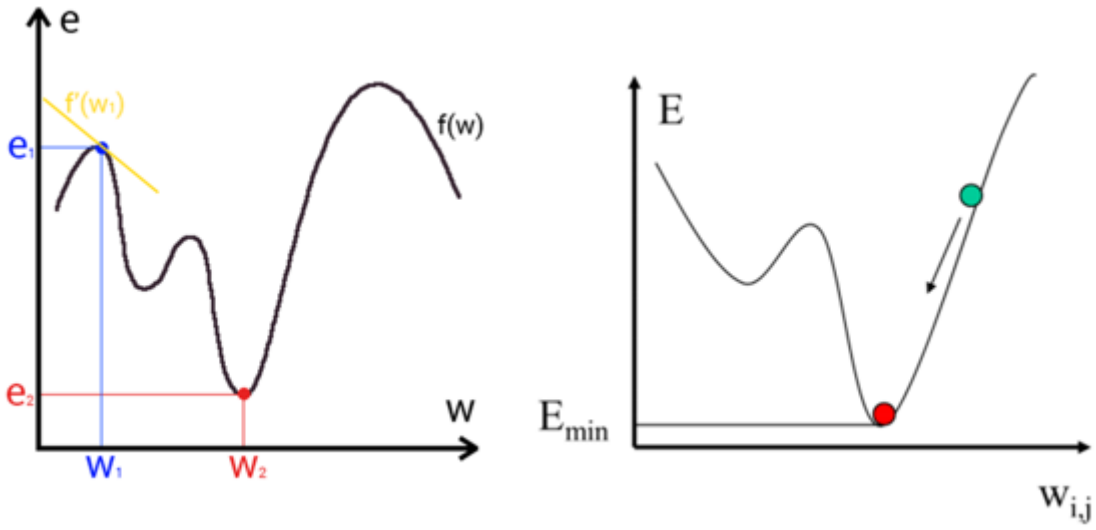
Алгоритми навчання нейронних мереж.

Алгоритми з використанням методів градієнтного спуску.

Градієнт - це вектор, що стосується функції і вказує на найбільше збільшення цієї функції. Градієнт дорівнює нулю за локального максимуму

або мінімуму, тому що немає єдиного напряму збільшення. У математиці градієнт визначається як частинна похідна кожної вхідної змінної функції.

$$\nabla f(x,y) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right)$$



Графічна інтерпретація методу найшвидшого спуску.

Оскільки градієнт - це вектор, що вказує найбільше збільшення функції, негативний градієнт - це вектор, що вказує найбільше зменшення функції. Отже, ми можемо мінімізувати функцію, інтерактивно трохи рухаючись у бік негативного градієнта. Це логіка градієнтного спуску.

Враховуючи відправну точку $\left(x_1^{(0)}, \dots, x_n^{(0)}\right)$ ми можемо побудувати ітераційну процедуру:

$$x_1^{(i+1)} = x_1^{(i)} - \alpha \cdot \frac{\partial f}{\partial x_i}\left(x^{(i)}\right), \dots, x_n^{(i+1)} = x_n^{(i)} - \alpha \cdot \frac{\partial f}{\partial x_n}(x^{(i)})$$

Параметр альфа в наведеній вище формулі називається швидкістю навчання, яка в більшості випадків є невеликою константою і знаходиться в діапазоні від 0 до 1. Ітераційна процедура не зупиниться, доки не зійдеться.

Тому ітераційна процедура градієнтного спуску може бути записана як:

$$x^{(i+1)} = x^{(i)} - \alpha \cdot \frac{\partial f}{\partial x}\left(x^{(i)}, y^{(i)}\right) = (1-\alpha) \cdot x^{(i)}$$

$$y^{(i+1)} = y^{(i)} - \alpha \cdot \frac{\partial f}{\partial y}\left(x^{(i)}, y^{(i)}\right) = (1-\alpha) \cdot y^{(i)}$$

Тоді ми можемо отримати:

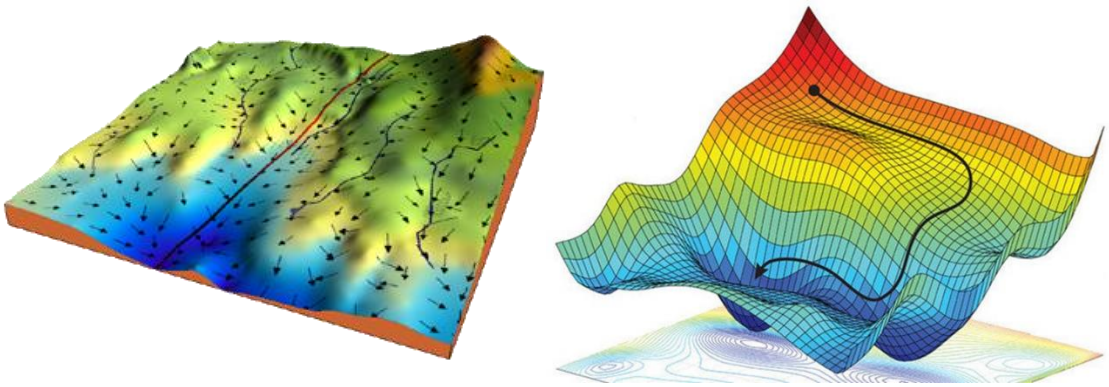
$$x^{(i+1)} = (1-\alpha)^i \cdot x^{(0)}$$

$$y^{(i+1)} = (1-\alpha)^i \cdot y^{(0)}$$

Нарешті, припускаючи, що $\alpha < 1$, тоді ми можемо отримати:

$$\lim_{\Delta x, \Delta y \to 0} \left(x^{(i)}, y^{(i)}\right) = (0,0)$$

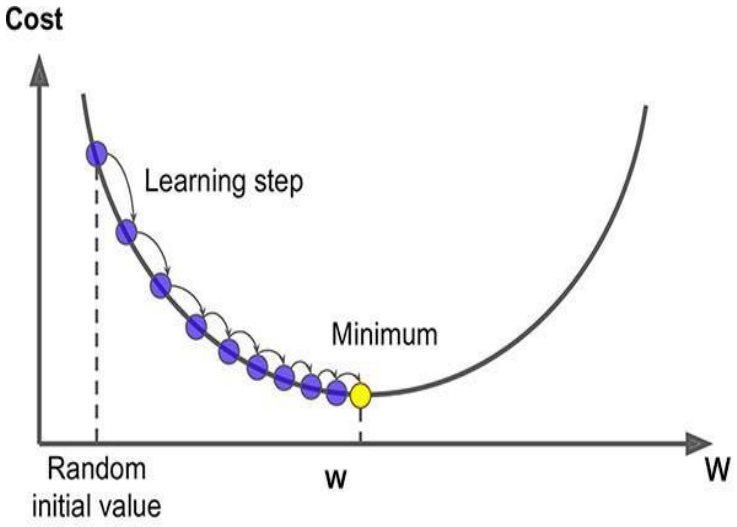
Інший інтуїтивно зрозумілий спосіб пояснити градієнтний спуск полягає в тому, що розглянемо тривимірний графік нижче в контексті функції. Наша мета – перейти від гори у верхньому правому кутку до синього моря у лівому нижньому кутку. Стрілки являють напрямок якнайшвидшого спуску (негативний градієнт) з будь-якої заданої точки - напрямок, який зменшує функцію якнайшвидше.



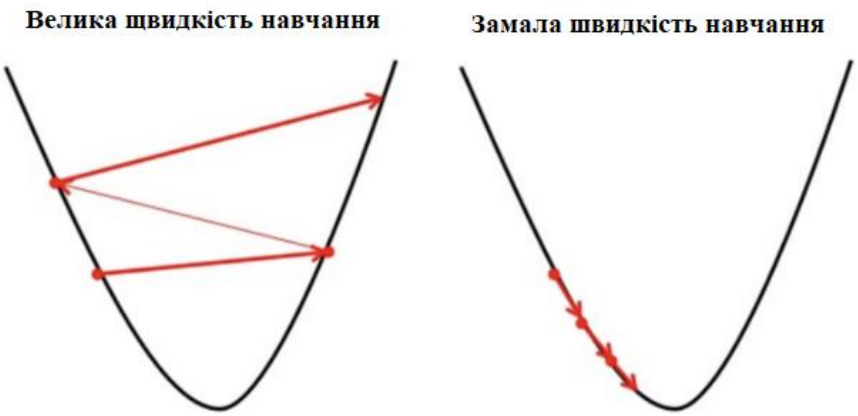
Ілюстрація методу градієнтного спуску в тривимірному просторі.

У машинному навчанні ми приділяємо більше уваги взаємозв'язку між функцією вартості та параметрами, а не залежними та незалежними змінними. Загальна ідея градієнтного спуску в машинному навчанні полягає в ітераційному налаштуванні параметрів мінімізації функції вартості.

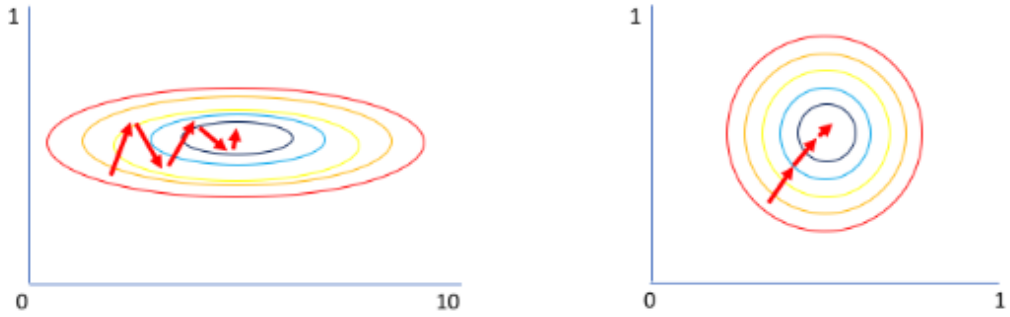
У деяких випадках ми можемо застосувати рівняння у замкнутій формі для безпосереднього розрахунку параметрів, які найкраще відповідають моделі для набору даних навчання.



Важливим параметром у градієнтному спуску є швидкість навчання, що визначає розмір кожного кроку. Коли швидкість навчання занадто велика, градієнтний спуск може перестрибнути через долину і опинитися з іншого боку. Це призведе до розходження функції вартості. З іншого боку, коли швидкість навчання занадто мала, алгоритм сходитиметься довго. Отже, перед початком градієнтного спуску потрібна правильна швидкість навчання.



Нормалізація відіграє важливу роль для градієнтного спуску. Якщо функції не нормалізовані, в оновленні ваг переважатимуть об'єкти з великим масштабом, тому алгоритм генеруватиме зигзагоподібний шлях навчання. Після нормалізації всіх функцій функція вартості набуває більш сферичної форми. Алгоритм градієнтного спуску йде до мінімуму. Один із способів нормалізації - це мінус середнє та поділ на стандартне відхилення.



Три типові градієнтні спуски

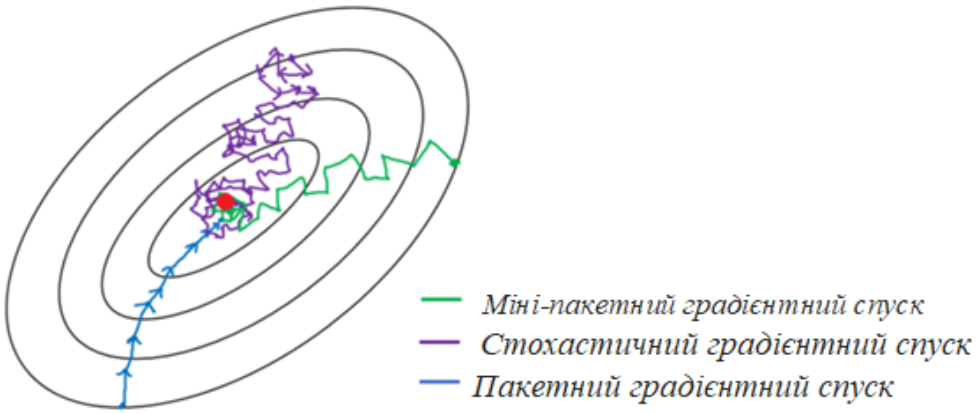
Пакетний градієнтний спуск

Пакетний градієнтний спуск використовує всю партію навчальних даних на кожному кроці. Він розраховує помилку для кожного запису і приймає середнє значення для визначення градієнта Перевага Batch Gradient Descent полягає в тому, що алгоритм ефективніший у обчислювальному відношенні та забезпечує стабільний шлях навчання, що полегшує збіжність. Тим не менш, пакетний градієнтний спуск займає більше часу, коли тренувальний набір великий.

Стохастичний градієнтний спуск

В іншому крайньому випадку Stochastic Gradient Descent просто вибирає один екземпляр з навчального набору на кожному кроці та оновлює градієнт лише на основі цього окремого запису. Перевага Stochastic Gradient Descent полягає в тому, що алгоритм набагато швидший ніж Batch Gradient Descent.

Міні-пакетний градієнтний спуск поєднує концепцію пакетного і стохастичного градієнтного спуску. На кожному етапі алгоритм обчислює градієнт на основі підмножини навчального набору замість повного набору даних або лише одного запису.



«дельта»-правило.

Дельта-правило – метод навчання персептрону на основі градієнтного спуску. Дельта-правило розвинулося з першого та другого правил Хебба. Його подальший розвиток спричинив створення алгоритму зворотного поширення помилки. Правило ґрунтується на зменшенні вихідної помилки персептрону.

Нехай $X = x_1, x_2, \dots, x_n$ - вектор вхідних сигналів, а $D = d_1, d_2, \dots, d_n$ - вектор сигналів, які повинні бути отримані від персептрону під впливом вхідного вектору. Тут n - число нейронів, що складають персептрон. Вхідні сигнали, надійшовши на входи персептрону, були зважені та підсумовані, в результаті чого отримано вектор $Y = y_1, y_2, \dots, y_n$ вихідних значень персептрону. Тоді можна визначити вектор помилок $E = e_1, e_2, \dots, e_n$ розмірність якого збігається з розмірністю вектору вихідних сигналів.

Компоненти вектору помилок визначаються як різниця між очікуваними та фактичними значеннями на вихідних нейронах персептрону:

$$E = D - Y.$$

За таких позначень формула для коригування $j-$ї ваги $i-$го:

$$w_j(t+1) = w_j(t) + e_i \cdot x_j$$

де $i, j \in \overline{1,n}$, n-кількість нейронів, t - номер поточної ітерації.

Підсумкова формула для коригування ваг:

$$w_j(t+1) = w_j(t) + \eta \cdot e_i \cdot x_j$$

де η коефіцієнт пропорційності (швидкістю навчання).

Алгоритм зворотного поширення помилки.

Алгоритм зворотного поширення - це ітеративний градієнтний алгоритм, який використовується з метою мінімізації середньоквадратичного відхилення поточного виходу багатошарового персептрона і бажаного виходу.

Він використовується для навчання багатошарових нейронних мереж з рослідовними зв'язками. Послідовність дій:

1.Ініціалізувати синаптичні ваги випадковими значеннями, близькими 0

2. Вибрати чергову навчальну пару (вхідні та вихідне значення) з навчальної множини; подати вхідний вектор на вхід мережі. При цьому нейрони послідовно від шару до шару функціонують за такими формулами:

- прихований шар $x_j = \sum_j w_{i,j} \cdot x_j^{\alpha}$ $y_j = \sum_j \sigma(x_j)$
- вихідний шар $x_k = \sum_j v_{i,j} \cdot y_j$ $y_k = \sum_k \sigma(x_k)$

де $\sigma(x)$– функція активації; x_j, x_k – входи; $w_{i,j}, v_{i,j}$ – ваги; y_j, y_k – виходи

3. Обчислити вихід мережі.

4. Обчислити різницю між виходом мережі та необхідним виходом (цільовим вектором навчальної пари). Функція помилки для цього вхідного набору обчислюється за такою формулою: $E = \frac{1}{2}\sum_{i=1}^{k}(y_i - y_i^{'})^2$

Ітераційне уточнення аргументу $v_{j,k}(t+1) = v_{j,k}(t) - h \cdot \frac{\partial E}{\partial v_{j,k}}$

де y_i – виходи зовнішнього шару, $y_i^{'}$ – реальні виходи

h – швидкість навчання

$\frac{\partial E}{\partial y_k} = (y_k - (y_k^{'})^{\alpha}) = \delta_k$ $\frac{\partial E}{\partial x_k} = \delta_k \cdot y_k(1 - y_k)$

$\frac{\partial E}{\partial v_{i.k}} = \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial x_k}\frac{\partial x_k}{\partial v_{i.k}} = \delta_k \cdot y_k(1 - y_k) \cdot y_j$

5. Підкоригувати ваги мережі для мінімізації помилки.

$$w_{i,j}(t+1) = w_{i,j}(t) - h \cdot \frac{\partial E}{\partial w_{i,i}}$$

6. Повторювати кроки з 2 по 5 для кожного вектору навчальної множини, поки помилка на всій множині не досягне прийнятного рівня, який задається критерієм зупинки навчання.

Для збільшення швидкості ітерацій розроблено велику кількість оптимізаторів градієнтного спуску, як то RMSProp, AdaDelta, Adam, Nadam тощо. На практиці найчастіше використовується алгоритм Левенберга-Марквардта.

Алгоритм Левенберга-Марквардта

Наприклад, якщо записати метод Ньютона для нашої задачі багатовимірної оптимізації, то отримаємо вираз:

$$w^{(t+1)} = w^{(t)} - \eta \cdot \left[H_i\left(w^{(t)}\right)\right]^{-1} \cdot \nabla L_i(w^{(t)})$$

Тут $\eta = 1$ (можна не ставити), а $H_i\left(w^{(t)}\right)$ - матриця Гессе (гесіан), розмірністю $n\ x\ n$:

$$H_i(w) =\cdot \nabla^2 L_i(w) = \begin{bmatrix} \frac{\partial^2 L_i}{\partial w_1^2} & \cdots & \frac{\partial^2 L_i}{\partial w_1 \cdot \partial w_n} \\ \cdots & \cdots & \cdots \\ \frac{\partial^2 L_i}{\partial w_n \cdot \partial w_1} & \cdots & \frac{\partial^2 L_i}{\partial w_n^2} \end{bmatrix}$$

Якщо обчислити саму матрицю якось можна, то на кожному кроці брати обернену матрицю в ознаковому просторі n, коли n = 100 000 і більше, обчислювально дуже затратно, а за розмірності 1 000 000 і більше – практично неможливо навіть на сучасних комп'ютерах. Тому було запропоновано чергову евристику (хитрість) – вважати матрицю Гессе діагональною, тобто обнулити всі змішані частинні похідні. Тоді для її обчислення досить взяти n других частинних похідних (по головній діагоналі) і ще n операцій поділу при обчисленні оберненої матриці, так як:

$$D = \begin{bmatrix} d_1 & 0 & 0 \\ 0 & \cdots & 0 \\ 0 & 0 & d_n \end{bmatrix} \Longrightarrow D^{-1} = \begin{bmatrix} 1/d_1 & 0 & 0 \\ 0 & \cdots & 0 \\ 0 & 0 & 1/d_n \end{bmatrix}$$

Знаходити обернену діагональну матрицю не становить великих проблем. Саме це і запропонували Левенберг та Марквардт – вважати гесіан діагональною матрицею. Тоді алгоритм оптимізації можна записати у вигляді:

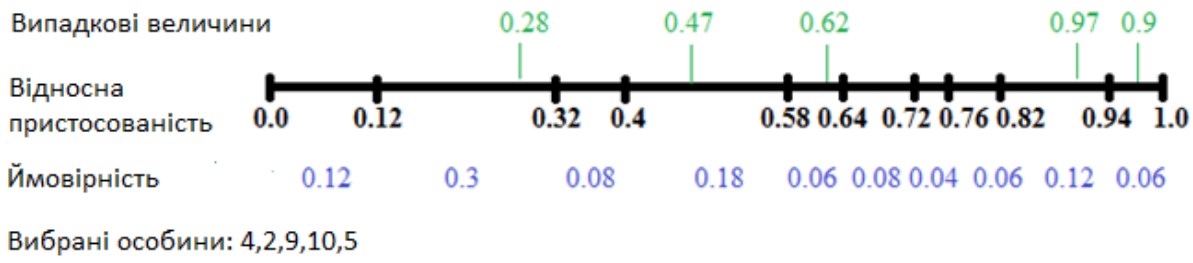
$$w_j = w_j - \eta \cdot \left[\frac{\partial^2 L_i(w)}{\partial w_j^2} + \mu\right]^{-1} \cdot \frac{\partial L_i(w)}{\partial w_j}$$

Тут - це значення, яке запобігає поділу на нуль і крім того, задає швидкість збіжності алгоритму, коли друга похідна обнуляється. Це відбувається у випадках лінійної зміни функції втрат (прискорення дорівнює нулю). Тут ми фактично слідує тільки за градієнтом, не звертаючи увагу на другі похідні.

Генетичний алгоритм

Опишемо докладніше принцип роботи генетичного алгоритму взаємодії з нейронними мережами.

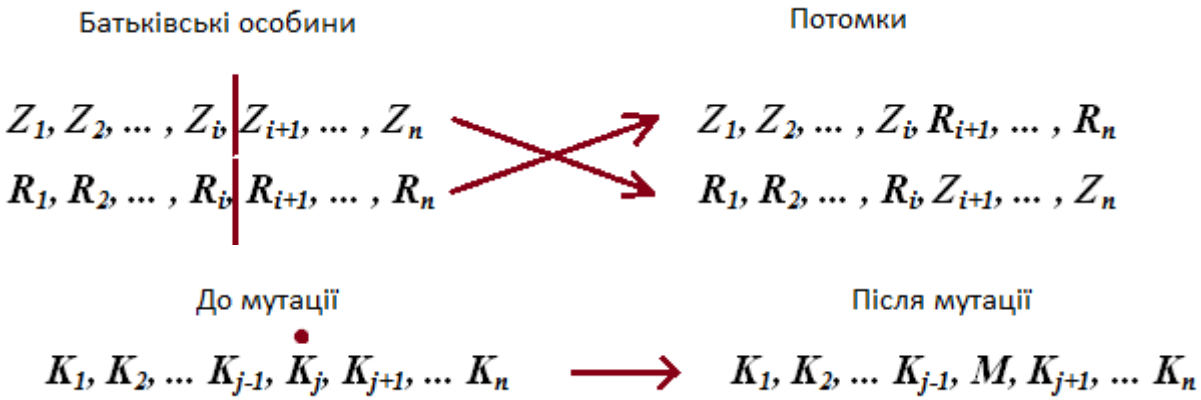
Відбір особин для схрещування здійснюється з використанням "рулеткового відбору". Даний вид відбір є імовірнісним методом випадкового вибору особин, де кожна особина має апріорну ймовірність бути обраною, що співвідноситься зі значенням її функції пристосованості. Кожній особини ставиться у відповідність відносна пристосованість - число з діапазону [0;1], що визначається ставленням значення цільової функції до суми значень цільових функцій всіх особин популяції. Після визначення відносин відрізок [0;1] розбивається на число відрізків, що дорівнює кількості особин у популяції так, що довжина кожного відрізка відповідає відносної пристосованості відповідної йому особини. Необхідну кількість разів запускається генератор неперервної випадкової величини, рівномірно розподіленої на інтервалі [0,1] (базовий випадкової величини). Особини для схрещування вибираються відповідно до відрізків, на які потрапили згенеровані величини..



Використовуємо для наступного покоління всіх особин, зберігаючи при цьому високі шанси за більш пристосованими. Це дозволяє періодично

розбавляти генофонд і завдяки цьому запобігати попадання до локального екстремуму.

Схрещування (кросовер) відбувається так - випадково визначається точка всередині хромосоми (набору параметрів), звана точкою розриву, у якій обидві хромосоми діляться на частини. Поєднавши першу частину однієї хромосоми з другою частиною іншої і навпаки, в результаті одержують дві нові хромосоми. На кожному циклі, для запобігання передчасній збіжності, з деякою низькою ймовірністю здійснюється мутація особин, яка полягає у зміні окремого гена-параметра особини на інший із загального набору параметрів.



Розглянемо задачу знаходження хромосоми, яка має максимальну кількість одиниць. Нехай хромосоми складаються з 12 генів, кожна популяція складається з 8 хромосом. Отже, найкраща хромосома складатиметься із 12 одиниць. Розглянемо розв'язання цієї задачі з використанням генетичного алгоритму.

Створення чи генерація початкової популяції. Для цього потрібно згенерувати 8 двійкових послідовностей, кожна завдовжки 12 біт. Їх можна отримати, наприклад, розігруючи базову випадкову величину 96 разів (при отриманні значення з інтервалу [0;0.5] приписується значення 1, у разі значення з інтервалу (0.5,1] - 0) або використанням будь-якого генератора двійкових випадкових чисел. популяцію, використавши будь-який із метод.

ch1 = [010111101010]; ch2 = [110000101101];

ch3 = [111000001001]; ch4 = [100010000001];

ch5 = [001101011000]; ch6 = [100010000101];

ch7 = [001000001010]; ch8 = [110011010110].

Оцінимо пристосованість хромосом у створеній популяції. Так як і в нашому простому прикладі ми шукаємо таку хромосому, яка містить максимальну кількість одиниць. Отже, функція приладдя має залежати від кількості одиниць у хромосомі. Позначатимемо функцію приналежності символом F. А за її значення брати суму генів хромосоми. У цьому випадку значення кожної хромосоми з отриманої популяції будуть наступні:

F(ch1) = 7; F(ch2) = 6;

F(ch3) = 5; F(ch4) = 3;

F(ch5) = 5; F(ch6) = 4;

F(ch7) = 3; F(ch8) = 7.

Хромосоми ch1 та ch8 мають найбільші значення функції пристосованості. Вони у цій популяції є основними претендентами вирішення завдання. Якщо умова зупинки алгоритму не виконується, то на наступному кроці проводиться відбір особин поточної популяції наступного схрещування.

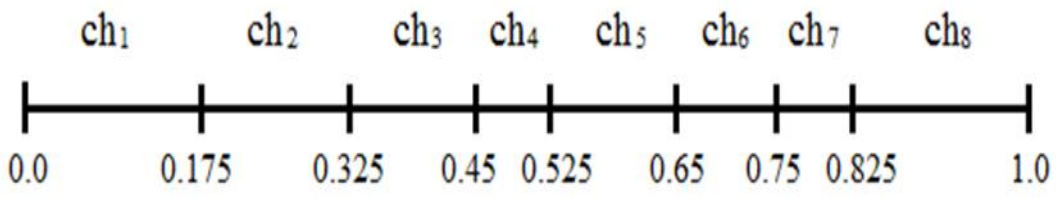
Селекція проводиться рулетковим методом. Для кожної з 8 хромосом вихідної популяції отримуємо сектори одиничного відрізка, виражені ставленням значення функції пристосованості окремої особини до суми значень функції пристосованості всіх особин (рисунок 6).

v(ch1) = 0.175; v(ch2) = 0.15;

v(ch3) = 0.125; v(ch4) = 0.075;

v(ch5) = 0.125; v(ch6) = 0.1;

v(ch7) = 0.075; v(ch8) = 0.175.



Розіграш за допомогою побудованого розбиття зводиться до випадкового вибору числа з інтервалу [0,1], що вказує відповідний сектор відрізка, тобто. на відповідну цьому відрізку конкретну хромосому.

Розіграємо 8 випадкових чисел:

0.79 0.44 0.9 0.74 0.04 0.86 0.48 0.23

Відповідно обрані хромосоми:

ch7 ch3 ch8 ch6 ch1 ch8 ch4 ch2

Як бачимо, хромосома ch8 була обрана двічі, а хромосома ch1 – один раз. Ці хромосоми мають значення функції приналежності. Однак, також вибрані хромосоми ch4 і ch7, що мають найменші значення функції приналежності. На основі всіх відібраних хромосом створюватиметься наступне покоління.

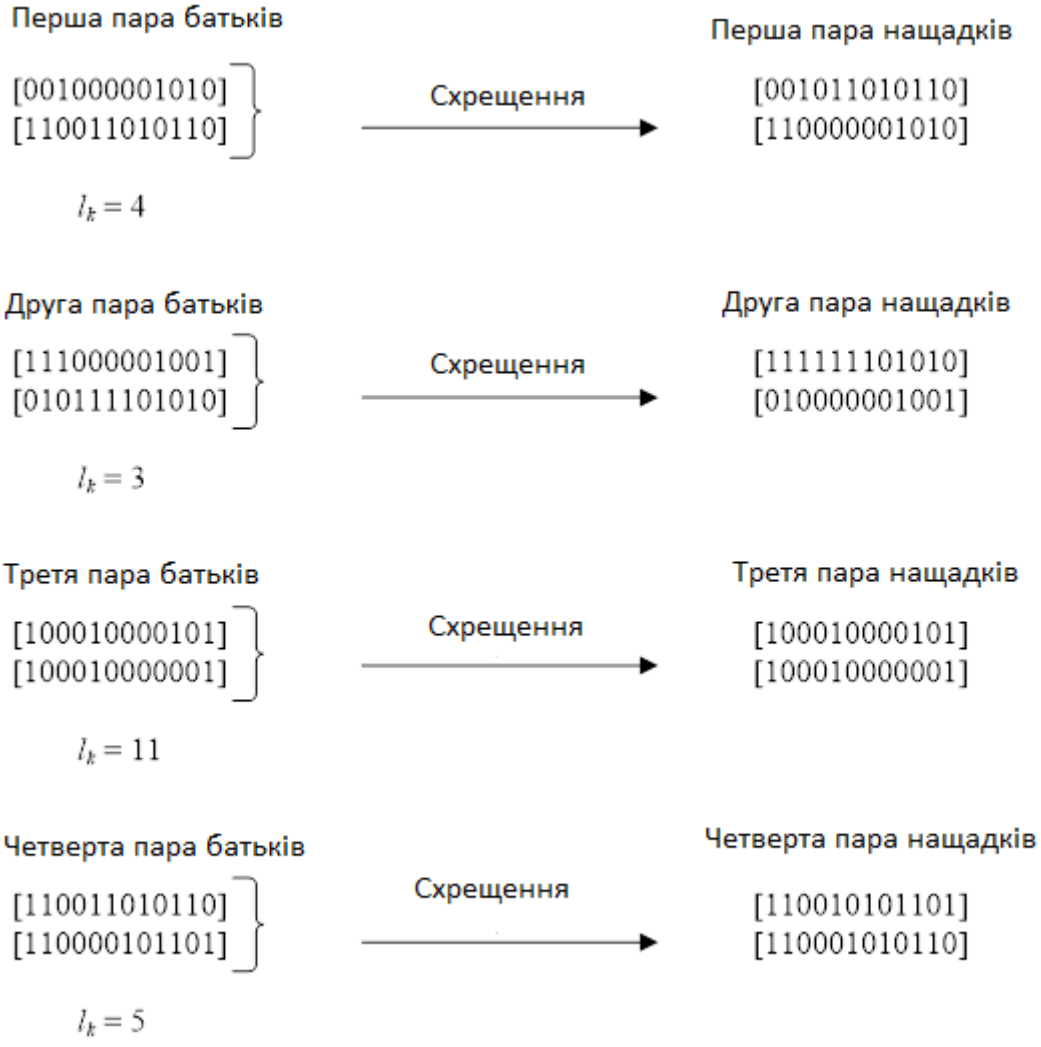
Застосування операторів схрещування. У прикладі не будемо піддавати відібрані для селекції хромосоми мутації, і всіх їх додамо в популяцію хромосом, для схрещування. Випадково сформуємо з цих хромосом чотири пари батьків:

ch7 и ch8 ch3 и ch1 ch6 и ch4 ch8 и ch2

Також випадково визначимо точки схрещування кожної з пар. Для першої пари lk = 4, для другої lk = 3, для третьої lk = 11, для четвертої lk = 5. Сам процес схрещування протікає так, як показано на рисунку 7. Після завершення роботи оператора схрещування отримуємо 4 пари нащадків.

Якби були об'єднані хромосоми ch8 з ch8 і ch7 з ch2 замість ch7 з ch8 і ch8 з ch2, а інші залишилися б у колишніх парах, то схрещування ch8 з ch8 дало б пару ідентичних батьківським хромосом за будь-якого разу точки схрещування. Зазначимо, що для хромосом, які мають найбільше значення функції корисності, така ситуація найімовірніша. Таким чином, чим більше у хромосоми значення функції пристосованості, тим більше шансів опинитися в наступній популяції.

Якби при відборі пар для схрещування були об'єднані хромосоми ch8 з ch8 і ch7 з ch2 замість ch7 з ch8 і ch8 з ch2, а інші залишилися б у колишніх парах без зміни, то схрещування ch8 з ch8 дало б пару ідентичних батьківським хромосом за будь-якого разу точки схрещування. Зазначимо, що для хромосом, які мають найбільше значення функції корисності, така ситуація найімовірніша. Таким чином, чим більше у хромосоми значення функції пристосованості, тим більше шансів опинитися в наступній популяції.



Якби при відборі пар для схрещування були об'єднані хромосоми ch8 з ch8 і ch7 з ch2 замість ch7 з ch8 і ch8 з ch2, а інші залишилися б у колишніх парах без зміни, то схрещування ch8 з ch8 дало б пару ідентичних батьківським хромосом за будь-якого разу точки схрещування. Зазначимо, що для хромосом, які мають найбільше значення функції корисності, така ситуація

найімовірніша. Таким чином, чим більше у хромосоми значення функції пристосованості, тим більше шансів опинитися в наступній популяції.

Після схрещування відібраних особин ми отримаємо (відповідно до рисунку 7) нову популяцію нащадків:

Ch_1 = [001011010110]; Ch_2 = [110000001010];

Ch_3 = [111111101010]; Ch_4 = [010000001001];

Ch_5 = [100010000101]; Ch_6 = [100010000001];

Ch_7 = [110010101101]; Ch_8 = [110001010110].

З метою відрізнити знову отримані хромосоми від хромосом попередньої популяції, починатимемо їх позначення з великої літери С.

Відповідно до генетичного алгоритму ми знову повертаємося до другого етапу - оцінки пристосованості хромосом із щойно сформованої популяції. Значення функцій адаптації хромосом нової популяції:

$F(Ch_1) = 6$; $F(Ch_2) = 4$;

$F(Ch_3) = 9$; $F(Ch_4) = 3$;

$F(Ch_5) = 4$; $F(Ch_6) = 3$;

$F(Ch_7) = 7$; $F(Ch_8) = 6$.

Видно, що у популяції нащадків середнє значення функції пристосованості вище, ніж у батьківської популяції. Зауважимо, що після схрещування була отримана хромосома Ch3 значенням функції пристосованості якої більше ніж у будь-якої хромосоми з батьківської популяції.

Але могло статися й протилежне, якби при випадковому виборі батьківських хромосом найпристосованіша не потрапила у вибірку. Незважаючи на це середня пристосованість нової популяції в окремих випадках виявляється нижчою за попередню, завдяки тому, що хромосоми з великими значеннями функції пристосованості мають більше шансів з'явитися в наступних поколіннях.

Завдання класифікації об'єктів

Розглянемо загальні принципи функціонування нейронних мереж з прикладу завдання класифікації.

Припустимо, ми хочемо створити автоматичну систему, яка розрізняє два типи об'єктів — A та B. Системи технічного зору дозволяють записати дані про об'єкти (ознаки об'єкта) у цифровому вигляді.

Припустимо, що ми характеризуємо об'єкт за допомогою набору ознак $x_1, x_2, \dots, x_n$. Ознаки можуть бути виражені за допомогою цілих чисел чи навіть булевських змінних, тобто є ознака — немає ознаки.

Сукупність ознак можна розглядати як вектор з n компонентами, або точку k-мірного Евклідова простору $X = (x_1, x_2, \dots, x_n)$. Тоді завдання класифікації зводиться до наступної математичної задачі: розділити дві множини точок A і B n-мірного евклідова простору деякою гіперповерхнею розмірності $n - 1$.

Зробимо деякі важливі коментарі. Вибір ознак для класифікації є виключно складним завданням, яке ми поки що не розглядаємо. Раніше це завдання вирішувалося вручну, останнім часом з'явилися ефективні методи автоматичного знаходження найефективніших ознак (так званий Deep Learning, про який окремо буде сказано нижче). Зазначимо, що вибір правильних ознак є важливим для успішності подальшої обробки системи ознак методами, які ми описуємо нижче.

Навчання нейронної мережі відбувається на наборі навчальних прикладів $X(1), X(2), \dots, X(P)$ для яких приналежність об'єкта до класу A чи класу B відома. Визначимо індикатор:

$$D(X) = \begin{cases} 1, & \text{якщо x} \in A \\ 0, & \text{якщо x} \in B \end{cases}$$

В результаті навчання, за «досвідом» будуємо нейромережу, яка проводить гіперповерхню.

Математично цей процес може бути описаний як пошук певної функції

$$y = F(X, W),$$

де W - набір параметрів нейронної мережі. Для нейромереж ці параметри, зокрема, задають силу зв'язку між нейронами та підбираються так, щоб помилка навчання (error training) була б мінімальною (якомога ближче до нуля). Як помилка навчання зазвичай розглядають функцію

$$E_{train}(W) = \sum_j |F(X(J), W) - D(X(J))|$$

де $X(J),\ J = \overline{1,n}$ беруться з навчальної множини.

Для перевірки ефективності навчання нейронної мережі беруть тестову множину об'єктів і обчислюють

$$E_{test}(W) = \sum_j |F(X(J), W) - D(X(J))|$$

де X(J) беруться з тестової множини.

Після того, як система навчена (що іноді вимагає великого процесорного часу), для будь-якого поданого на вхід системи об'єкта X вона автоматично вирішує, якого класу він належить.

Основні моделі теорії нейронних мереж

Спочатку перерахуємо основні моделі теорії нейронних мереж, виділивши окремо два різні класи мереж:

Мережі прямого розповсюдження

• Одношаровий персептрон.

• Багатошаровий персептрон.

• Мережі радіальних базових функцій (RBF).

• Машини опорних векторів (SVM).

Рекурентні мережі

• Рекурентні мережі.

• Атракторні нейронні мережі.

Огляд моделей

Одна з перших моделей нейронної мережі була запропонована Ф. Розенблаттом у 50-ті роки XX століття та отримала назву одношарового

персептрона. У цій моделі вхідний сигнал проходив лише один шар нейронів і після цього формувався вихідний сигнал за допомогою сигмоїдальної функції.

Надалі з'ясувалося, що можливості таких найпростіших систем обмежені. Було запропоновано узагальнення - модель багатошарового персептрона, відмінність якого від одношарового полягає у проходженні вхідного сигналу через кілька шарів.

Багатошарові персептрони можуть моделювати будь-яку систему "вхід-вихід", тобто будь-який чорний ящик. Однак вони мають ряд недоліків, зокрема, вибір числа шарів, нейронів та зв'язків між ними не такий простий.

Такі системи, як мережі радіальних базисних функцій і машини опорних векторів, можна як спеціальні модифікації багатошарових персептронів з подолання недоліків останніх.

Усі перелічені моделі характеризуються наявністю шарів, якими проходить сигнал. У рекурентних нейронних мережах неможливо виділити окремі шари. Сигнали можуть циркулювати по мережі у всіх напрямках, утворюючи складну просторово-тимчасову структуру (pattern). Такі мережі мають колосальні можливості. Так, наприклад, відомо, що вони можуть моделювати будь-яку динамічну систему.

Одношаровий персептрон

Це найпростіша моделлю нейронної мережі.

Одношаровий персептрон отримує на вхід сигнал, заданий вектором X і на виході видає число

$$y = \sigma(S),$$

Вхідний вектор $X = (x_1, x_2, \dots, x_n)$. складається з n компонент кожна з яких є чисельною характеристикою аналізованого нейронною мережею об'єкта.

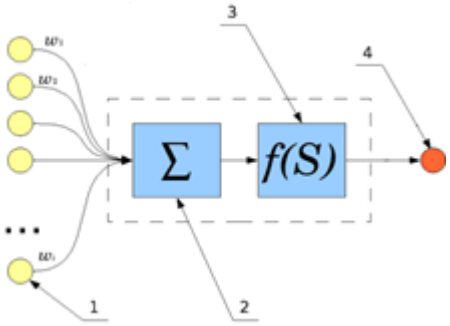
Через w_i , $i = \overline{1,n}$, і h позначені параметри персептрону *синаптичні ваги та поріг* (зсув) відповідно. Синаптичні ваги вказують на силу впливу

конкретної характеристики на вихідне значення. Значення цих параметрів може бути як позитивними, і негативними.

Зазначимо, що змінні x_i , $i = \overline{1,n}$ можуть бути булевими, тобто є приймати значення 0 або 1. Це означає, що об'єкт має дану ознаку, якщо $x_i = 1$, і не має, у разі, якщо $x_i = 0$.

У цьому випадку використовується функція Хевісайда або функція стрибка.

$$\sigma(S) = H(S) = \begin{cases} 1, & \text{якщо } S > 0 \\ 0, & \text{якщо } S \leq 0 \end{cases}$$



Модель одношарового персептрону

Наведено модель одношарового персептрона, цифрами 1-4 позначено послідовність дій під час роботи алгоритму.

Перевага персептрона полягає у його простоті. Вкажемо і його недолік - персептрон класифікує лише лінійно розділені об'єкти (тобто тільки такі множини векторів $X(t)$ між якими можна провести розділяючу ці множини гіперплощину).

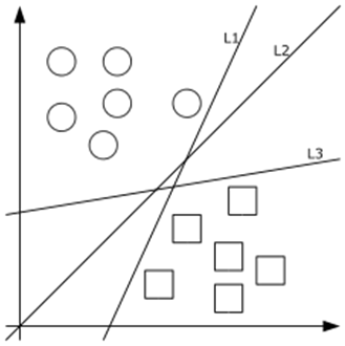
Геометрична інтерпретація

Одношаровий персептрон працює як класифікатор об'єктів, які математично можуть бути задані як елементи двох різних множин у багатовимірних лінійних просторах. Щоб зрозуміти, як це відбувається, розглянемо випадок, коли функція сигмоїди є функція стрибка $y = \sigma(S)$. Допустимо, потрібно класифікувати об'єкти у два класи об'єктів A і B, які не перетинаються. Для цього визначимо вихідні значення: покладемо $D = 1$, якщо об'єкт належить до класу A, та $D = 0$, якщо об'єкт належить до класу B:

$$D(X) = \begin{cases} 1, & \text{якщо x} \in A \\ 0, & \text{якщо x} \in B \end{cases}$$

Тоді, з визначення функції стрибка $\sigma = H(S)$ випливає, що об'єкт лежить у класі А за умови $S > 0$ і об'єкт лежить у класі В при $S < 0$. Отже, рівняння $S = 0$ може бути розглянуто як якийсь роздільник множини об'єктів різних класів.

З іншого боку, рівняння $S = 0$ рівносильно рівнянню $w1\ x1 + w2\ x2 + \cdots + wn\ xn - h = 0$ яке визначає гіперплощину в n-мірному лінійному просторі, що складається з векторів (рядків) $X = (x_1, x_2, \ldots, x_n)$.



Геометрична інтерпретація одношарового персептрону

Нагадаємо, що у двовимірному просторі гіперплощина – це пряма, а у тривимірному – звичайна площина.

Ілюстрація на рис. 4 демонструє вищесказане на прикладі гри квадратики (клас А) і кульки (клас В) у разі $n = 2$, тобто на площині. Прямі $L1, L2, L3$ - приклади «розділювачів» цих класів.

Таким чином, одношаровий персептрон є лінійним розділювачем.

Моделювання логічних функцій одношаровим персептроном

Одношаровий персептрон дозволяє нескладним чином реалізувати низку логічних функцій, таких як

• кон'юнкція,

• диз'юнкція,

• заперечення,

• демократичне голосування.

Результат кожної з цих функцій визначається значенням двох булевих змінних x_1 та x_2. Уявімо значення цих змінних як вектор вхідних значень $X = (x_1, x_2)$, а результат як значення функції стрибка, тобто $y = H(S) = w_1 x_1 + w_2 x_2 - h$

Пояснимо сказане з прикладу функції кон'юнкції — булевої функції, визначеної такою таблицею:

X_1	0	1	0	1
X_2	0	0	1	1
$X_1 \wedge X_2$	**0**	**0**	**0**	**1**

Наше завдання – представити цю логічну функцію за допомогою одношарового персептрона. Розглянемо вектор вхідних (x_1, x_2) значень як точки площини, а результат логічної операції — як форму точки: квадрат, у разі, коли результат операції істина $(y = 1)$, і коло, якщо результат операції покладено $(y = 0)$. Тоді наше завдання стає еквівалентним такому завданню класифікації: знайти пряму, що розділяє два класи об'єктів — квадрати та кола.

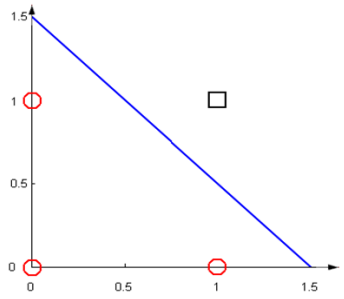


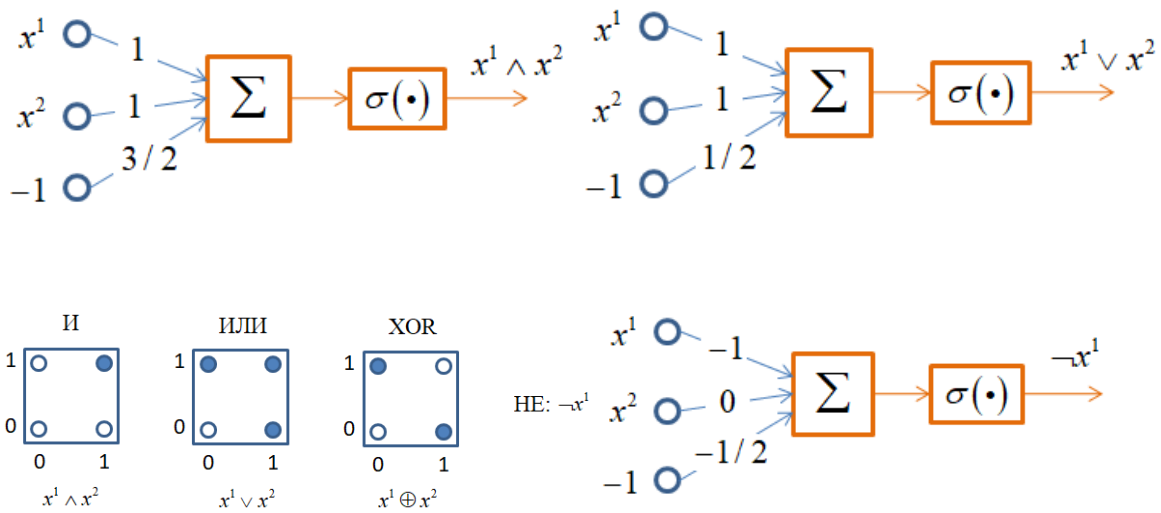
Рис. 5. Моделювання кон'юнкції одношаровим персептроном

Прямою, що розділяє ці множини об'єктів, є, наприклад, пряма

$$x_1 + x_2 - 1.5 = 0$$

Це означає, що синаптичні ваги $w_1 = 1, w_2 = 1$, поріг h=1.5. Ілюстрація вищенаведеного наведено на рис. 5.

На наступних рисунках наведено реалізацію нейромереж логічних функцій.



Приклад Мінського

Однак виявляється, що іноді квадрати та кола нероздільні.

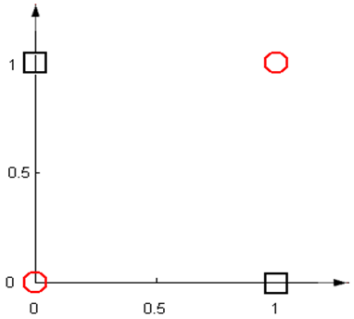


Рис. 6. Приклад Мінського

Марвін Мінський показав, що можливості персептрону обмежені, тому що він поділяє множину за допомогою гіперплощини. Насправді, є такі явно помітні класи об'єктів, котрим пряма як і роздільник не підходить. Наприклад, розглянемо логічну функцію «що виключає АБО» (XOR), що визначається формулою

$$XOR(x, y) = x + y \ (mod\ 2).$$

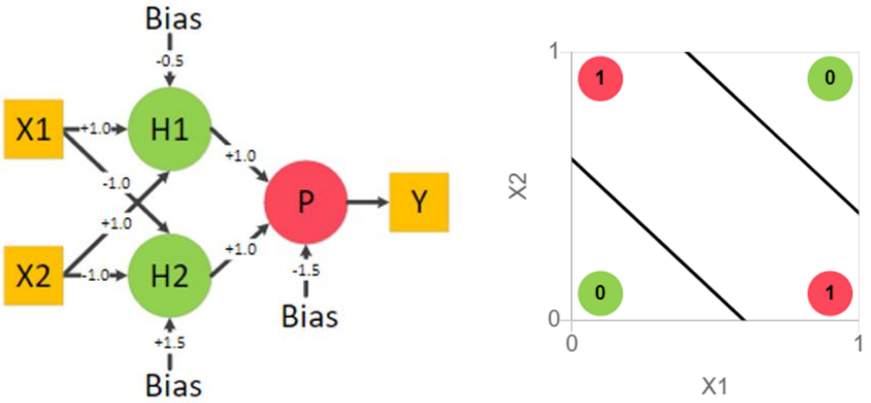
Для кращого розуміння наведемо таблицю значень функції XOR для різних змінних значень:

X_1	0	1	0	1
X_2	0	0	1	1
$XOR(X_1, X_2)$	**0**	**1**	**1**	**0**

Ця функція не реалізується одношаровим персептроном, тобто не існує такої прямої, яка розділила б ці два класи об'єктів. На рис. 6 проілюстровано

результат цієї логічної операції у вигляді квадратів, у разі, коли результат операції істина (y=1), і коло, якщо результат операції покладено (y=0). Очевидно, що ці дві множини (квадрати і кола) не розділені ніякою прямою.

Одношарова модель персептрона не може розв'язати функцію XOR, оскільки не можна намалювати одну пряму лінію для розділення та групування шаблонів виводу. Однак можна намалювати дві прямі лінії, щоб розділити та згрупувати вихідні візерунки. Багатошаровий персептрон, що містить додатковий шар прихованих нейронів, здатний вирішувати проблеми в тривимірному гіперпросторі, такі як проблема XOR. Тепер дані лінійно розділяються за допомогою 2-вимірної гіперплощини.



Використання одношарового персептрону

Існують ітераційні алгоритми, які знаходять параметри, якщо вихідні об'єкти лінійно розділюються.

Навчання нейронної мережі у завданнях класифікації відбувається на наборі навчальних прикладів $X(1), X(2), \dots, X(P)$ для яких приналежність об'єкта до класу A чи класу B відома. Щоб математично формалізувати цей факт, визначимо індикатор:

$$D(X) = \begin{cases} 1, & \text{якщо x} \in A \\ 0, & \text{якщо x} \in B \end{cases} \quad (1)$$

де кожен вектор X складається з n компонентів: $X = (x_1, x_2, \dots, x_n)$.

Завдання навчання персептрона полягає у знаходженні таких параметрів $w_1, w_2, \dots, w_n, h$, що на кожному навчальному прикладі персептрон видавав би правильну відповідь, тобто

$$y\big(X(i)\big) = D\big(X(i)\big),\ i = \overline{1,P} \qquad (2)$$

Інтуїтивно здається очевидним, що якщо персептрон навчений на великій кількості коректно підібраних прикладів, і рівність (2) виконується для багатьох $X(i)$, то надалі персептрон буде з близькою до одиниці ймовірністю може проводити правильну класифікацію для інших прикладів. Цей інтуїтивно очевидний факт був математично доведений (за деяких припущень).

На практиці, проте, оцінки з теорії Вапника–Червоненскиса іноді дуже зручні, особливо у складних моделей нейронних мереж. Тому практично, щоб оцінити помилку класифікації, часто надходять таким чином: множину навчальних прикладів розбивають на два випадково вибрані підмножини, при цьому навчання йде на одній множині, а перевірка навченого персептрона - на іншій.

Розглянемо детальніше алгоритм навчання персептрону

Крок 1. Ініціалізація синаптичних ваг та зміщення.

Значення всіх синаптичних ваг моделі вважають рівними нулю: $w_\text{i}, i = \overline{1,n}$; зміщення нейрона h встановлюють рівним деякому малому випадковому числу. Нижче, з міркувань зручності викладу та проведення операцій користуватимемося позначенням $w_0 = -h$. Позначимо через $w_\text{i}(t), i = \overline{1,n}$ вагу зв'язку від і-го вхідного елемента на момент часу t.

Крок 2. Подання векторів нового вхідного та очікуваного вихідного сигналів.

Вхідний сигнал $\text{X} = (\text{x}_1, \text{x}_2, \dots, \text{x}_\text{n})$ пред'являється нейрону разом з бажаним вихідним сигналом D.

Крок 3. Адаптація (налаштування) значень синаптичних ваг.

Оцінка вихідного сигналу нейрона. Переналаштування (адаптація) синаптичних вагів проводиться за такою формулою:

$$w_\text{i}(t+1) = w_\text{i}(t) + \big(rD(t) - y(t)\big)x_i(t), \qquad i = \overline{0,n-1}$$

де під $D(t)$ мається на увазі індикатор, визначений рівністю (1), а r - параметр навчання, що приймає значення менші 1.

Описаний вище алгоритм є алгоритмом градієнтного спуску, який шукає параметри, щоб мінімізувати помилку. Алгоритм ітеративний. Формула ітерацій виводиться в такий спосіб. Введемо ризик

$$R(t) = \sum_{t=1}^{T} (D(t) - F(X(t), w(t)))^2$$

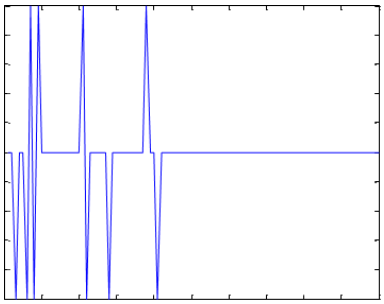
де підсумовування йде за кількістю дослідів (t - номер досліду), при цьому задано максимальну кількість дослідів - T.

Підставимо замість F формулу для персептрона, обчислимо градієнт w. В результаті ми отримаємо вказану вище формулу переналаштування ваги.

Ілюстрація роботи алгоритму

У процесі навчання обчислюється помилка

$$\delta(t) = D(t) - y(t)$$



Побудуємо графік, який показує, як змінюється ця помилка під час навчання мережі та адаптації ваг. На ньому добре видно, що, починаючи з деякого кроку, величина $\delta(t) = 0$. Це означає, що персептрон навчений.

Побудова нейронної мережі для операції кон'юнкції у пакеті Matlab

Побудуємо нейронну мережу для логічної операції кон'юнкції за допомогою пакету Matlab. Таблиця цієї логічної операції наведена вище. У цій реалізації масив входів P задає значення $X(1), X(2)$ для всіх навчальних прикладів і є перші два рядки цієї таблиці. Цільовий вектор T містить значення індикаторної функції $D(x)$ 4 прикладу і є останнім рядком цієї таблиці. Наведений нижче скрипт виконує побудову нейронної мережі для операції кон'юнкції та для зручності містить докладні коментарі.

```
% апроксимація кон'юнкції одношаровим персептроном
% побудова функції
y = H (w_1 * X_1 + w_2 * X_2 + w_0)
% (P,T) задають таблицю істинності
%задамо вектор входів
P = [0101; 0011];
%задамо вектор виходів
T = [0001];
% вбудована функція побудови мережі за допомогою персептрона,
% за умовчанням використано модель 'hardlim'
net = perceptron;
% визначимо максимальну кількість ітерацій на навчання мережі
net.trainParam.epochs = 20;
% навчання мережі
% у процесі навчання відбувається корекція терезів w
net = train(net,P,T);
% симуляція
% обчислення значень апроксимуючої функції у точках P
y = sim(net, P)
% синаптичні ваги після навчання мережі
A = net.IW; celldisp(A)
% коефіцієнт зсуву b після навчання мережі
w_0 = net.b
```

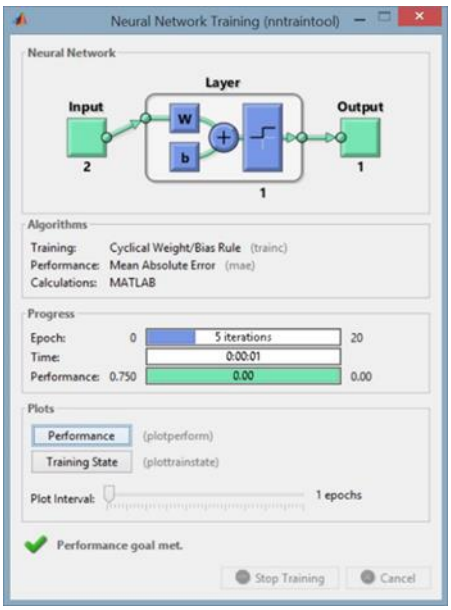


Рис. 8. Нейронна мережа, що реалізує коньюнкцію

У процесі роботи скрипта Matlab відображає поточну інформацію про модель та параметри навчання у спеціальному вікні Neural Network Training (рис. 8). Зокрема, у цьому вікні показано структуру нейронної мережі — кількість вхідних нейронів (2), кількість рівнів персептрону (1), число вихідних нейронів (1), кількість ітерацій під час навчання мережі (у разі — 5), час навчання мережі.

У той же час згідно з інструкціями скрипта у вікні команд буде виведена інформація за результатами симуляції (вектор y) та значення вагових коефіцієнтів ($w_1 = 1, w_2 = 2$) та коефіцієнта зсуву ($w_0 = -3$):

Y=0 0 0 1

A{1} = 1 2

w_0 = [-3]

Таким чином, отримана модель персептрону, що розділяє класи об'єктів та визначається рівнянням

$$x_1 + 2x_2 - 3 = 0$$

що геометрично означатиме розбиття двох множин заданої прямої (рис. 9).

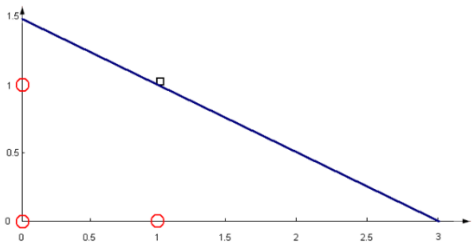


Рис. 9. Моделювання кон'юнкції одношаровим персептроном у Matlab

Завдання про моделювання чорної скриньки у загальній постановці

Нехай відомий реальний вихід пристрою $D(t)$ на вході $X(t)$ де $X(t)$ — вектор з компонентами $(x_1, x_2, \dots, x_n)$, t — номер проведеного досвіду $t = \overline{1,T}$ (максимальна кількість дослідів T заздалегідь визначено).

Необхідно знайти параметри моделі $w = (w_0, w_1, \dots, w_n)$ такі, що вихід моделі $F(X, w)$ та реальний вихід D(t) якомога ближче в середньоквадратичному сенсі, тобто

$$R(t) = \sum_{t=1}^{T} (D(t) - F(X(t), w(t)))^2 \to min$$

Функцію $R(t)$ називають **емпіричним ризиком**.

Побудова нейронної мережі для класифікації множини точок площини в пакеті Matlab

Як інший приклад застосування одношарового персептрона наведемо приклад класифікації точок площини на два класи A і B - точки, що знаходяться всередині еліпса, лежать у класі A і точки, що знаходяться поза еліпсом, лежать у класі B. Еліпс визначено загальним рівнянням кривої другого порядку виду

$$ax^2 + bxy + cy^2 + dx + ey = 1.$$

Нижче наведено скрипт, що реалізує цю класифікацію для еліпса

$$1.5\,x^2 + 2\,xy + 1.1y^2\ - 2x - 3\,y = 1.$$

У цьому прикладі застосовується одна з найважливіших ідей теорії нейронних мереж. А саме, у вихідному просторі двох змінних одношаровий персептрон не може розділити зовнішність і нутрощі еліпса. Але ми можемо зробити нелінійне перетворення даних, відобразивши їх у простір вищої розмірності. Тут застосовується так звана теорема Ковера, яка стверджує, що після такого відображення дані можуть стати лінійно роздільними.

Застосуємо в нашому випадку нелінійне відображення точки площини (x, y) в простір розмірності 5 наступним чином:

$$(x, y) \to (ax^2, bxy, cy^2, dx, ey).$$

% побудова одношарового персептрону для класифікації

% точок площини відповідно до заданого еліпса

% (всередині та поза еліпсом (ax^2 + bxy + cy^2 + dx + ey = 1))

% Результат роботи нейронної мережі:

```
% вектор значень t:
% 0 - точка поза еліпсом, 1 - точка всередині еліпса
N = 300; % кількість точок для навчання
% Коефіцієнти заданого еліпса a = 1.5; b = 2; c = 1.1; d = -2; e = -3;
% збудуємо графік вихідного еліпса syms X Y;
f = a * X. ^ 2 + b * X. * Y + c * Y. ^ 2 + d * X + e * Y-1; ellips = ezplot (f); set
(ellips, 'Color', 'b'); hold on
% згенеруємо вихідні дані для навчання:
% точки площини (x, y), чиї координати мають
% нормальний розподіл на інтервалах [-2a;2a] та [-2c;2c] x =
randn(1,N)*(2*a) - d/(2*a); y = randn (1, N) * (2 * c) - e / (2 * c);
% визначимо відображення
% (x1,x2)->(a*x1^2,b*x2^2,c*x1*x2,d*x1,e*x2)
% матриця входів
P = [a * x. ^ 2; b * x. * y; c*y.^2; d*x; e*y];
% сформуємо вектор виходів T, приписуючи значення 0 та 1
% для точок, що лежать поза та всередині еліпса відповідно T = ones(1,N);
k = find (sum (P)> = 1); T(k) = 0;
% функція побудови мережі за допомогою персептрона,
% за замовчуванням використано модель 'hardlim'
net = newp(P,T);
% визначимо максимальну кількість ітерацій на навчання мережі
net.trainParam.epochs = 50;
% функція навчання мережі; у своїй відбувається корекція терезів W net =
train(net,P,T);
% симуляція
% обчислення значень апроксимуючої функції у точках P t = sim(net, P)
% синаптичні ваги після навчання мережі A = net.IW;
celldisp(A)
% коефіцієнт зсуву b після навчання мережі b = net.b
```

% геометричні побудови

% розпізнавання точок площини після навчання мережі k = find (t = = 0); % індекси точок поза еліпсом

x_out = x (k); y_out = y(k);

*% побудова точок поза еліпсом plot(x_out, y_out, '*r');*

clear k;

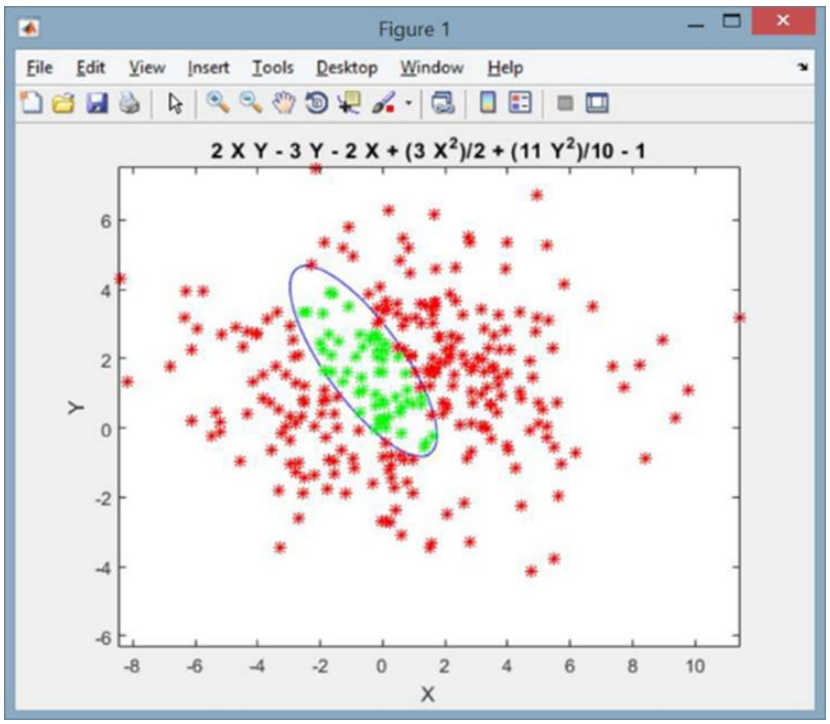
k = find (t = = 1); % індекси точок усередині еліпса x_out = x (k); y_out = y(k);

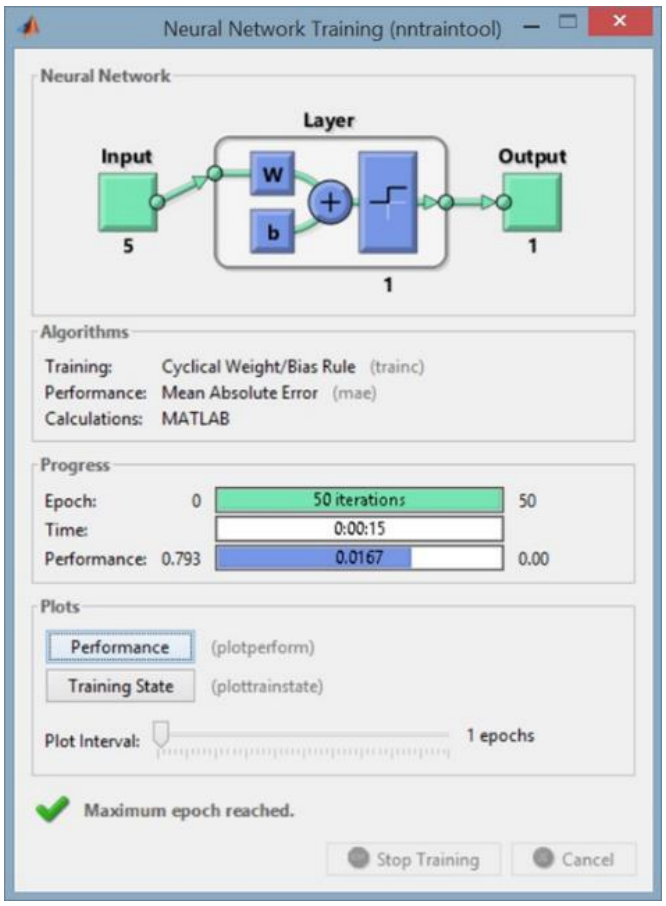
*% побудова точок усередині еліпса plot(x_out, y_out, '*g')*

У процесі роботи скрипта Matlab відображає поточну інформацію про модель та параметри навчання у спеціальному вікні Neural Network Training (рис. 10). Зокрема в цьому вікні показана структура нейронної мережі - число вхідних нейронів (50), число шарів персептрону (1) і моделі шарів, число вихідних нейронів (1), число ітерацій при навчанні мережі (у нашому випадку - 50), час навчання мережі (у нашому випадку – 15 с).

У вікні команд будуть виведені значення вагових коефіцієнтів (w1=-138.9032, w2=-164.07, w3=-179.1546, w4=- 153.6901, w5=-188.0759) та коефіцієнта зсуву (w0=151):

A {1} =-138.9032 -164.0700 -179.1546 -153.6901 -188.0759 b = [151]





Класифікація точок одношаровим персептроном

Таким чином, побудована модель має наступний вигляд

$$y = H(S) = H(w_1\,x_1 + w_2\,x_2 + \cdots + w_5\,x_5 + \ w_0) = H(-138{,}9\,x_1 - 164{,}07\,x_2 - 179{,}15\,x_3 - 153{,}69\,x_4 - 181{,}07\,x_5 + +151)$$

Результат класифікації множини точок після навчання персептрона представлений графічно на рис. 11; точки «начинки» та «зовнішності» еліпса зображені зеленим і червоним кольором відповідно.

Фундаментальна теорема про роздільність класів

Для будь-якої неперервної функції $F(x) = f(x_1, x_2, \dots, x_n)$ визначеної на обмеженій замкнутій області, та кожного $\varepsilon > 0$ існують коефіцієнти $w = w_0, w_1, w_2, \dots, w_m)$ і $V_k = (V_{1k}, V_{2k}, \dots, V_{nk}), h_k,\ k = \overline{1, m}$ такі, що

$$\left| f(X) - \left(\sum_{k=1}^{m} w_k \cdot \sigma(V_{1k}x_1 + V_{2k}x_2 + \dots + V_{nk}x_n - h_k) + w_0 \right) \right| < \varepsilon$$

Зауважимо, що вираз

$$\sum_{k=1}^{m} w_k \cdot \sigma(V_{1k}x_1 + V_{2k}x_2 + \ldots + V_{nk}x_n - h_k) + w_0$$

описує персептрон з одним прихованим шаром. Таким чином, теорема стверджує, що одного прихованого шару достатньо, щоб апроксимувати все (тобто будь-яку неперервну функцію, визначену на обмеженій множині).

У доведенні теореми одна функція векторного аргументу зі скалярними значеннями наближалася за допомогою двошарового (вхідний шар та прихований шар) персептрона з одним виходом. Векторну функцію можна апроксимувати за допомогою декількох паралельних персептронів або навіть за допомогою одного персептрону (з'єднавши паралельні разом).

Навчання багатошарового персептрона відбувається за алгоритмом **градієнтного спуску**, аналогічним до одношарового. Один із знаменитих варіантів цього алгоритму отримав назву метод **зворотного розповсюдження помилки** (back propagation error).

Мережі радіальних базисних функцій.

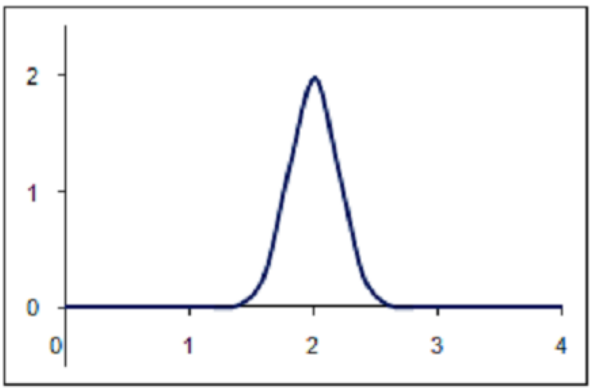
Введемо так звані радіальні базисні функції (RBF). Радіальні базисні функції є спеціальним класом функцій, характерною властивістю яких є наявність центральної точки та монотонне спадання з віддаленням від цієї центральної точки:

$$g(X) = G(\lambda|X - t|)$$

де $X = (x_1, x_2, \ldots, x_n)$, t — деяка точка, λ — параметр локалізації, що відповідає за «крутість» кривої в деякій околиці центру кривої t. Якщо значення параметра λ досить велике (більше 3), то графік цієї функції нагадує вузький «пік», локалізований біля центру t .

Такі властивості має, наприклад, функція Гауса

$$g(X) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{(X-t)^2}{2\sigma^2}}$$



Функція Гауса з центром у точці $t = 2$ *та параметром* $\lambda = 12.5$

Система радіальних базисних функцій

$$g_j(X) = G(\lambda \cdot \|X - t_j\|)$$

з центрами у точках t_j та загальним параметром λ дозволяє апроксимувати будь-яку функцію $f(X)$ за допомогою лінійної комбінації

$$f(X) = \sum_{j=1}^{M} w_j g_j(X) = F(X, w, M)$$

Останній вираз визначає мережу радіально-базових функцій (RBF). Такі мережі, як і багатошарові персептрони, є універсальними роздільниками і апроксиматорами, що й затверджується в наступній теоремі:

Функції $F(X, w, M)$ є універсальними апроксиматорами в компактних областях $D, X \in D$.

Метод опорних векторів.

Метод опорних векторів у завданнях класифікації запропоновано В. Вапником та його колегами. Він та набув широкого поширення на початку 2000 р.р. Розглянемо завдання класифікації на два класи, що не перетинаються, в якій об'єкти описуються n-мірними дійсними векторами, вихід

$$D = \{-1; +1\}.$$

Значення 1 відповідає належності одному класу, а -1 - іншому.

Будуватимемо лінійний пороговий класифікатор:

$$y(X) = sign(w \cdot X + w_0)$$

де $X = (x_1, x_2, \dots, x_n)$ - ознаковий опис об'єкта X; вектор $w = (w_1, w_2, \dots, w_m)$ і скалярний поріг w_0 є параметрами алгоритму;

Нагадаємо, що рівняння $w \cdot X + w_0$ визначає гіперплощину, що розділяє класи в n-мірному просторі.

Критерій та методи налаштування параметрів у SVM радикально відрізняються від персептронних (градієнтних) методів навчання. Використовуються три основні ідеї:

• оптимальна роздільна гіперплощина;

• можливий неточний поділ, але за помилки у поділі сплачується штраф (математичний, звичайно);

• нелінійне відображення даних.

Оптимальна роздільна гіперплощина

Враховується факт, що при нульовому функціоналі похибок (якщо існують такі параметри w , w_0 розділяюча гіперплощина не одна. Тоді з множини розділяючих гіперплощин виділяють оптимальну – ту яка знаходиться від класів на максимальній відстані.

Це приводить до завдання квадратичного програмування, оскільки середнє видалення роздільної площини від точок, що розділяються, пропорційно квадрату довжини вектора невідомих ваг W. Завдання квадратичного програмування, взагалі кажучи, важке (значно складніше, ніж лінійне програмування), але у багатьох практичних випадках успішно вирішується.

Штраф за помилки у розподілі

Друга важлива ідея — можна застосовувати метод навіть тоді, коли множини лінійно нероздільні. У цьому випадку пропонується запровадити поняття штрафу та за помилки у поділі сплачувати цей штраф. Це класична ідея оптимізації, що йде ще від Лагранжа.

Нехай X^i - і-й приклад. Тоді система рівнянь методу SVM набуває вигляду:

$$\|w\| + C \sum_i T\xi_i \to min$$

$$y_i(wX^i + w_0) \geq 1 - \xi_i$$

У цій системі невідомими є ξ_i та коефіцієнти w.

Невід'ємні змінні ξ_i описують штрафні санкції за те, що приклад X^i неправильно класифіковано; тут wX^i скалярний добуток.

Якщо $\xi_i = 0$, то ми отримуємо звичайне завдання поділу для персептрона, де шукаємо оптимальну гіперплощину:

$$\|w\| \to \min \qquad\qquad y_i\left(wX^i + w_0\right) \geq 1$$

Остання умова означає, що вихід персептрону $wX^i + w_0$ та реальний вихід y_i мають однаковий знак.

Це складне завдання квадратичного програмування. За наявності штрафів ще є параметр C, який треба підбирати. Для цього, взагалі кажучи, непростого завдання, розроблено методи її вирішення.

Зауважимо, що багато практично важливих завдань, включаючи квадратичне програмування, є важкими, проте, на практиці вони успішно вирішуються спеціальними методами та за допомогою евристичних ідей.

Нелінійне відображення даних. Ядра (kernels).

Нелінійне відображення на інший простір з іншим скалярним добутком може перетворювати лінійно нероздільні множини в лінійно розділені: $x \to \Psi(x)$.

Взагалі кажучи, якщо розмірність Ψ вища, ніж розмірність x, то ми можемо отримати лінійний поділ образів гіперплощиною у просторі Ψ.

Як це відбувається?

Нехай є дві множини A і B. Вони можуть бути нероздільні гіперплощиною. Розглянемо їх образи $\Psi(A)$ і $\Psi(B)$ внаслідок дії деякого нелінійного відображення $x \to \Psi(x)$.

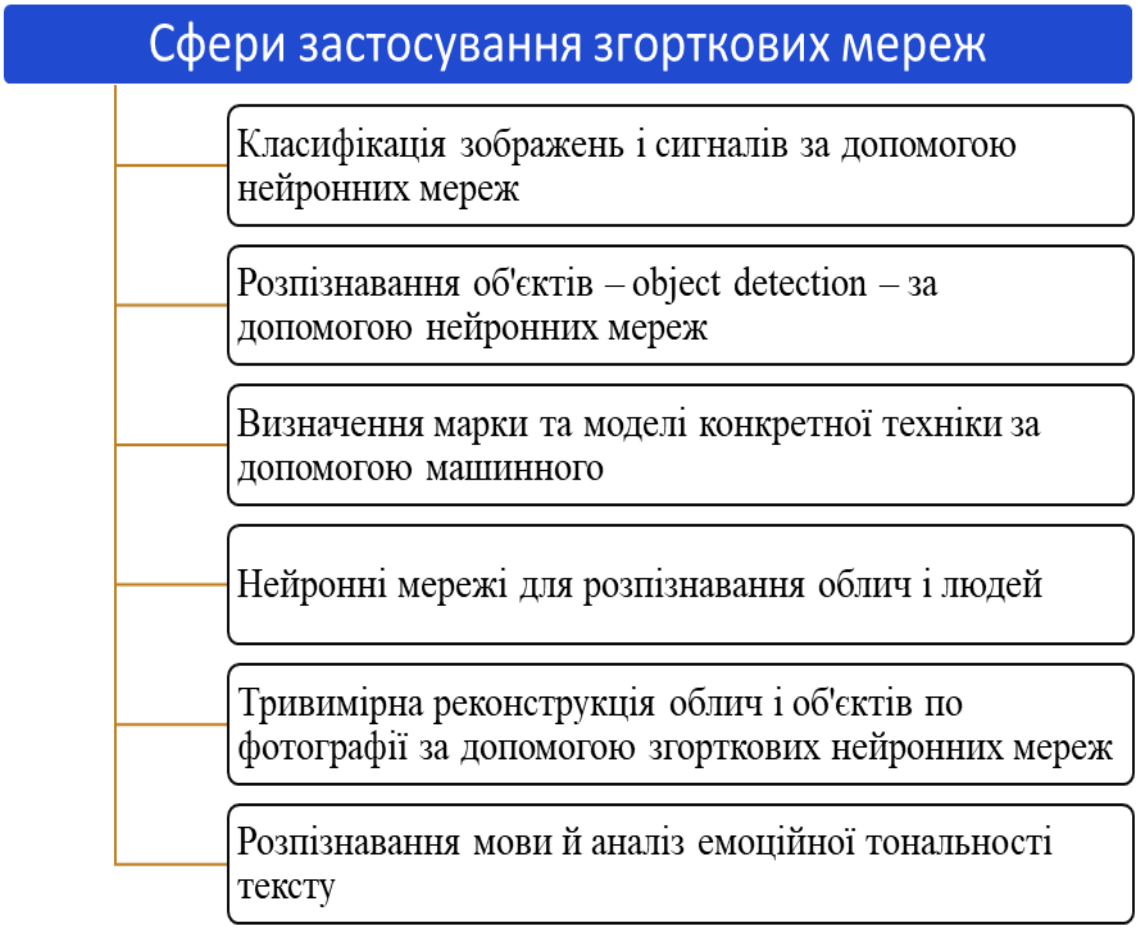
Як приклад розглянемо поділ начинки A і зовнішності B еліпса, визначеного рівнянням.

$$2x^2 + 3y^2 - xy = 1$$

Множини A і B лінійно нероздільні у вихідній множині (точок площини). Однак, вони розділяються у просторі Ψ, до якого можна перейти за допомогою відображення $\Psi = (\Psi_1, \Psi_2, \Psi_3) = (x^2, y^2, xy)$.

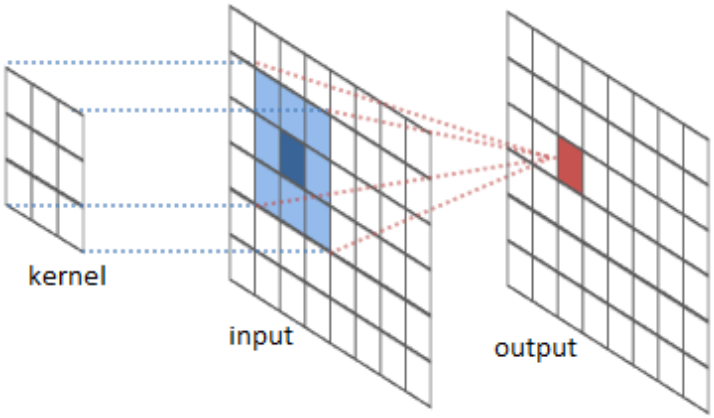
Рівняння площини у просторі Ψ для класифікатора набуває вигляду $2\Psi_1 + 3\Psi_2 - \Psi_3 = 1$.

Згорткові нейронні мережі



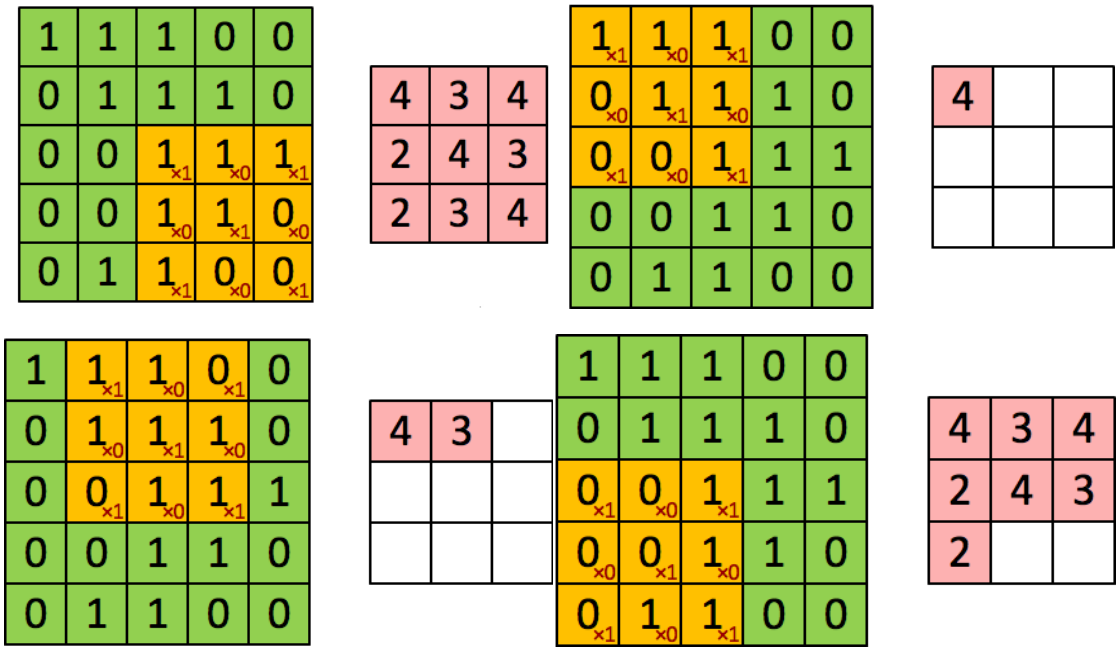
Згорткова нейронна мережа (CNN) — це тип глибокої нейронної мережі, яка довела свою ефективність у таких завданнях комп'ютерного зору, як класифікація зображень, виявлення об'єктів, локалізація об'єктів тощо. ЇЇ характеризують: згортковий рівень, рівень об'єднання та повно зв'язаний рівень.

Шар згортки



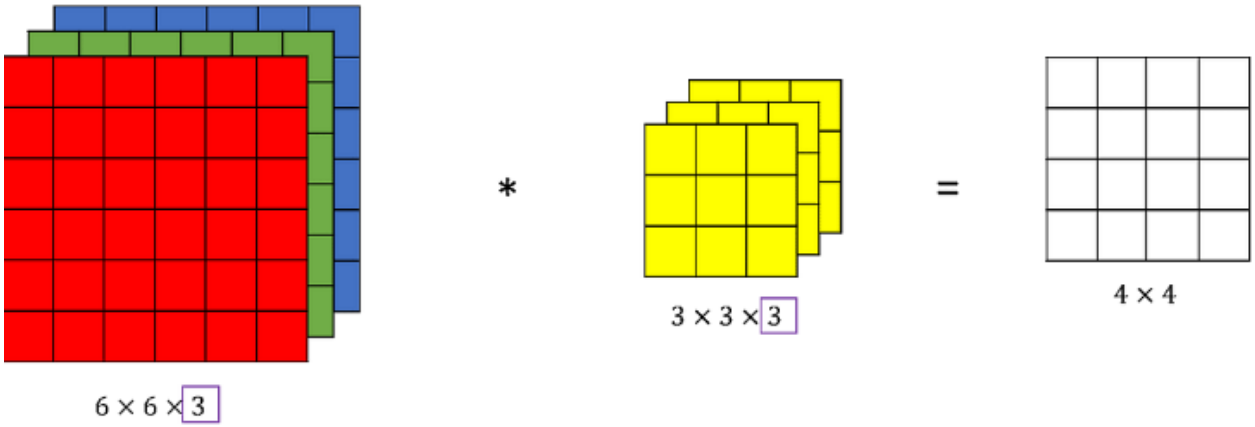
Шар згортки перетворює вхідне зображення, щоб витягти з нього особливості. У цьому перетворенні зображення згортається за допомогою ядра (або фільтра).

Ядро — це маленька матриця, висота й ширина якої менші за зображення, яке потрібно згорнути. Вона також відома як матриця згортки або маска згортки. Це ядро ковзає по висоті та ширині вхідного зображення, а скалярний добуток ядра та зображення обчислюється в кожній просторовій позиції.

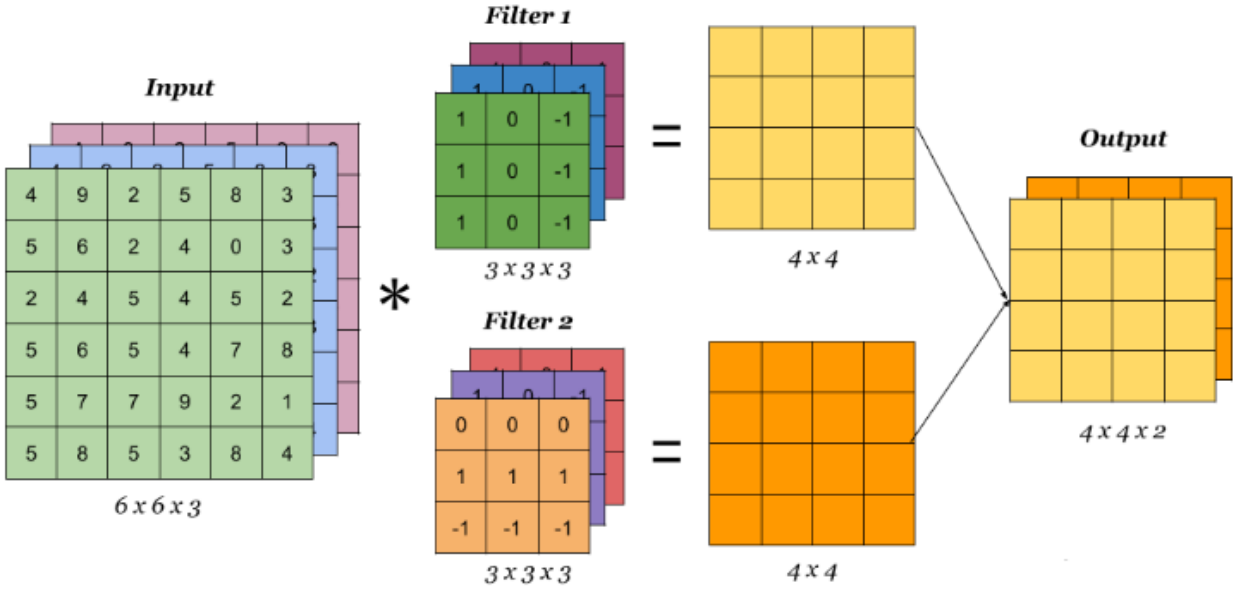


Довжина, на яку ковзає ядро, називається довжиною кроку. На зображенні нижче вхідне зображення має розмір 5X5, ядро — 3X3, а довжина кроку — 1. Вихідне зображення також називається згорнутою функцією.

Під час згортання кольорового зображення (RGB-зображення) з 3 каналами, каналів фільтрів також має бути 3. Іншими словами, у згортці кількість каналів у ядрі має бути такою ж, як кількість каналів у вхідному зображенні.

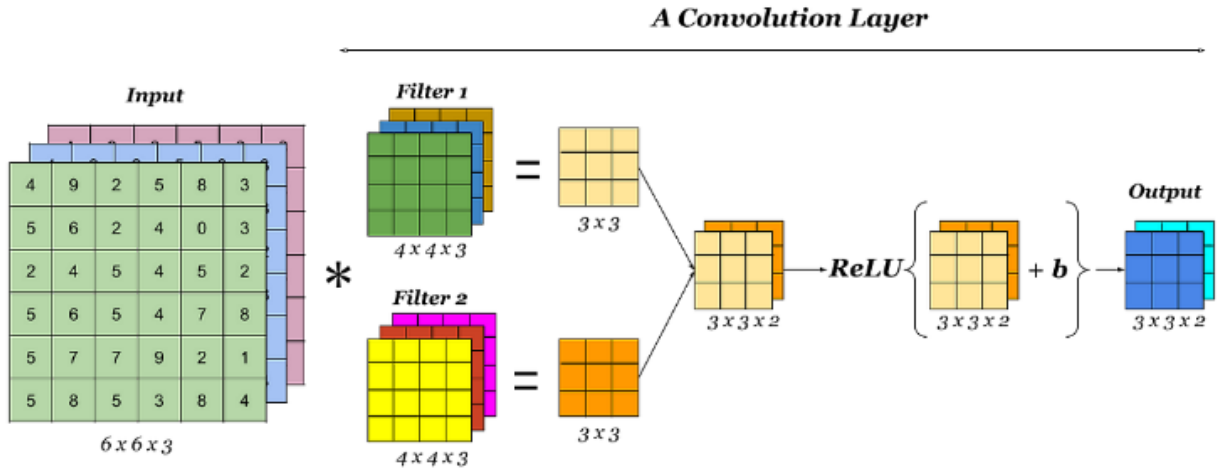


Якщо ми хочемо отримати більше однієї функції із зображення за допомогою згортки, ми можемо використати кілька ядер замість одного. У цьому випадку розмір усіх ядер повинен бути однаковим. Згорнуті характеристики вхідного зображення та вихідного зображення складаються одна за одною для створення виходу таким чином, щоб кількість каналів дорівнювала кількості використаних фільтрів.



Функція активації є останнім компонентом згорткового шару для збільшення нелінійності на виході. Зазвичай функція ReLu або функція Tanh використовується як функція активації в шарі згортки. Ось зображення

простого шару згортки, де вхідне зображення 6X6X3 згортається з двома ядрами розміром 4X4X3, щоб отримати згорнуту функцію розміром 3X3X2, до якої застосовується функція активації, щоб отримати результат.

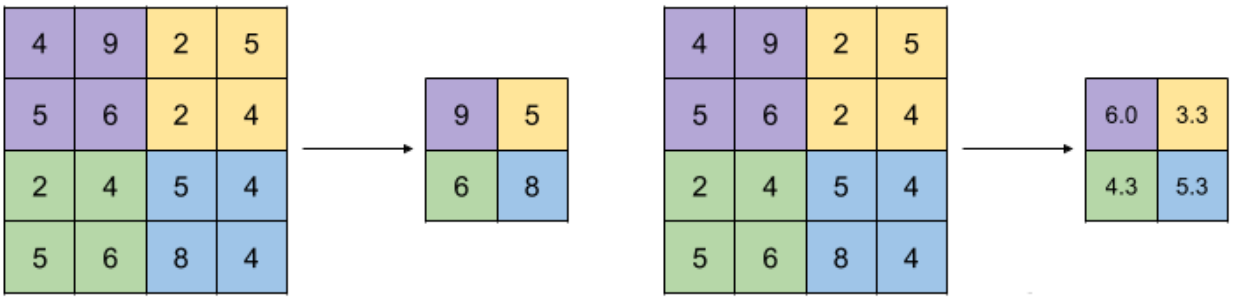


Шар об'єднання

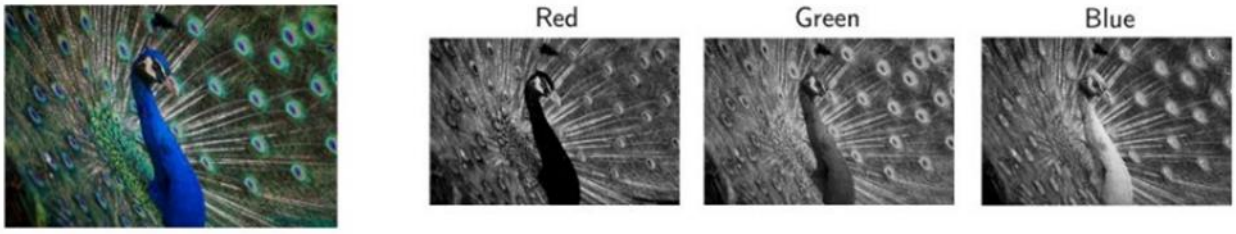
Шар об’єднання використовується для зменшення розміру вхідного зображення. У згортковій нейронній мережі за згортковим рівнем зазвичай слідує шар об’єднання. Шар об’єднання зазвичай додається, щоб прискорити обчислення та зробити деякі виявлені функції більш надійними.

Операція об’єднання також використовує ядро. На прикладі зображення нижче фільтр 2X2 використовується для об’єднання розміру вхідного зображення 4X4 із кроком 2.

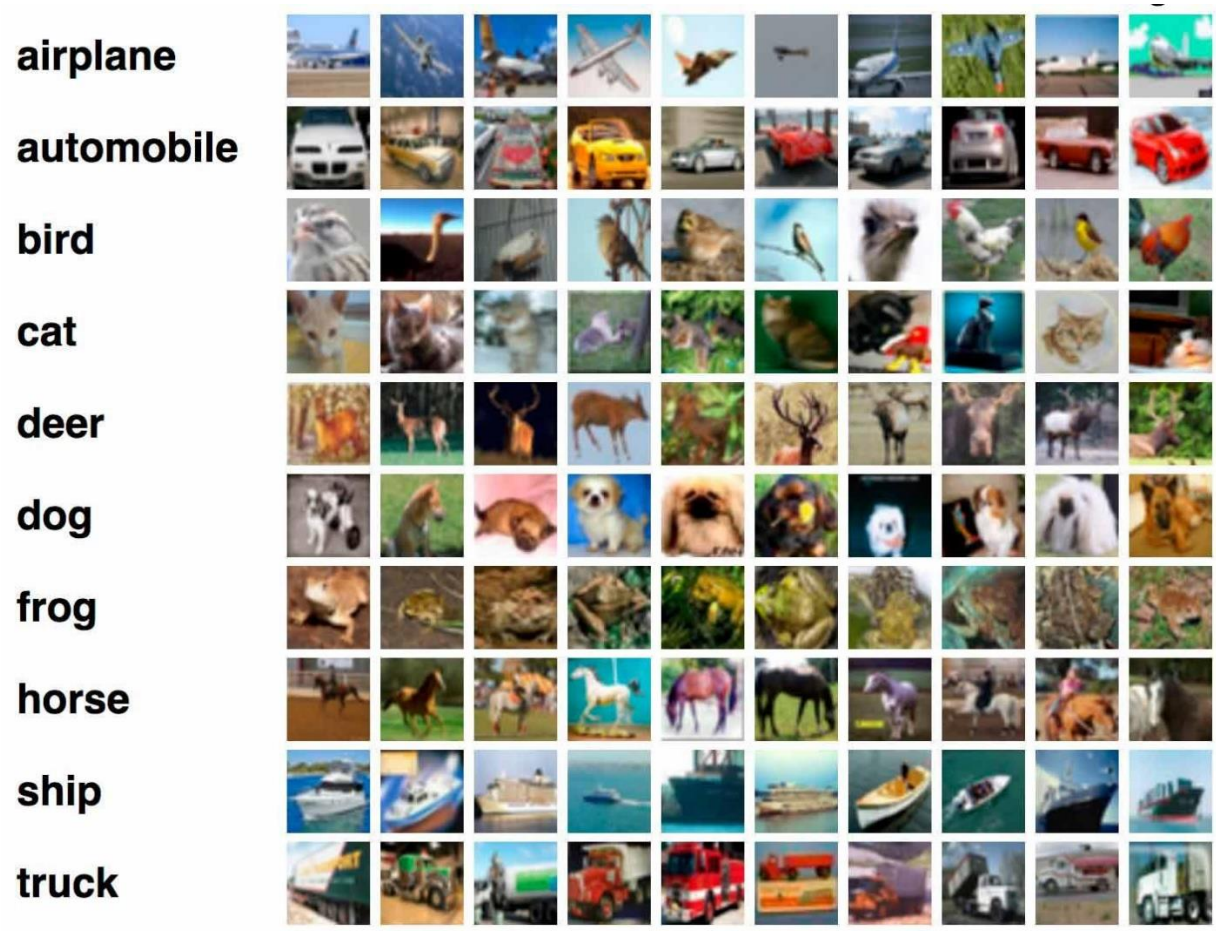
Існують різні типи об'єднання. Максимальний і середній пул є найбільш часто використовуваним методом об’єднання згорткової нейронної мережі.



Максимальне об’єднання: у максимальному об’єднанні з кожного фрагмента карти об’єктів вибирається максимальне значення для створення зменшеної карти.



Фотографія після використання трьох фільтрів.

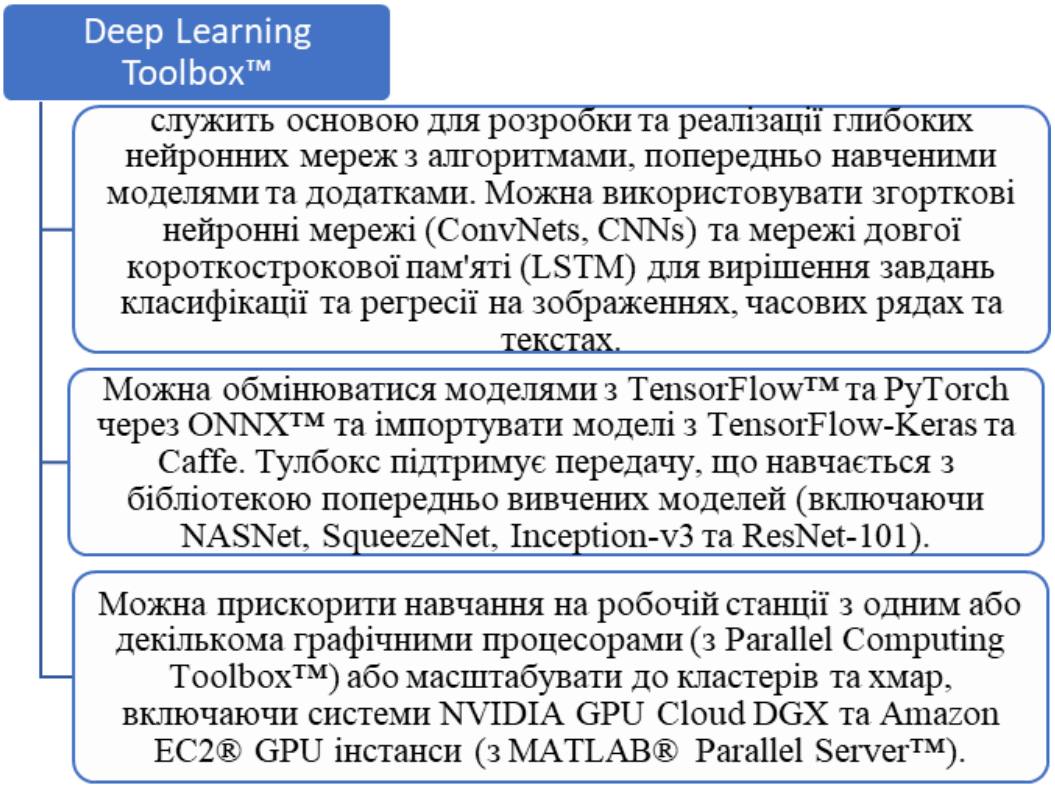


Класифікація образів

Розділ 3 Рішення задач з використанням Matlab та C#.

Deep Learning Toolbox

Deep Learning Toolbox (раніше Neural Network Toolbox) забезпечує основу для проектування та реалізації глибоких нейронних мереж, використовуючи як попередньо навчені моделі, так і програмb та інструментb для проектування розробки архітектур нейронних мереж.



Додаток Deep Network Designer дозволяє створювати, візуалізувати, редагувати та навчати мережі глибокого навчання. За допомогою цієї програми ви можете:

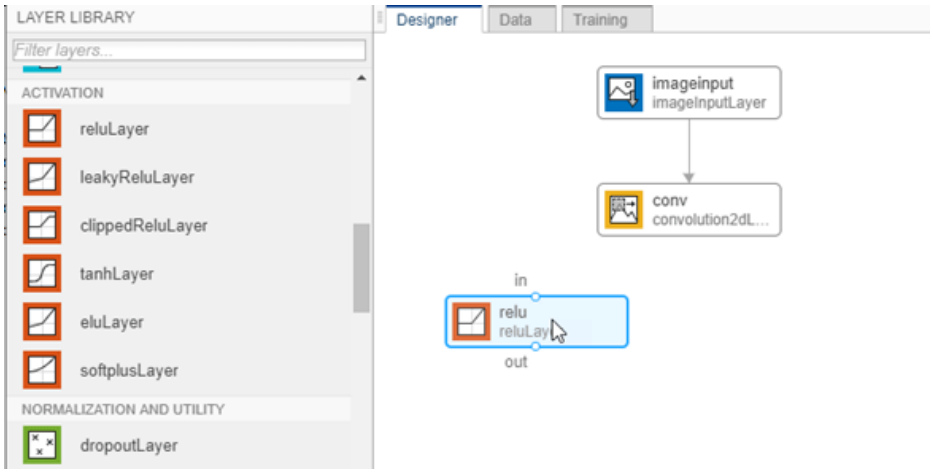
- створювати, імпортувати, редагувати та об'єднувати мережі.
- завантажувати попередньо навчені мережі та відредагувати їх для трансферного навчання.
- переглядати та редагувати властивості шарів та додавати нові шари та з'єднання.
- застосовувати розширення до навчальних даних щодо класифікації зображень та візуалізувати розподіл міток класів.
- навчати мережі та відстежувати навчання з діаграмами точності, втрат та валідації.
- експортувати навчені мережі до робочого простору або до Simulink®.
- створювати код MATLAB® для побудови та навчання мереж.

Потім ви можете тренувати свою мережу за допомогою Deep Network Designer або експортувати мережу для навчання за допомогою командного рядка.

Задачі, які вирішуються за допомогою Deep Network Designer

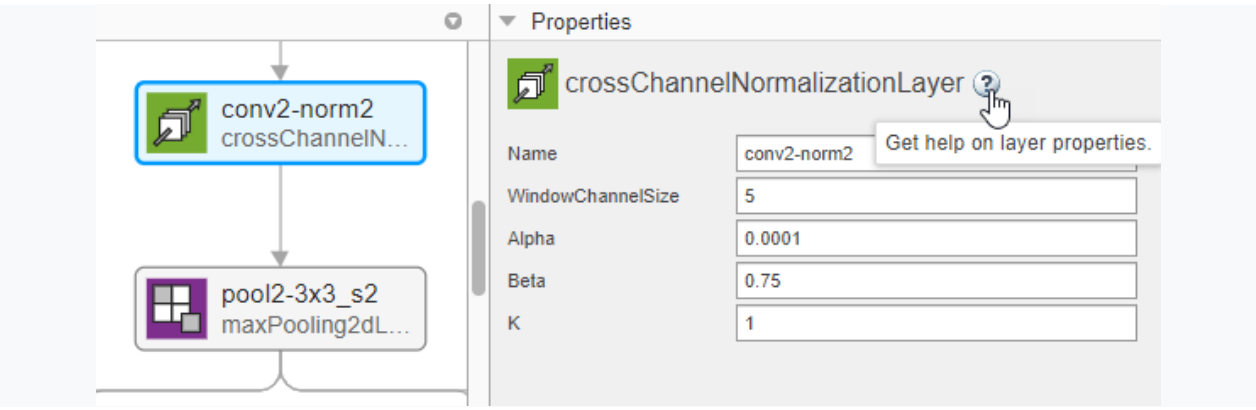
- Трансферне навчання
- Класифікація зображень
- Класифікація послідовності
- Числова класифікація даних
- Перетворення класифікаційної мережі в регресійну мережу
- Мережі з декількома входами та з багатьма виходами
- Глибокі мережі
- Розширені програми глибокого навчання
- dlnetwork для спеціальних навчальних циклів

Швидко сконструювати мережу можна перетягнувши блоки з бібліотеки шарів і з'єднавши їх. Для швидкого пошуку шарів скористайтеся вікном пошуку «Фільтрувати шари» на панелі «Бібліотека шарів».



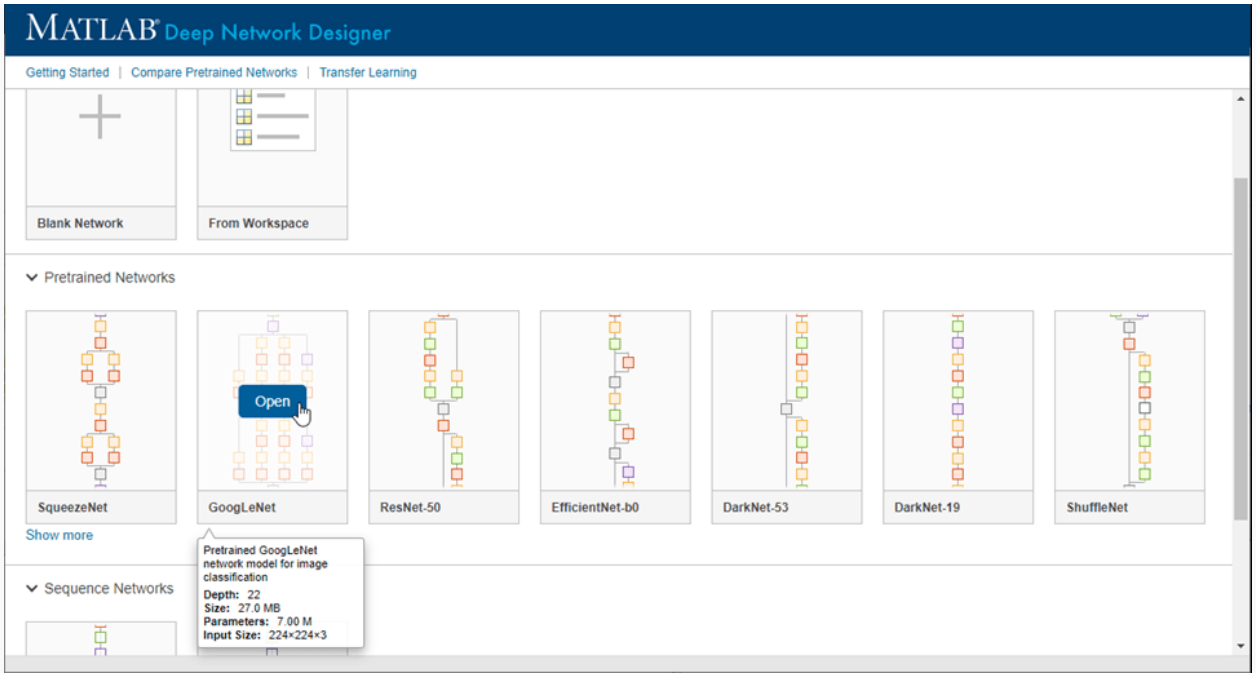
Ви можете додавати шари з робочого простору до мережі на панелі конструктора.

Ви також можете завантажити попередньо навчені мережі, натиснувши кнопку та вибравши їх на початковій сторінці. Для перегляду та редагування властивостей шару виберіть шар. Натисніть значок довідки біля назви шару, щоб отримати інформацію про властивості шару.



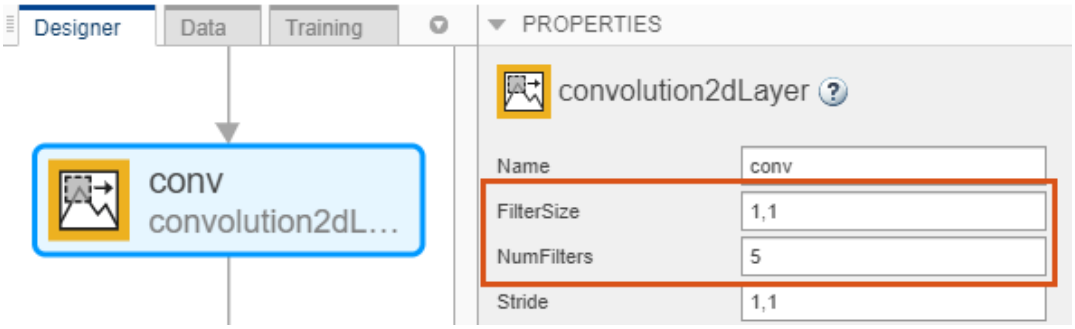
Deep Network Designer має вибір попередньо підготовлених мереж, придатних для передачі навчання з даними зображення.

Відкрийте програму та виберіть попередньо навчену мережу. Ви також можете завантажити попередньо підготовлену мережу, вибравши вкладку Конструктор і натиснувши Створити.



Адаптувати мережу для навчання

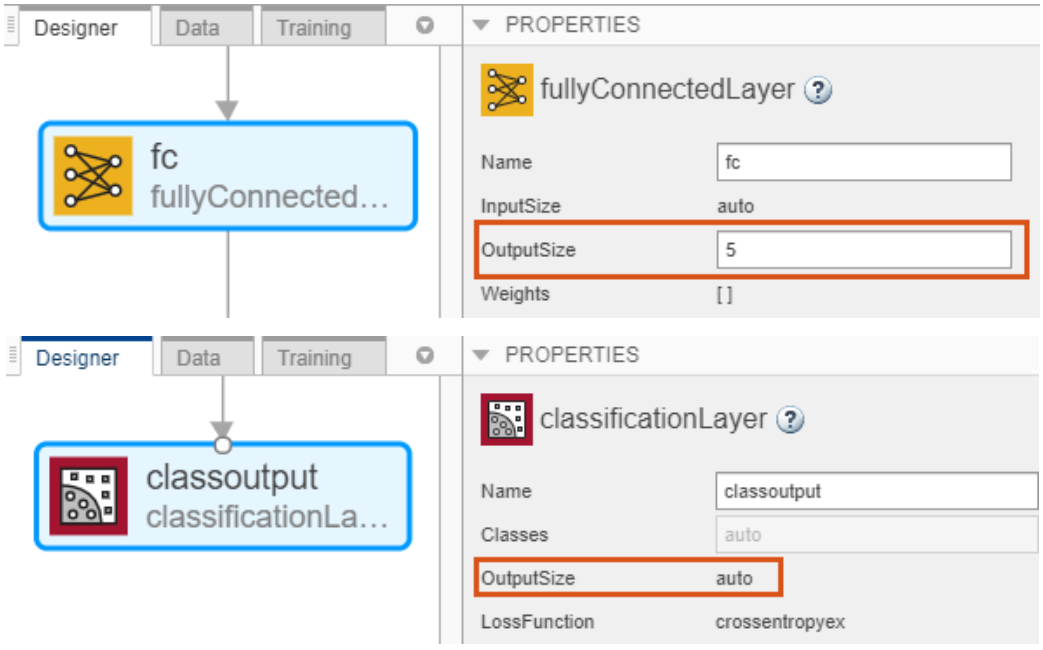
Щоб підготувати мережу до трансферного навчання, замініть останній засвоюваний рівень та останній рівень класифікації.



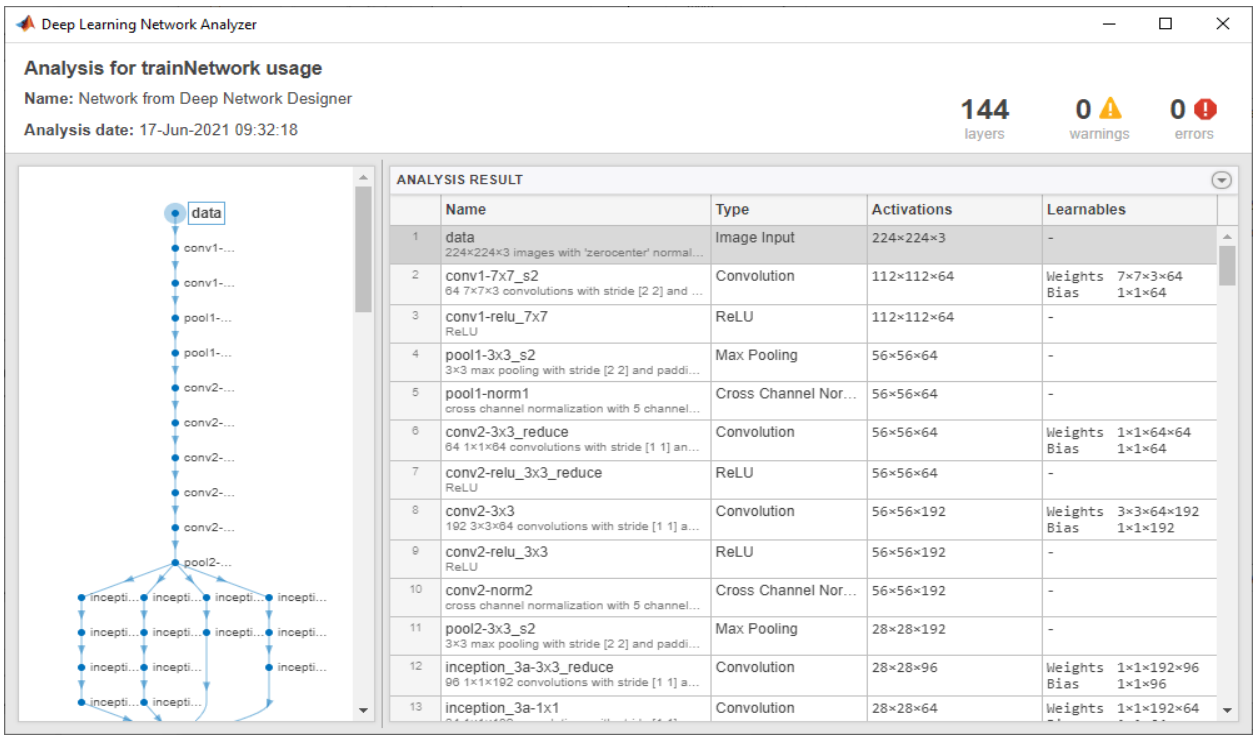
Якщо останній шар, що засвоюється, - це повнозв'зний шар (більшість попередньо навчених мереж, наприклад, GoogLeNet):

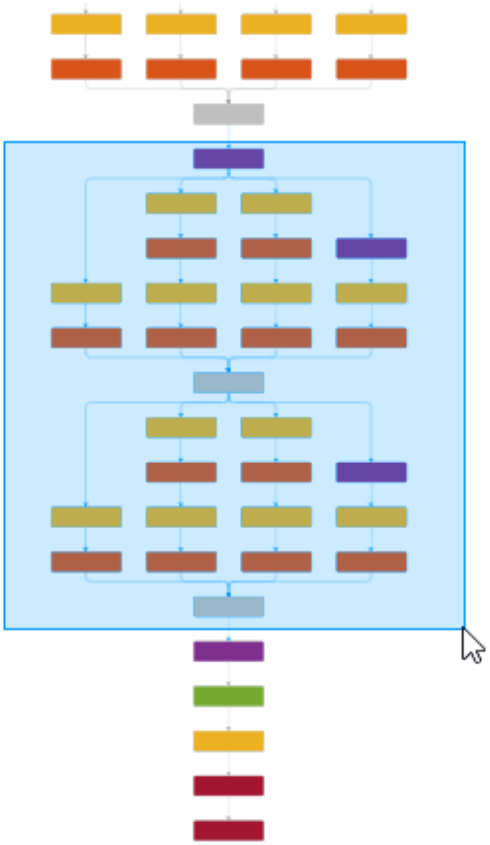
Перетягніть новий повністюConnectedLayer на полотно і встановіть для властивості OutputSize нову кількість класів. Змініть швидкість навчання так, щоб навчання було швидшим у новому рівні, ніж у переданих шарах, збільшивши значення WeightLearnRateFactor та BiasLearnRateFactor. Видаліть останній повністюConnectedLayer і замість цього підключіть новий шар.

Далі видаліть вихідний шар класифікації. Потім перетягніть на полотно новий класифікатор і під'єднайте його.



Щоб перевірити, чи мережа готова до навчання, на вкладці Конструктор (**Designer**) натисніть Аналіз.



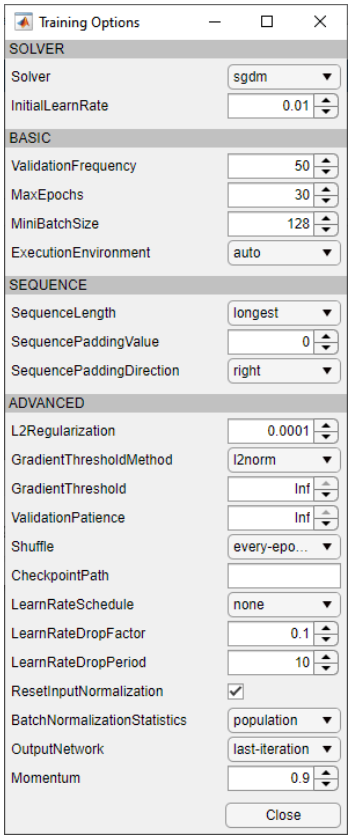


Ви можете використовувати Deep Network Designer для створення мережі послідовностей з нуля, або ви можете використовувати одну з попередньо побудованих нетренованих мереж зі стартової сторінки. Відкрийте початкову сторінку Deep Network Designer. Зупиніться на Sequence-to-Label і натисніть Open. Це відкриває попередньо створену мережу, придатну для вирішення проблем класифікації послідовностей.

Створення великих мереж може бути складним, ви можете використовувати Deep Network Designer для прискорення будівництва. Ви можете одночасно працювати з блоками шарів. Виділіть кілька шарів, потім скопіюйте та вставте або видаліть. Наприклад, можна використовувати блоки шарів для створення кількох копій груп згортки, пакетної нормалізації та ReLU шарів.

Навчання нейронних мереж.

Додаток Deep Network Designer дозволяє створювати та навчати глибокі нейронні мережі. Deep Network Designer підтримує навчання



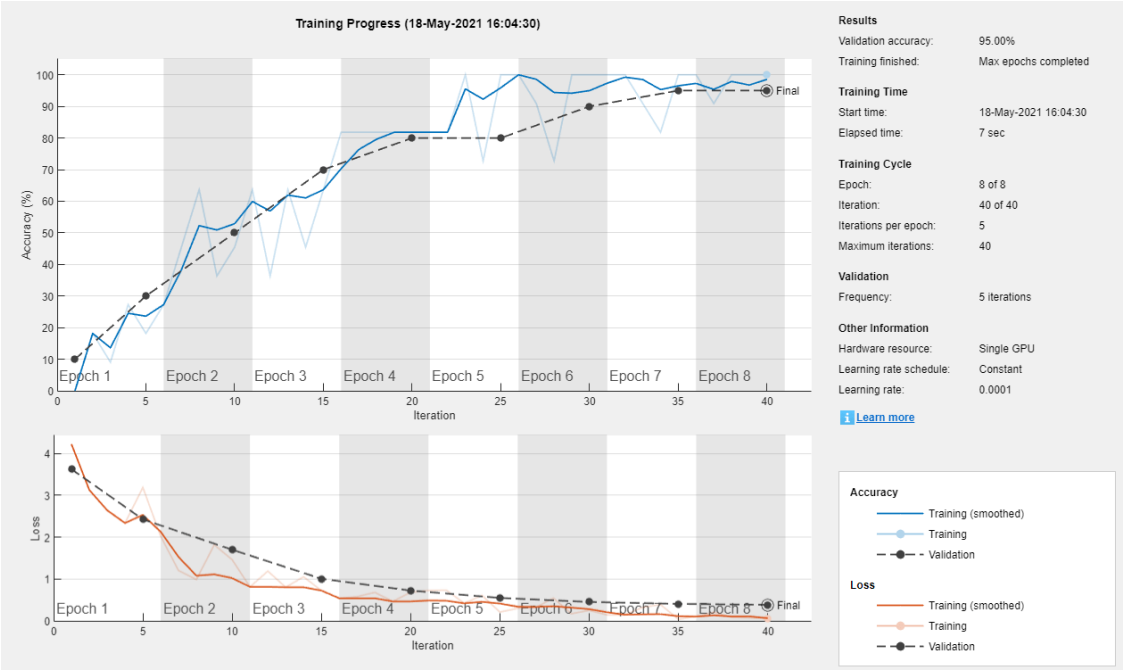
trainNetwork за допомогою даних зображень або об’єктів сховища даних. Ви також можете експортувати свою непідготовлену мережу для навчання за допомогою командного рядка, наприклад, для навчання вашої мережі за допомогою спеціальних навчальних циклів.

Щоб навчити мережу, виконайте такі дії:

1. Створити мережу
2. Імпортувати дані
3. Виберіть варіанти навчання
4. Навчання мережі
5. Експорт мережі.

Вибравши варіанти навчання, тренуйтеся в мережі, натиснувши **Train**. Додаток Deep Network Designer відображає анімований графік проходження навчання. Сюжет показує втрати та точність міні-партії, втрати та точність перевірки та додаткову інформацію про хід навчання. Ділянка має кнопку зупинки у правому верхньому куті. Натисніть кнопку, щоб припинити навчання та повернути поточний стан мережі.

За допомогою Deep Network Designer можна навчати різноманітні мережі. Наприклад, мережі класифікації або регресії зображень, мережі послідовностей, мережі числових даних, мережі семантичної сегментації та мережі регресії зображення до зображення. У Deep Network Designer можна навчити мережу за допомогою функції trainNetwork будь -які дані, які можна виразити як об’єкт сховища даних.



Графік навчання та похибки ШНМ

Обчислення виходу простого нейрона

Визначте параметри нейрона

```
close all, clear all, clc, format compact
% Ваги нейронів
w = [4 -2]
% Початкове зміщення
b = -3
% Активаційна функція
%func = 'tansig'
 func = 'purelin'
% func = 'hardlim'
% func = 'logsig'
```

Визначити вхідний вектор

```
p = [2 3]
```

Розрахувати вихід нейронів

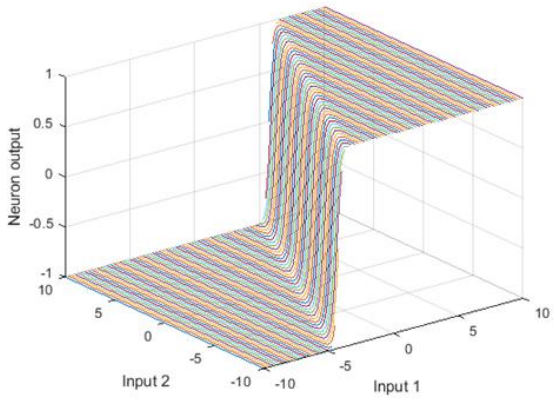
```
activation_potential = p*w'+b
neuron_output = feval(func, activation_potential)
```

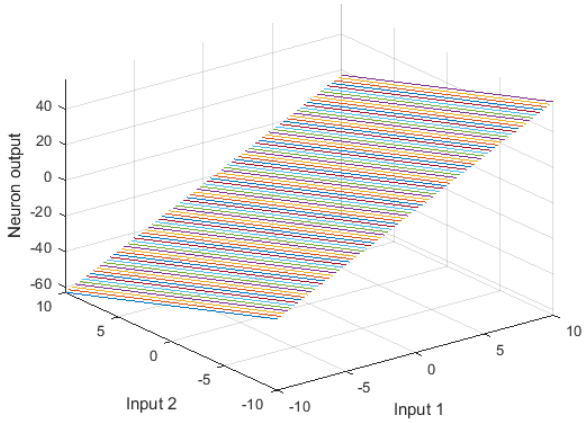
Побудуйте вихід нейрона в діапазоні вхідних даних

```
[p1,p2] = meshgrid(-10:.25:10);
z = feval(func, [p1(:) p2(:)]*w'+b );
z = reshape(z,length(p1),length(p2));
plot3(p1,p2,z)
grid on
```

```
xlabel('Input 1')
ylabel('Input 2')
zlabel('Neuron output')

activation_potential =-1   neuron_output =-0.7616
```

Вихід нейрона при різних функціях активації





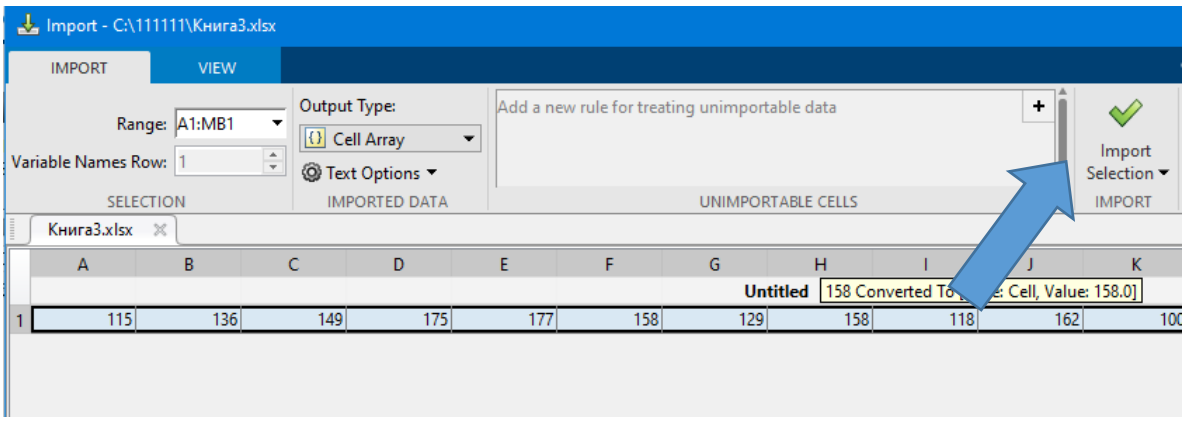
Прогнозування часових рядів

Щоб спрогнозувати значення майбутніх значень послідовності, ви можете навчити нейромережу, де значення представляють собою навчальні послідовності зі значеннями зміщеними на один часовий крок. Тобто на кожному часовому кроці вхідної послідовності мережа вчиться передбачати значення наступного значення.

У цьому прикладі використовується набір даних який навчає мережу LSTM прогнозувати кількість позитивних результатів з урахуванням кількості випадків в попередні місяці.

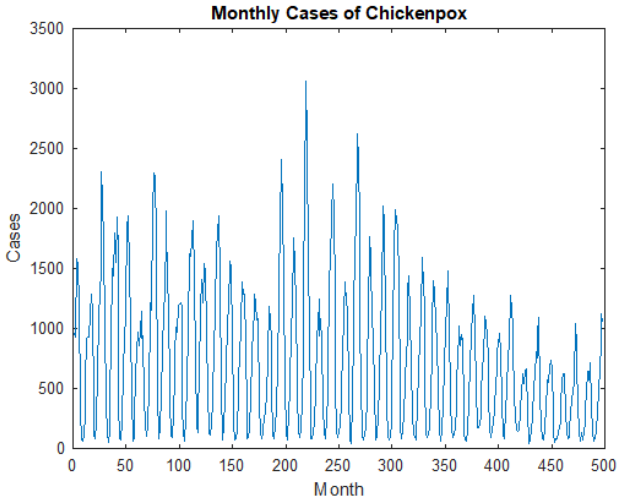
1. Введення даних

Дані вводяться за допомогою опції Import data. В прикладі дані вводяться з файлу Microsoft Exel.



Імпортуємо дані та виводимо графік на монітор.

```
>> data = Untitled;
>> data = [data{:}];
figure
plot(data)
xlabel("Month")
ylabel("Cases")
title("Monthly Cases of Chickenpox")
```



2. В вікні MatLab введемо код:

```
numTimeStepsTrain = floor(0.9*numel(data));
dataTrain = data(1:numTimeStepsTrain+1);
dataTest = data(numTimeStepsTrain+1:end);
```

В цих рядках дані розділяються на навчальні, які складають 90%, та тестові. Але для підвищення якості дані стандартизуються (згладжуються) відносно середнього значення та дисперсії.

```
mu = mean(dataTrain);
sig = std(dataTrain);
dataTrainStandardized = (dataTrain - mu) / sig;
```

Навчання здійснюється за даними, зміщеними на один часовий крок. Тобто на кожному часовому кроці вхідної послідовності мережа LSTM вчиться передбачати значення наступного часового кроку.

```
XTrain = dataTrainStandardized(1:end-1);
YTrain = dataTrainStandardized(2:end);
```

В процесі навчання створюється три масиви: adsXTrain навчальні значення, adsYTrain – прогрозні значення, cdsTrain – їх об’єднання.

```
adsXTrain = arrayDatastore(XTrain);
adsYTrain = arrayDatastore(YTrain);
cdsTrain = combine(adsXTrain,adsYTrain);
```

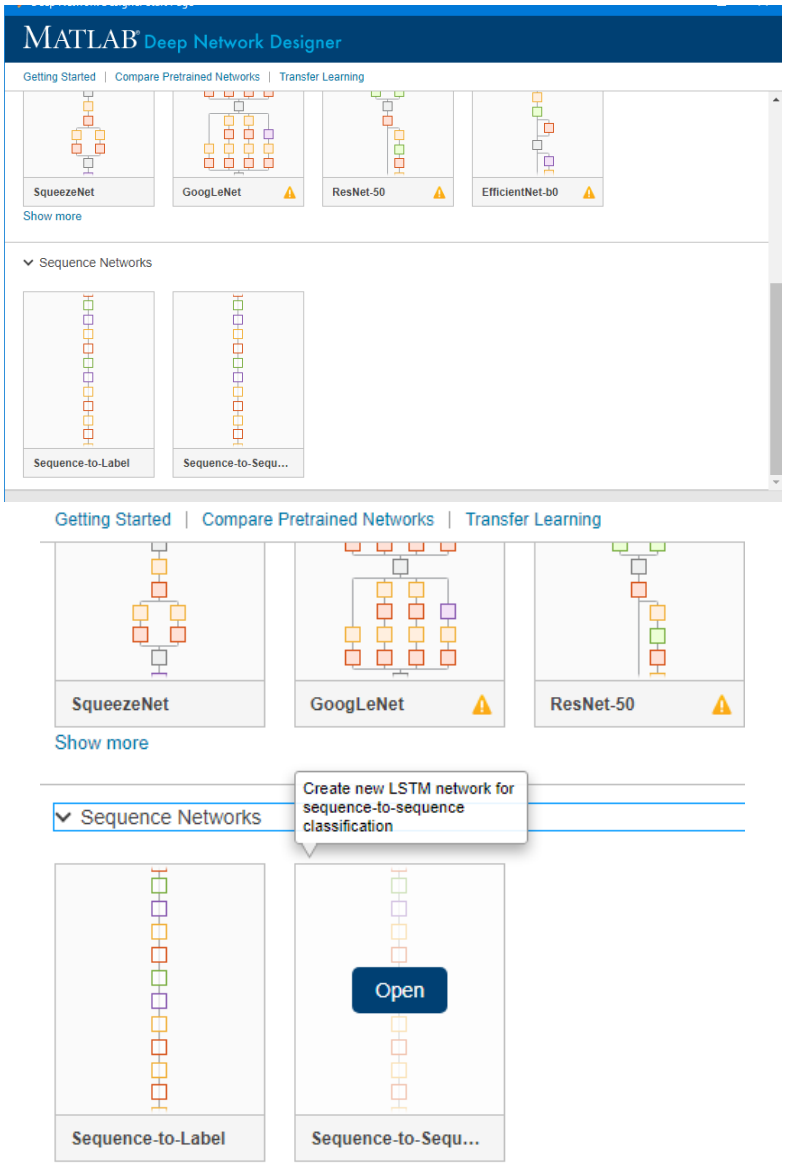
Створіть мережу регресії LSTM та її параметри. Наприклад numHiddenUnits = 200 означає 200 нейронів в прихованому шарі.

```
numFeatures = 1;
numResponses = 1;
numHiddenUnits = 200;
layers = [ ...
    sequenceInputLayer(numFeatures)
    lstmLayer(numHiddenUnits)
    fullyConnectedLayer(numResponses)
    regressionLayer];
options = trainingOptions('adam', ...
    'MaxEpochs',250, ... час навчання 250 епох;
    'GradientThreshold',1, ... степінь активаційної функції -1;
    'InitialLearnRate',0.005, ... початкова швидкість навчання 0,005;
    'LearnRateSchedule','piecewise', ...
    'LearnRateDropPeriod',125, ...
    'LearnRateDropFactor',0.2, ...
    'Verbose',0, ...
    'Plots','training-progress');
```

Але більш доречним є створення нейромережі конструктором. Для цього в вікні MatLab введемо код:

```
>> deepNetworkDesigner
```

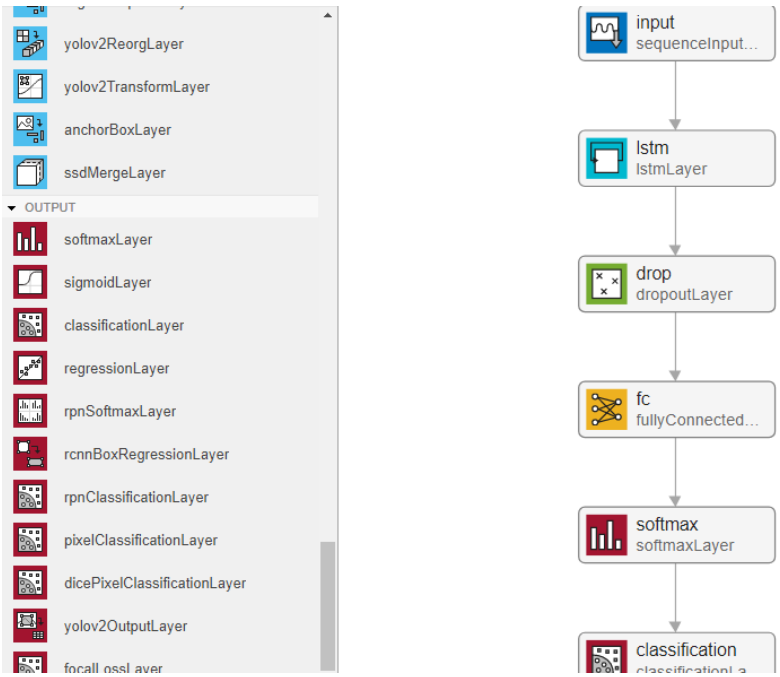
Відкривається вікно:

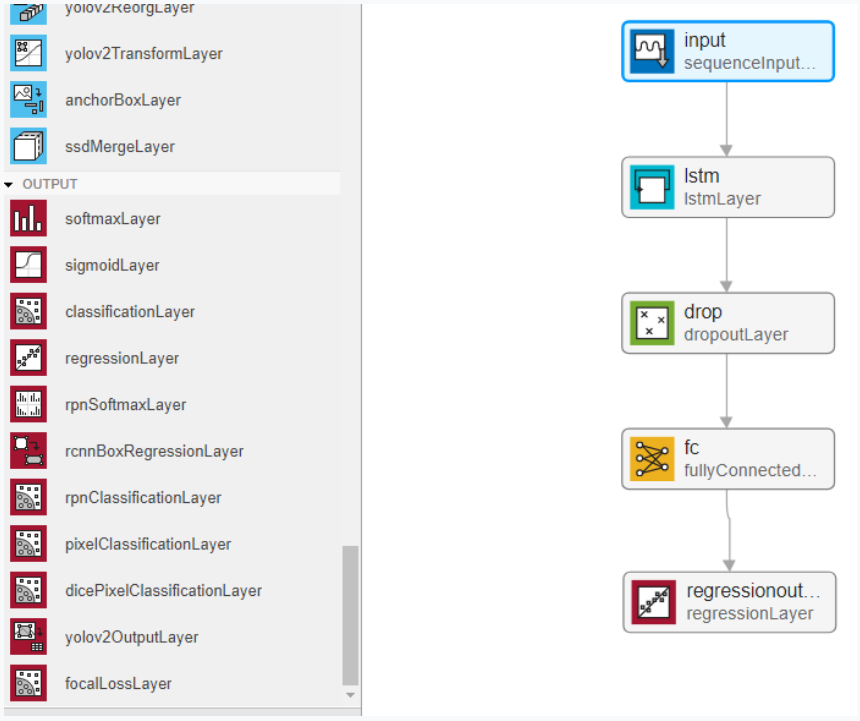


Для прогнозування використовуються мережі типу Sequence Networks, за допомогою яких вирішуються задачі класифікації, регресії, інтерполяції тощо.

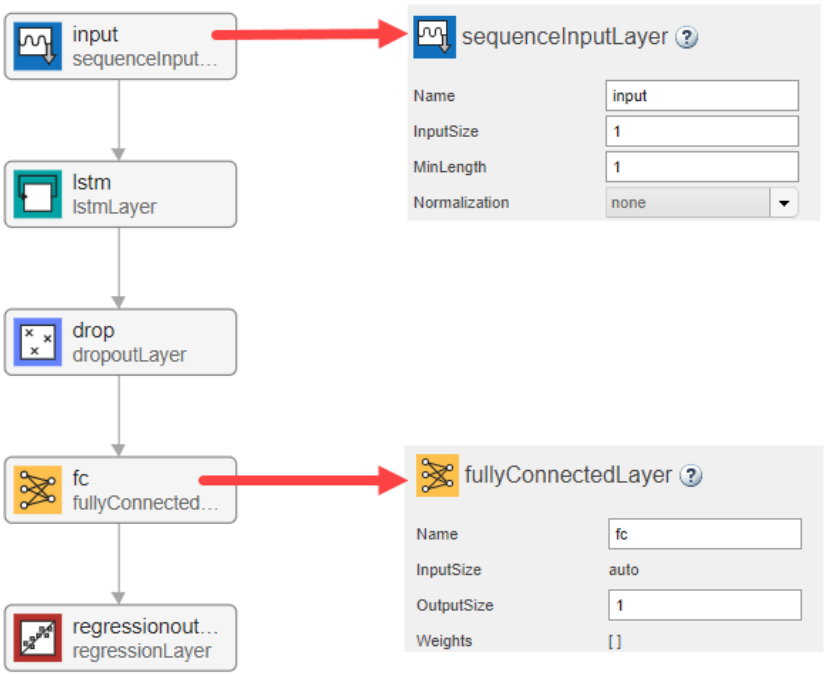
Створюється мережа.

Але ця мережа призначена для задач класифікації. Для задачі прогнозування її треба модифікувати. Замість двох останніх шарів softmax та classificfnion вставляємо модуль regressional, який знаходиться на панелі зліва.

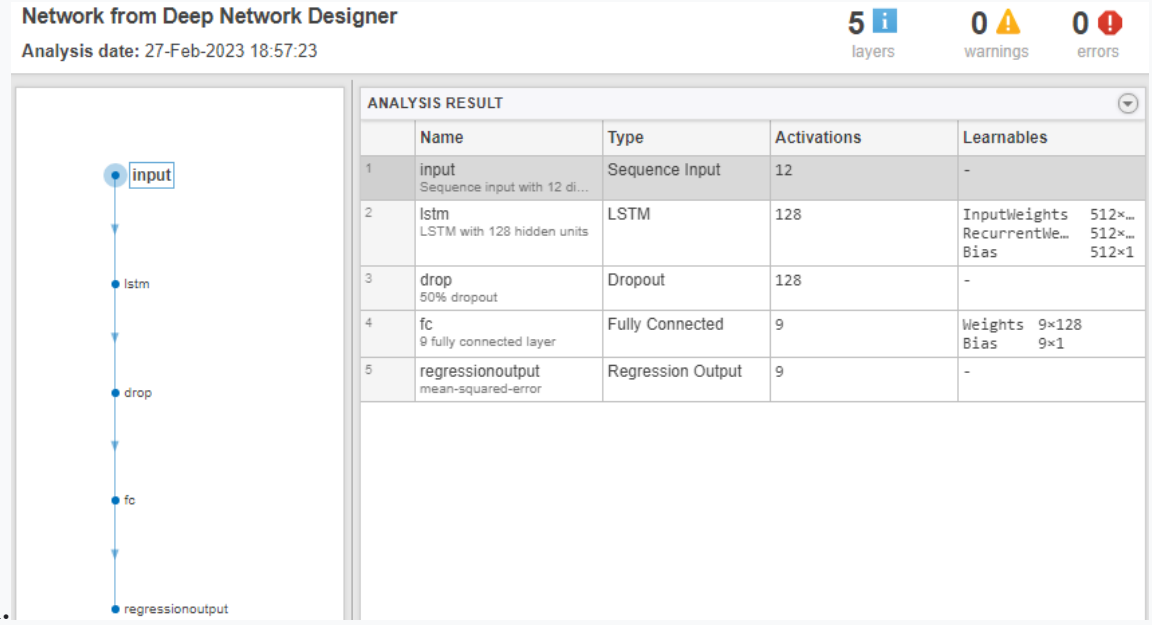


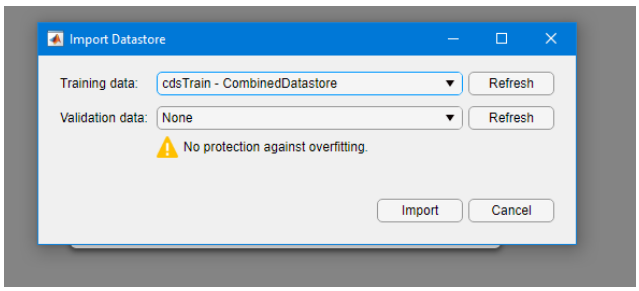


Після цього встановлюємо параметри кожного шару створеної нейромережі.

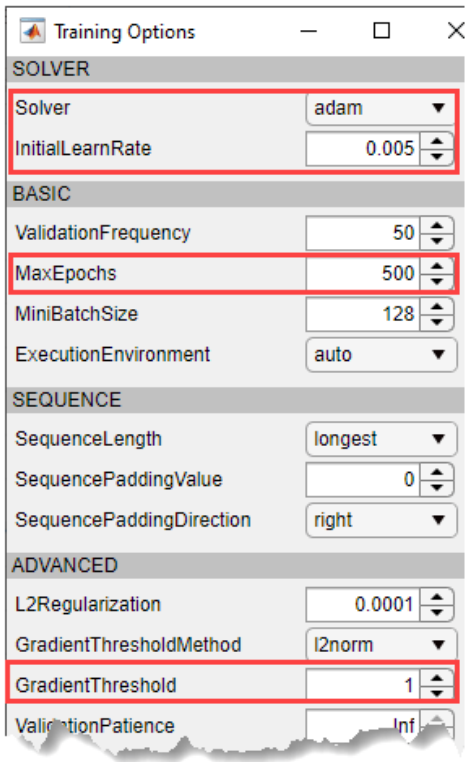


Натиснувши вкладку Analyse ми можемо отримати характеристики нейронної мережі, яку ми створили. змінити деякі з них.





Щоб імпортувати сховище навчальних даних, перейдіть на вкладку Data і натисніть Import Data > Import Datastore. Виберіть cdsTrain як навчальні дані та None як дані для перевірки. Натисніть кнопку Import.



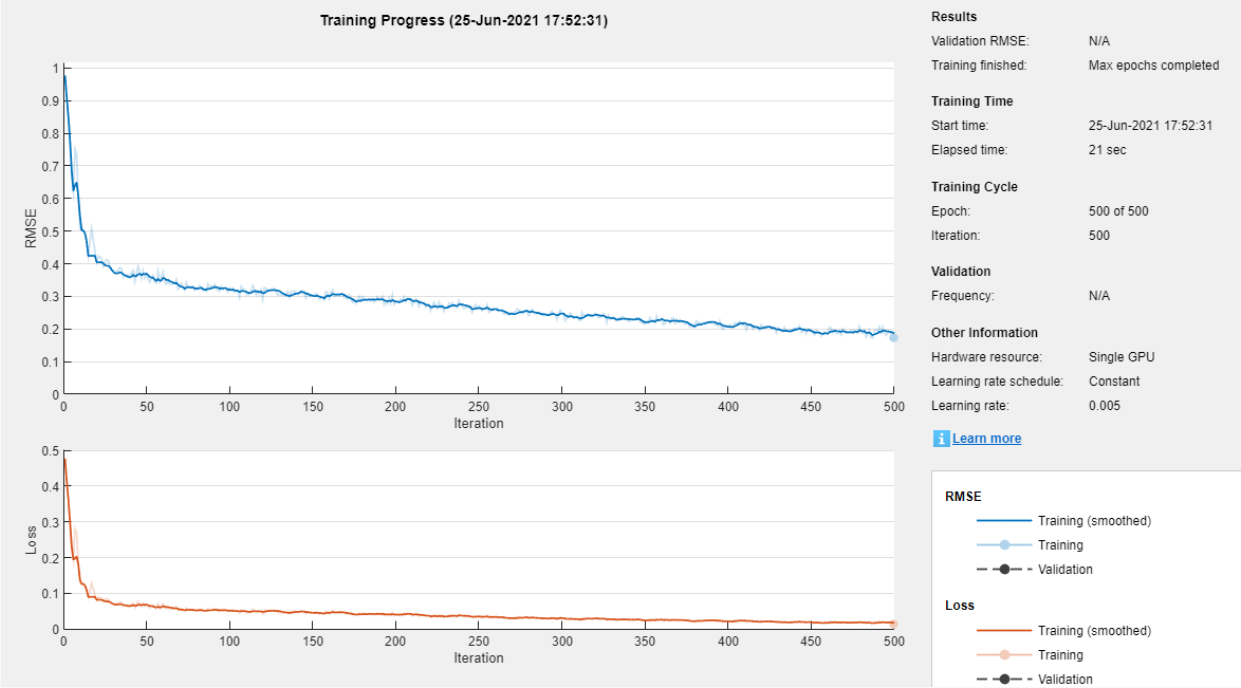
Попередній перегляд даних показує один вхідний часовий ряд і один часовий ряд відгуку, кожен з яких має 448 часових кроків.

Вкажіть параметри тренування.

На вкладці **Training** натисніть кнопку **Training Options**. Встановіть для параметра **Solver** значення **adam**, **InitialLearnRate** - 0.005, а **MaxEpochs** - 500. Щоб запобігти вибуху градієнтів, встановіть **GradientThreshold** рівним 1.

Натисніть **Train**.

Deep Network Designer відобразить анімовану діаграму, яка показує хід навчання. Графік показує втрати і точність міні-партій, втрати і точність валідації, а також додаткову інформацію про хід навчання.



Після завершення навчання експортуйте навчену мережу, натиснувши кнопку **Export** на вкладці **Training**. Навчена мережа зберігається як змінна trainedNetwork_1.

Прогнозування майбутніх часових кроків

Протестуйте навчену мережу, спрогнозувавши кілька часових кроків у майбутньому. Використовуйте функцію predictAndUpdateState для прогнозування часових кроків по одному і оновлення стану мережі після кожного прогнозу. Для кожного прогнозу використовуйте попередній прогноз як вхідні дані для функції. Стандартизуйте тестові дані, використовуючи ті самі параметри, що й навчальні дані.

```
dataTestStandardized = (dataTest - mu) / sig;
XTest = dataTestStandardized(1:end-1);
YTest = dataTest(2:end);
```

Щоб ініціалізувати стан мережі, спочатку зробіть прогноз на навчальних даних XTrain. Потім зробіть перше передбачення, використовуючи останній часовий крок навчальної відповіді YTrain(end). Пройдіться циклом по решті передбачень і введіть попереднє передбачення в predictAndUpdateState.

Для великих наборів даних, довгих послідовностей або великих мереж передбачення на GPU зазвичай обчислюються швидше, ніж передбачення на CPU. В інших випадках прогнози на CPU зазвичай обчислюються швидше. Для прогнозів з одним часовим кроком використовуйте CPU. Щоб використовувати процесор для прогнозування, встановіть для параметра 'ExecutionEnvironment' функції predictAndUpdateState значення 'cpu'.

```
net = predictAndUpdateState(trainedNetwork_1,XTrain);
[net,YPred] = predictAndUpdateState(net,YTrain(end));
numTimeStepsTest = numel(XTest);
for i = 2:numTimeStepsTest
    [net,YPred(:,i)] = predictAndUpdateState(net,YPred(:,i-
1),'ExecutionEnvironment','cpu');
end
```

Отримайте прогнози, використовуючи розраховані раніше параметри.

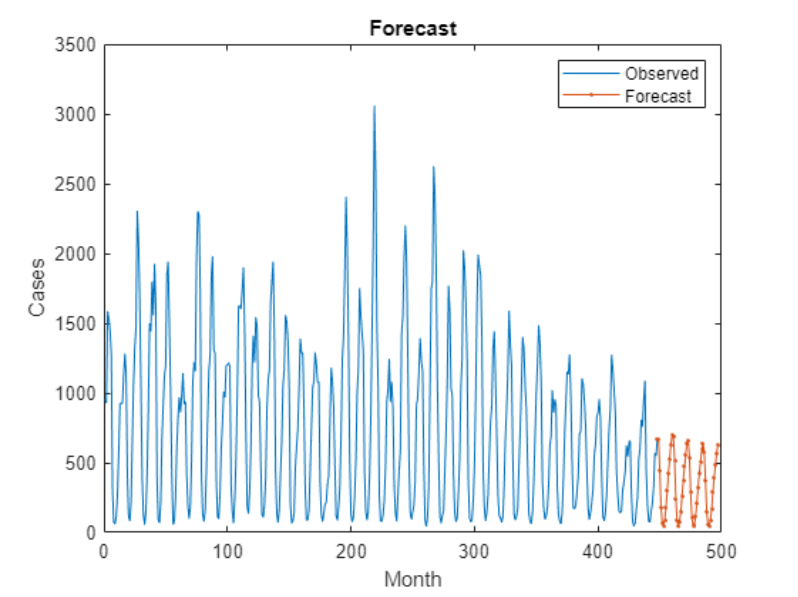
```
YPred = sig*YPred + mu;
```

Графік прогресу навчання показує середньоквадратичну похибку (RMSE), розраховану за стандартизованими даними. Обчисліть RMSE для нестандартизованих прогнозів.

```
rmse = sqrt(mean((YPred-YTest).^2))
```

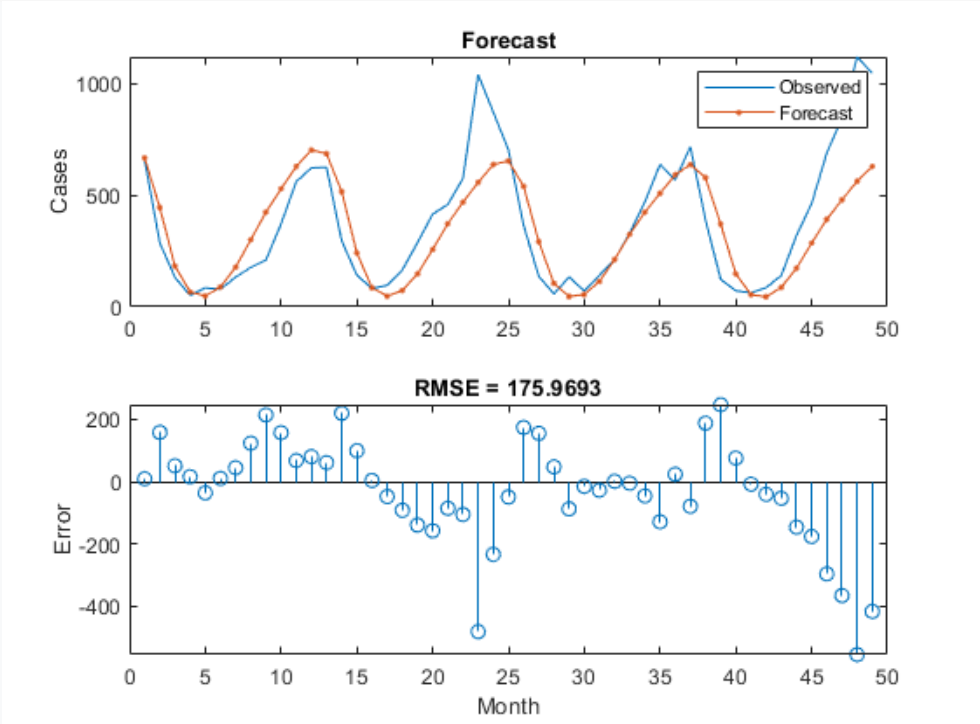
Побудуйте навчальний часовий ряд з прогнозованими значеннями.

```
figure
plot(dataTrain(1:end-1))
hold on
idx =
numTimeStepsTrain:(numTimeStepsTrain+numTimeStepsTest);
plot(idx,[data(numTimeStepsTrain) YPred],'.-')
hold off
xlabel("Month")
ylabel("Cases")
title("Forecast")
legend(["Observed" "Forecast"])
```



Порівняйте прогнозовані значення з тестовими даними.

```
figure
subplot(2,1,1)
plot(YTest)
hold on
plot(YPred,'.-')
hold off
legend(["Observed" "Forecast"])
ylabel("Cases")
title("Forecast")
subplot(2,1,2)
stem(YPred - YTest)
xlabel("Month")
ylabel("Error")
title("RMSE = " + rmse)
```



Використання радіальних базисних функцій.

Апроксимація набору даних

Представимо функцію

$$f(x) = \sum_{i=1}^{N} a_i \varphi_i(x)$$

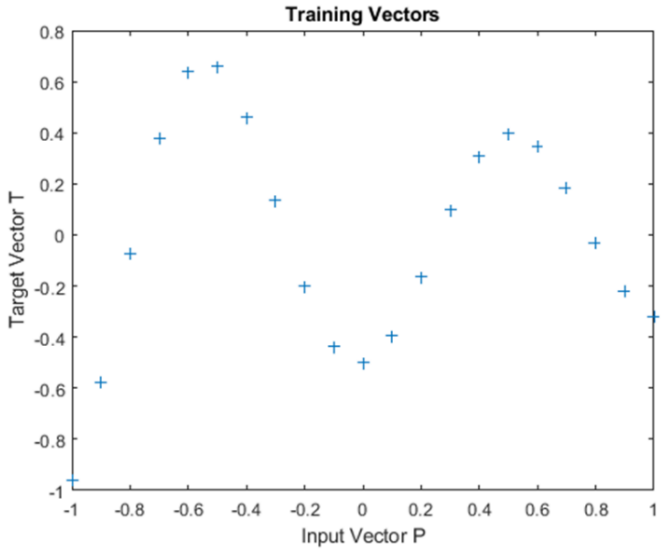
де $\varphi_i(x)$ радіальна базисна функція.

Використовується функція NEWRB для створення радіальної базової мережі.

Вхідні дані:

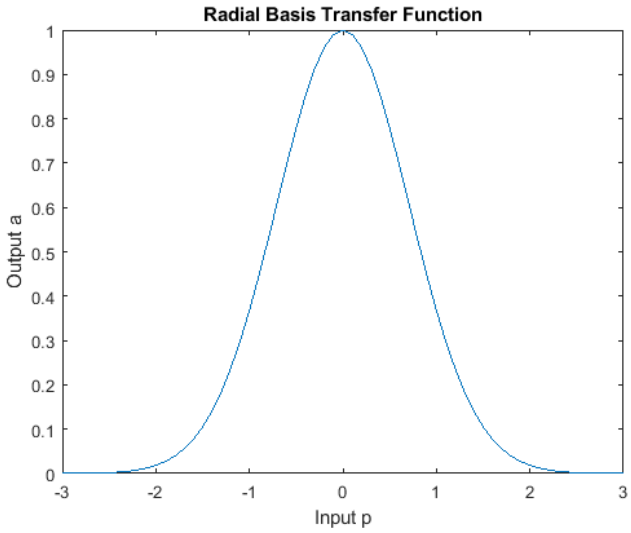
```
X = -1:.1:1;
T = [-.9602 -.5770 -.0729  .3771  .6405  .6600  .4609 ...
      .1336 -.2013 -.4344 -.5000 -.3930 -.1647  .0988 ...
      .3072  .3960  .3449  .1816 -.0312 -.2189 -.3201];
plot(X,T,'+');
title('Training Vectors');
xlabel('Input Vector P');
ylabel('Target Vector T');
```

Радіально-базисна мережа – двошарова. Тобто має прихований шар радіальних базових нейронів і вихідний шар лінійних нейронів.



Функція передачі радіальної основи, яка використовується прихованим шаром.

```
x = -3:.1:3;
a = radbas(x);
plot(x,a)
title('Radial Basis Transfer Function');
xlabel('Input p');
ylabel('Output a');
```

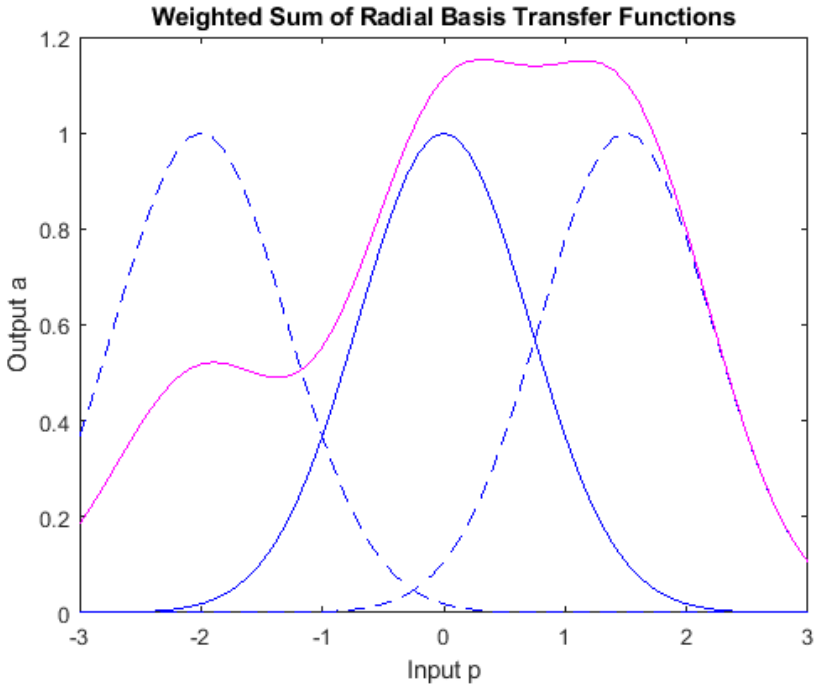


Ваги та зміщення кожного нейрона в прихованому шарі визначають положення та ширину радіальної базисної функції. Кожен лінійний вихідний нейрон утворює зважену суму цих радіальних базисних функцій. Завдяки правильній вазі та значенням зсуву для кожного шару та достатній кількості прихованих нейронів радіальна базисна мережа може відповідати будь-якій функції з будь-якою бажаною точністю. Це приклад трьох радіальних базисних функцій, які масштабуються та підсумовуються для створення апроксимуючої функції.

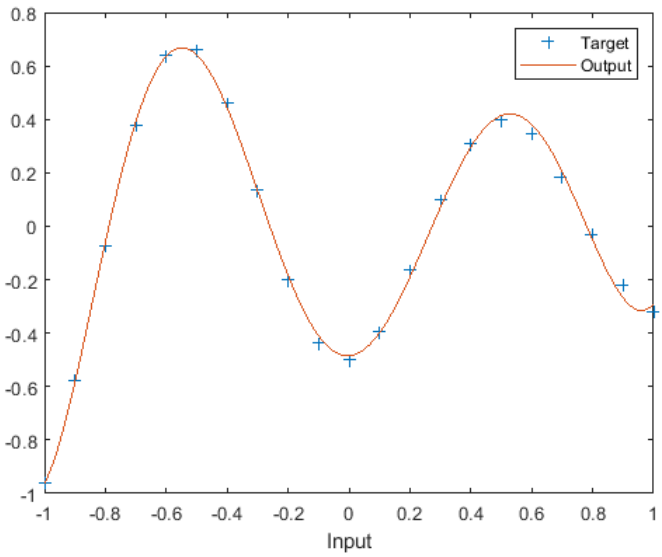
```
a2 = radbas(x-1.5);
a3 = radbas(x+2);
a4 = a + a2*1 + a3*0.5;
plot(x,a,'b-',x,a2,'b--',x,a3,'b--',x,a4,'m-')
title('Weighted Sum of Radial Basis Transfer Functions');
xlabel('Input p');
ylabel('Output a');
```

Функція NEWRB швидко створює радіальну базисну мережу, яка апроксимує функцію, визначену P і T. Окрім навчального набору та цілей, NEWRB приймає два аргументи: мінімум суми квадратів помилки та коефіцієнт зміщення.

```
eg = 0.02; % бажана помилка
sc = 1;    % зміщення
net = newrb(X,T,eg,sc);
```



Для того, щоб побачити, як працює мережа, перебудовується навчальний набір та моделюється відповідь мережі для входів у тому ж діапазоні.



Для того, щоб побачити, як працює мережа, перебудовується навчальний набір та моделюється відповідь мережі для входів у тому ж діапазоні.

```
plot(X,T,'+');
xlabel('Input');

X = -1:.01:1;
Y = net(X);

hold on;
plot(X,Y);
hold off;
legend({'Target','Output'})
```

Рішення диференціальних рівнянь (функція Бесселя)

$$t^2 \cdot y'' + t \cdot y' + (t^2 - \alpha^2) \cdot y = 0$$

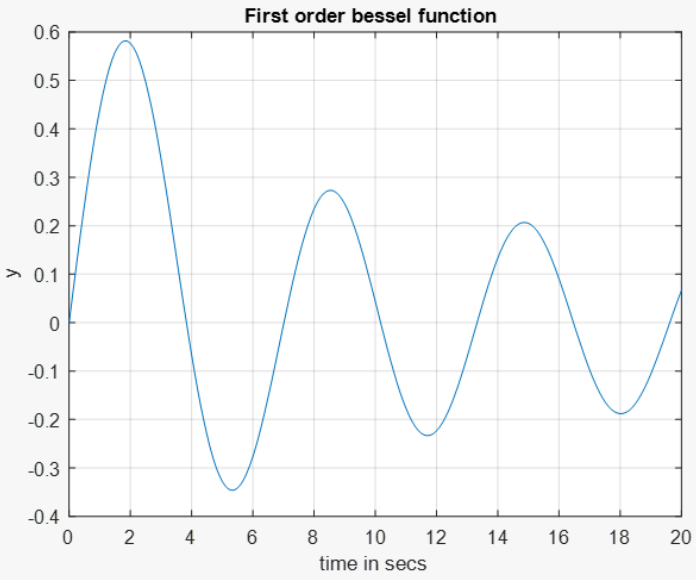
функція Бесселя першого порядку $\alpha = 1, t \in [0,20]$.

Завдання.

Спробуйте різні конструкції для підгонки результату. Вам також знадобляться різні алгоритми навчання.

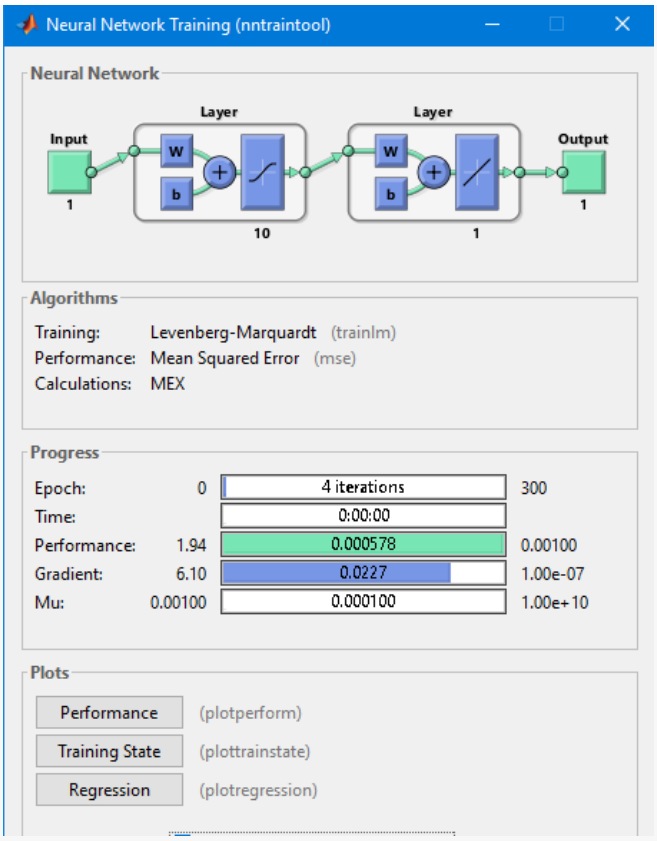
Введення даних.

```
t=0:0.1:20; y=bessel(1,t);
plot(t,y)
grid
xlabel('time in secs');ylabel('y'); title('First order
bessel function');
```

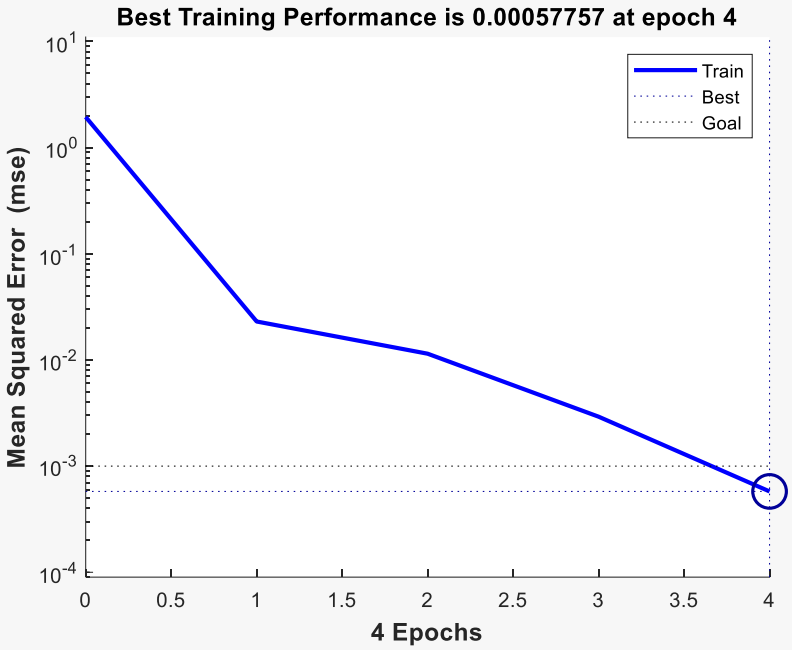


Створимо нейромережу оберненого розповсюдження.

```
P=t; T=y;
net=newff([0 20], [10,1], {'tansig','purelin'},'trainlm');
net.trainParam.show = 50;
net.trainParam.lr = 0.05;
net.trainParam.epochs = 300;
net.trainParam.goal = 1e-3;
net1 = train(net, P, T);
```

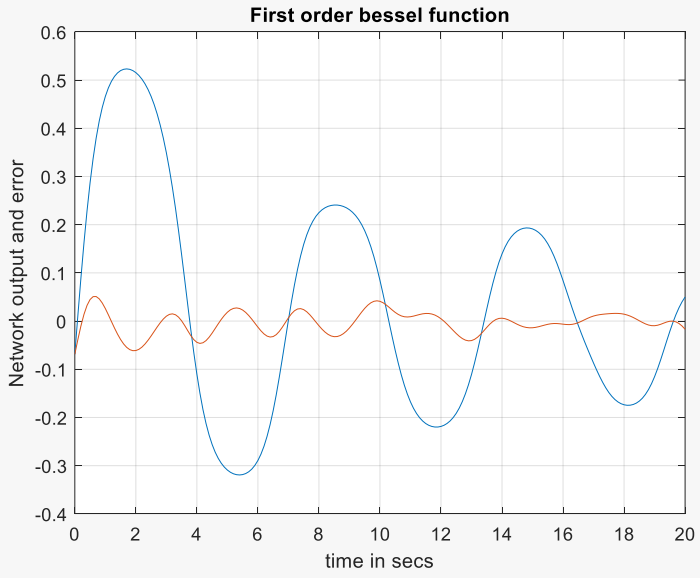



```
TRAINLM, Epoch 0/300, MSE 11.2762/0.001, Gradient
1908.57/1e-010
TRAINLM, Epoch 3/300, MSE 0.000417953/0.001,
Gradient 1.50709/1e-010
TRAINLM, Performance goal met.
```

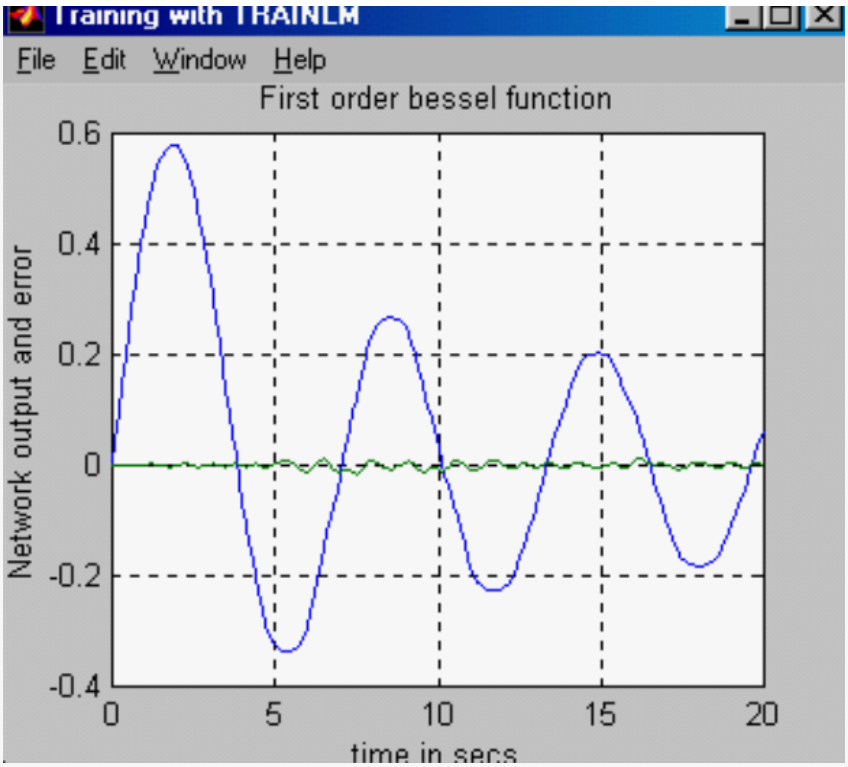


Симуляція результату

```
a= sim(net1,P);
%Plot result and compare
plot(P,a,P,a-T)
xlabel('time in secs');ylabel('Network output and error');
title('First order bessel function'); grid
```



Так як помилка досить значна, зменшимо її, подвоївши кількість вузлів у першому прихованому шарі до 20 для зниження допустимої похибки до 10^{-4}.



Результат значно кращий, хоча й потребує доопрацювання.

Побудова 3D поверхні та перевірка її на точність

$$z = \cos(x) \cdot \sin(y)$$

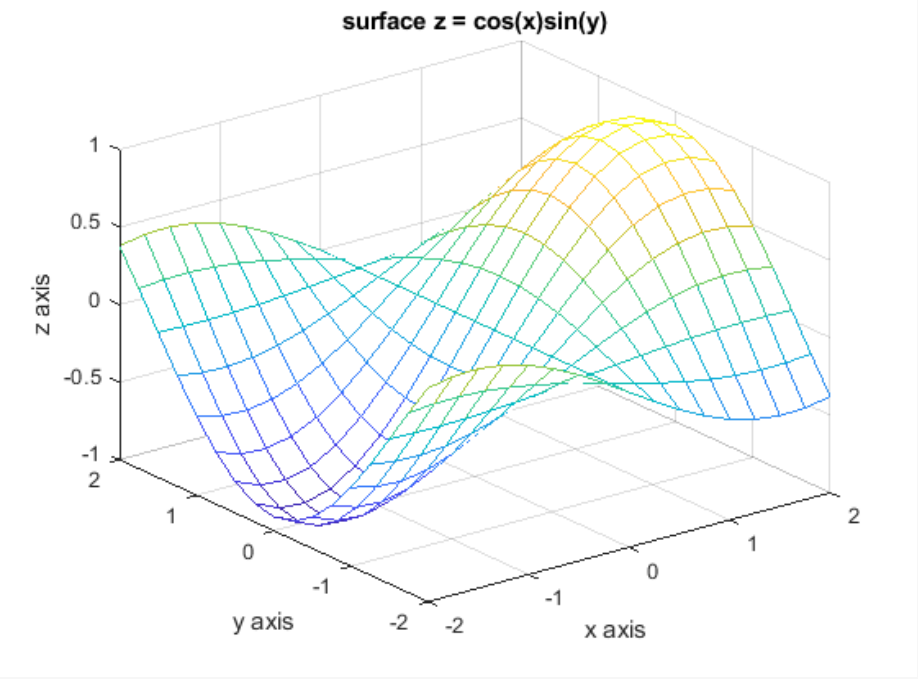
$$-2 \le x \le 2 \quad -2 \le y \le 2$$

Початкові дані

```
x = -2:0.25:2; y = -2:0.25:2;
z = cos(x)'*sin(y);
```

Побудова поверхні засобами Матлаб.

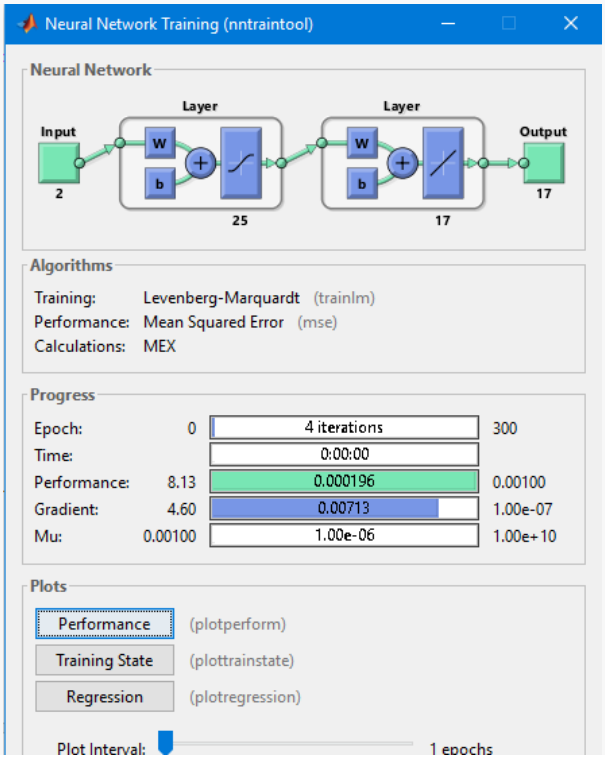
```
mesh(x,y,z)
xlabel('x axis'); ylabel('y axis'); zlabel('z axis');
 title('surface z = cos(x)sin(y)');
gi=input('Strike any key ...');
pause
```



Побудова нейромережі

матриця P і вихідний вектор T

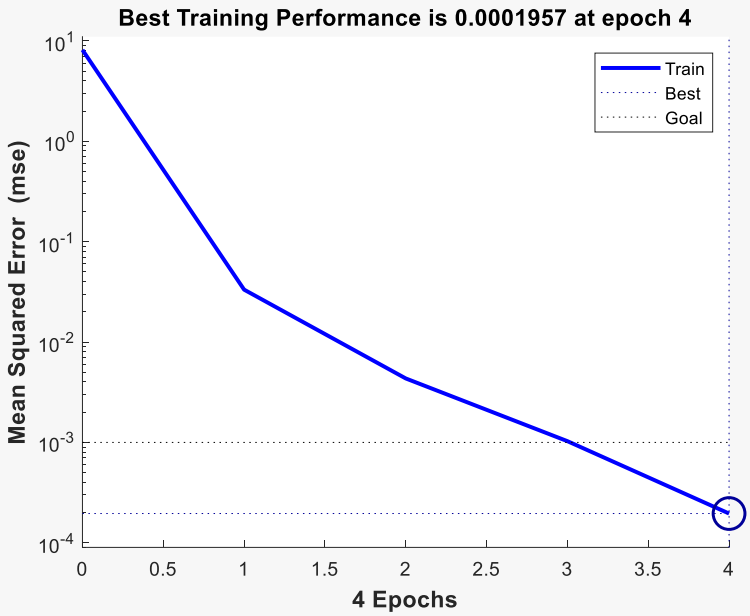
```
P = [x;y];  T = z;
net=newff([-2 2; -2 2], [25 17], {'tansig' 'purelin'},'trainlm');
net.trainParam.show = 50;
net.trainParam.lr = 0.05;
net.trainParam.epochs = 300;
net.trainParam.goal = 1e-3;
```

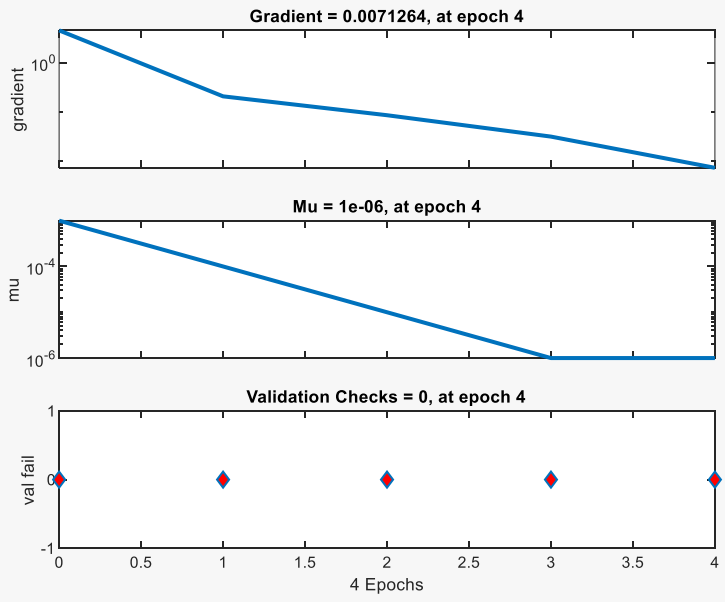


Навчання мережі

```
%Train network
net1 = train(net, P, T);
gi=input('Strike any key ...');
```

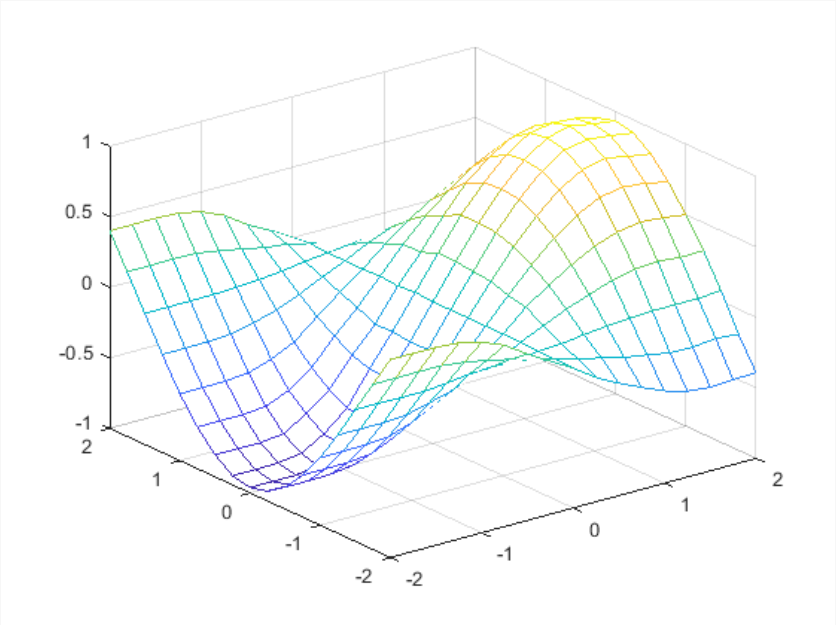
```
TRAINLM, Epoch 0/300, MSE 9.12393/0.001, Gradient
684.818/1e-010
TRAINLM, Epoch 3/300, MSE 0.000865271/0.001,
Gradient 5.47551/1e-010
TRAINLM, Performance goal met.
```





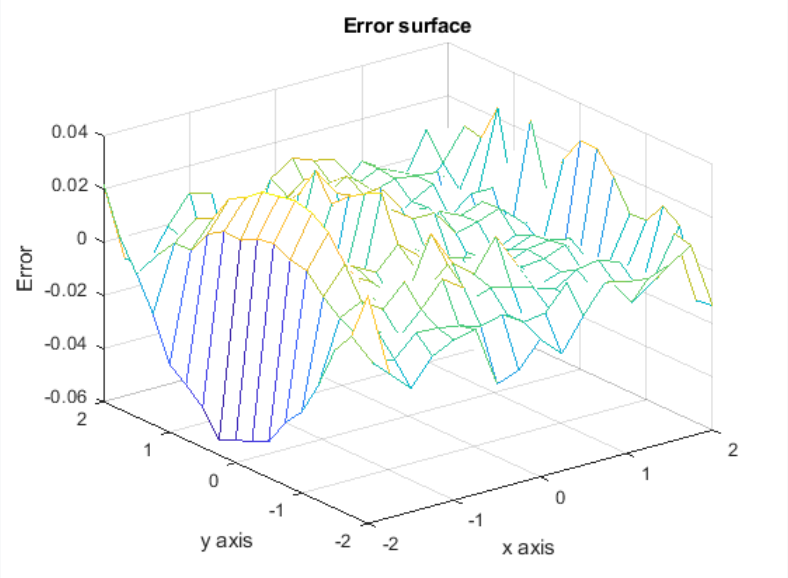
Побудова поверхні за допомогою створеної штучної нейронної мережі.

```
a= sim(net1,P);
 mesh(x,y,a)
```



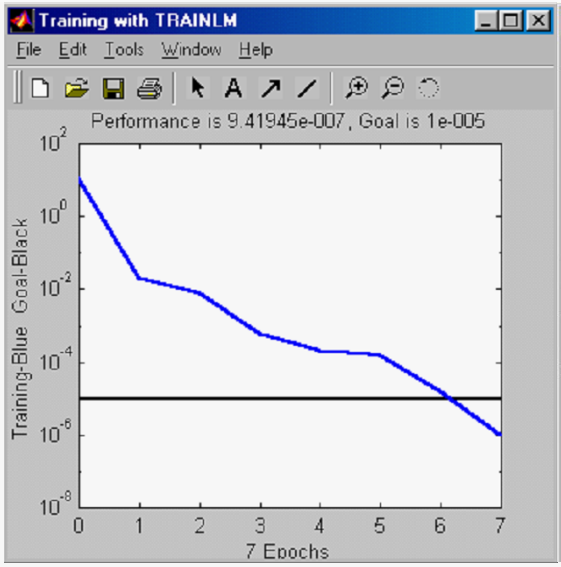
Результат виглядає задовільним, але при ближчому розгляді виявляється, що в деяких областях наближення є не так добре. Це краще видно, намалювавши поверхню помилок.

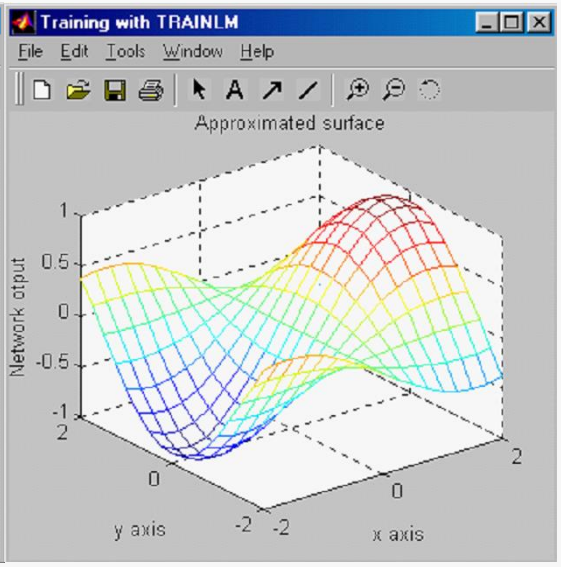
```
mesh(x,y,a-z)
xlabel('x axis'); ylabel('y axis'); zlabel('Error');
title('Error surface')
```

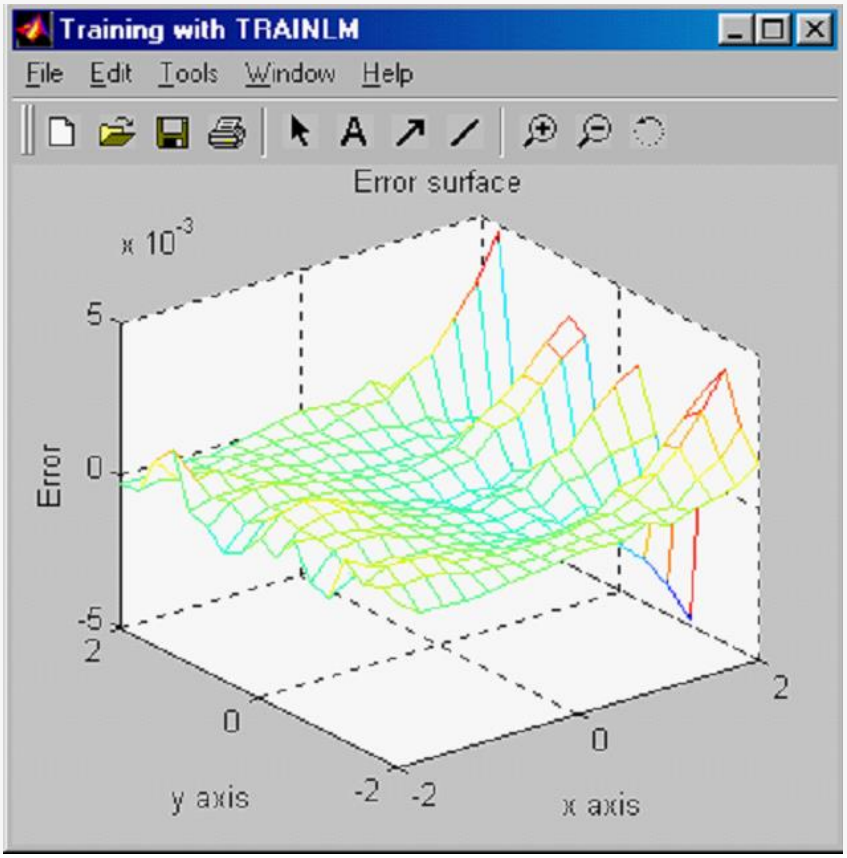


```
% Maximum fitting error
Maxfiterror = max(max(z-a))
Maxfiterror = 0.1116
```

Залежно від обчислювальної потужності вашого комп’ютера допуск до помилок може бути суворішим, скажімо, 10^{-5}







Класифікація за допомогою логічної операції XOR

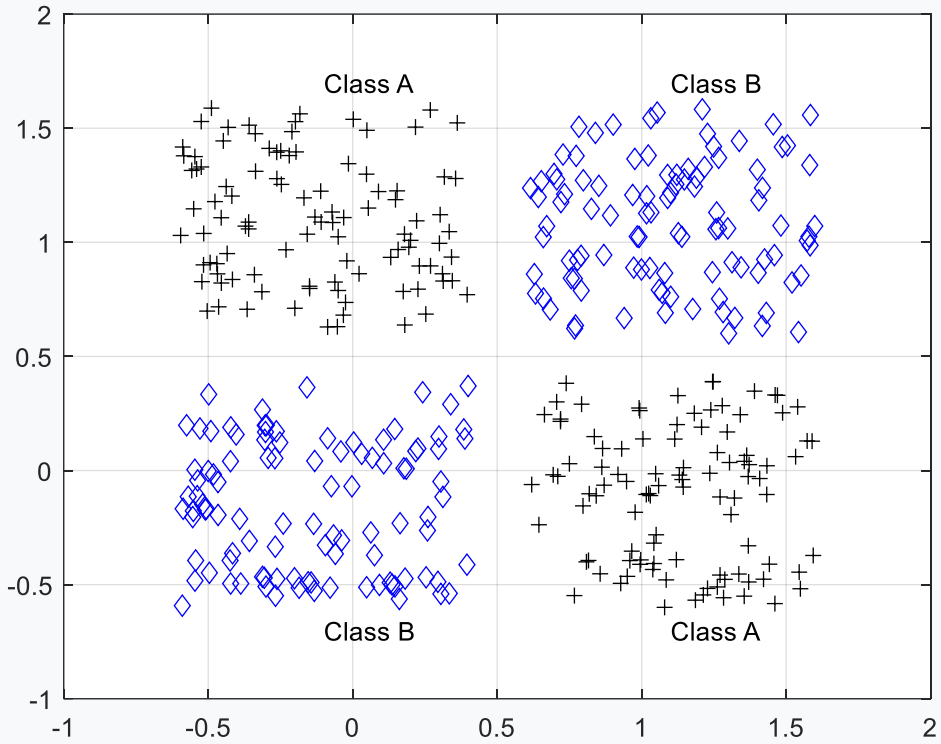
Розв'язування задачі XOR з багатошаровим персептроном

Постановка задачі

4 кластери даних (A, B, C, D) визначені у двовимірному просторі введення. (A,C) і (B,D) кластери представляють проблему класифікації XOR. Завданням є визначити нейронну мережу для вирішення проблеми XOR.

Визначте 4 кластери вхідних даних

```
close all, clear all, clc, format compact

% кількість зразків кожного класу
K = 100;
% визначити 4 кластери вхідних даних
q = .6; % границі класів
A = [rand(1,K)-q; rand(1,K)+q];
B = [rand(1,K)+q; rand(1,K)+q];
C = [rand(1,K)+q; rand(1,K)-q];
D = [rand(1,K)-q; rand(1,K)-q];
% plot clusters
figure(1)
plot(A(1,:),A(2,:),'k+')
hold on
grid on
plot(B(1,:),B(2,:),'bd')
plot(C(1,:),C(2,:),'k+')
plot(D(1,:),D(2,:),'bd')
% text labels for clusters
text(.5-q,.5+2*q,'Class A')
text(.5+q,.5+2*q,'Class B')
text(.5+q,.5-2*q,'Class A')
text(.5-q,.5-2*q,'Class B')
```



Визначте вихідне кодування для проблеми XOR

```
% кодувати кластери a і c як один клас, a b і d як інший
клас
a = -1; %  a  |  b
c = -1; %  -------
b =  1; %  d  |  c
d =  1; %
```

Підготуйте входи та виходи для навчання мережі

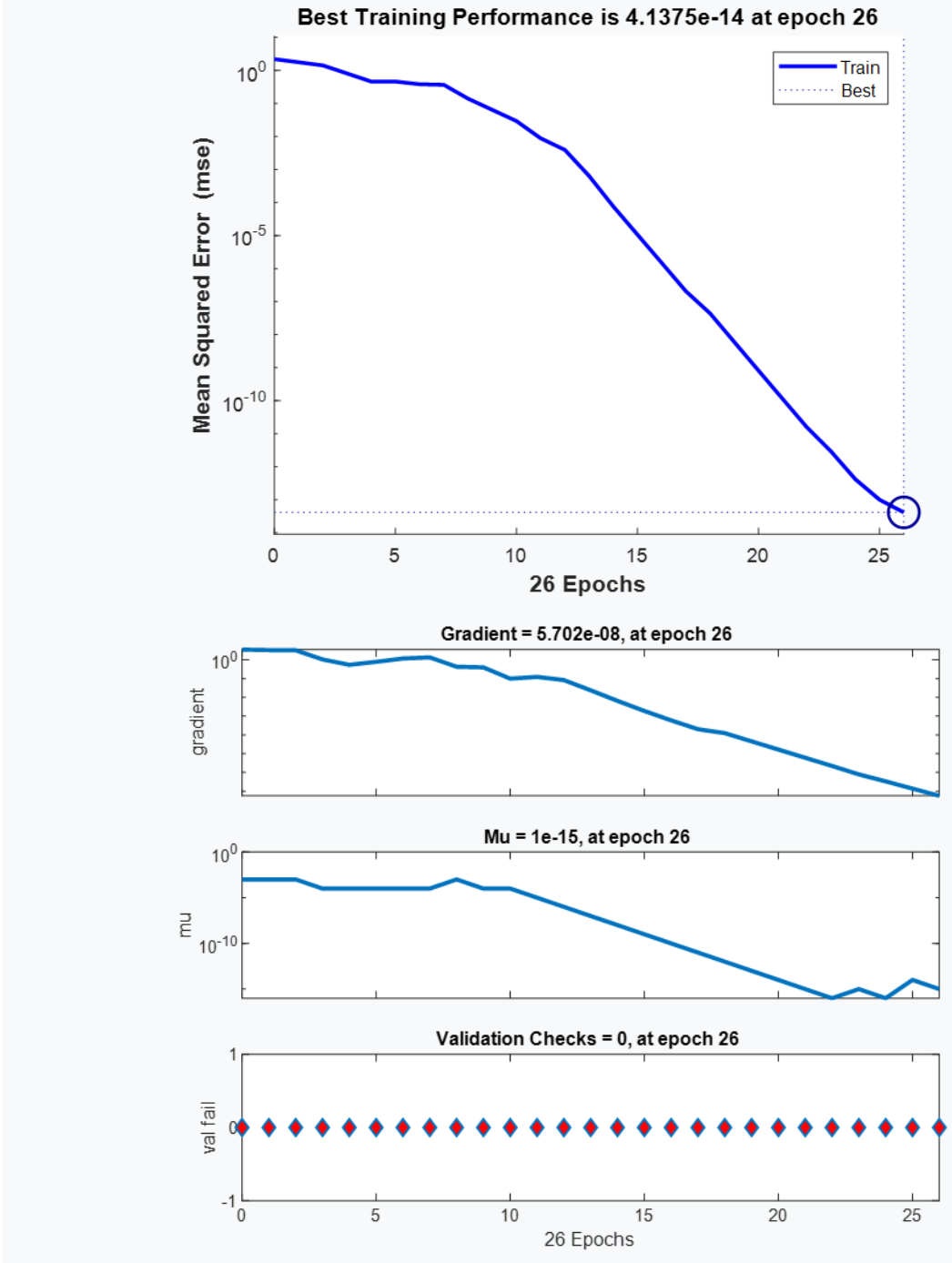
```
% визначити вхідні дані (поєднати зразки з усіх чотирьох
класів)
P = [A B C D];
% визначити цілі
T = [repmat(a,1,length(A)) repmat(b,1,length(B)) ...
     repmat(c,1,length(C)) repmat(d,1,length(D)) ];
```

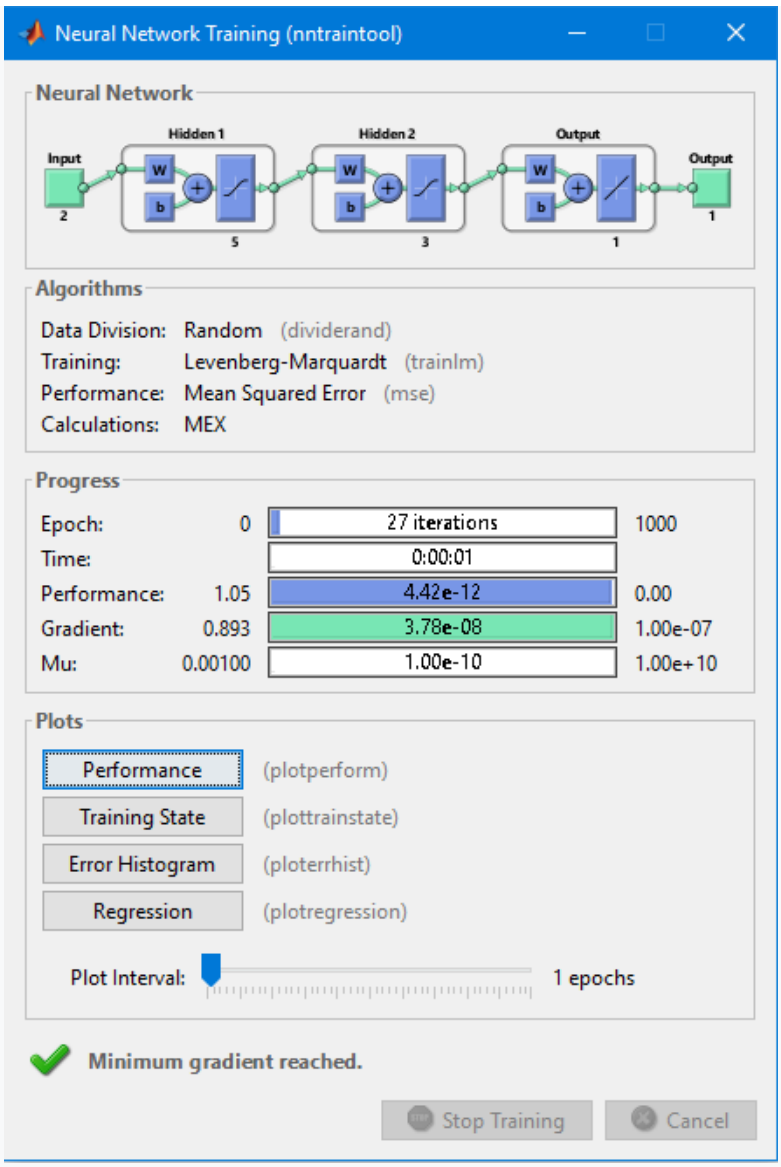
Визначити вхідні дані (поєднати зразки з усіх чотирьох класів)

```
P = [A B C D];
% визначити цілі
T = [repmat(a,1,length(A)) repmat(b,1,length(B)) ...
     repmat(c,1,length(C)) repmat(d,1,length(D)) ];
```

Створіть і навчіть багатошаровий персептрон

```
% створити нейронну мережу
net = feedforwardnet([5 3]);
% параметри навчання
net.divideParam.trainRatio = 1; % training set [%]
net.divideParam.valRatio   = 0; % validation set [%]
net.divideParam.testRatio  = 0; % test set [%]
% навчання
[net,tr,Y,E] = train(net,P,T);
% показати мережу
view(net)
```





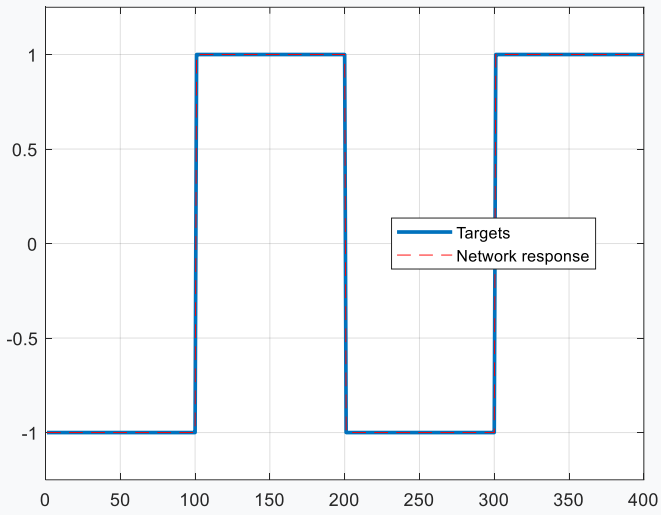

```
net = feedforwardnet(hiddenSizes,trainFcn)
```

повертає пряму нейронну мережу з розміром прихованого шару hiddenSizes і функцією навчання, визначеною trainFcn. Мережі прямого зв'язку складаються з ряду рівнів. Перший рівень має підключення з мережевого входу. Кожен наступний шар має з'єднання з попереднім шаром. Останній рівень створює мережевий вихід.

Побудувати цілі та відповідь мережі, щоб побачити, наскільки добре мережа вивчає дані

```
figure(2)
plot(T','linewidth',2)
hold on
plot(Y','r--')
grid on
legend('Targets','Network response','location','best')
ylim([-1.25 1.25])
```

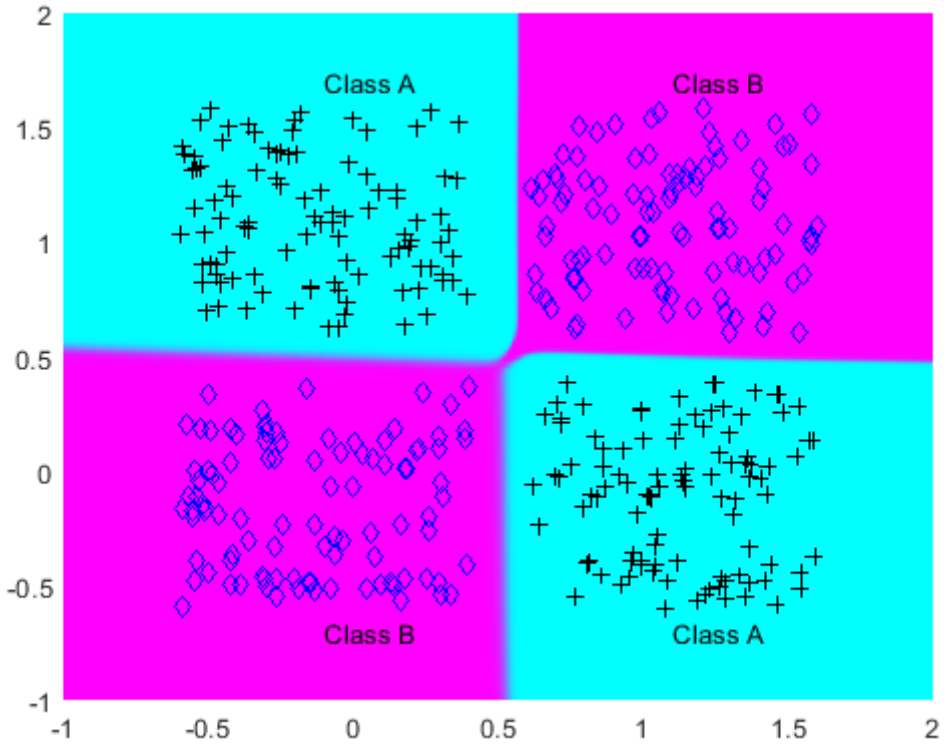
Можливі варіанти функції навчання



Training Function	Algorithm
'trainlm'	Levenberg-Marquardt
'trainbr'	Bayesian Regularization
'trainbfg'	BFGS Quasi-Newton
'trainrp'	Resilient Backpropagation
'trainscg'	Scaled Conjugate Gradient
'traincgb'	Conjugate Gradient with Powell/Beale Restarts
'traincgf'	Fletcher-Powell Conjugate Gradient
'traincgp'	Polak-Ribiére Conjugate Gradient
'trainoss'	One Step Secant
'traingdx'	Variable Learning Rate Gradient Descent
'traingdm'	Gradient Descent with Momentum
'traingd'	Gradient Descent

Побудуйте результат класифікації для повного вхідного простору

```
% створити сітку
span = -1:.005:2;
[P1,P2] = meshgrid(span,span);
pp = [P1(:) P2(:)]';
% імітувати нейронну мережу на сітці
aa = net(pp);
% перевести вихід у [-1,1]
%aa = -1 + 2*(aa>0);
% побудова класифікації ділянок
figure(1)
mesh(P1,P2,reshape(aa,length(span),length(span))-5);
colormap cool
view(2)
```

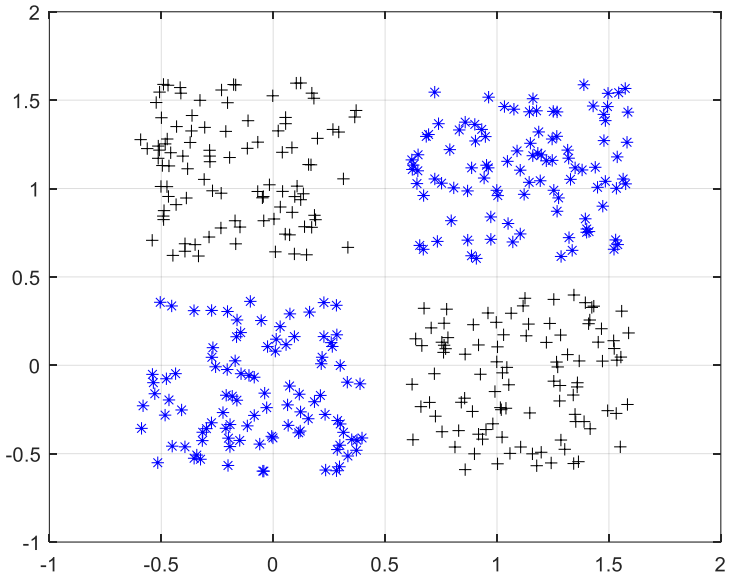


RBFN мережа

Мережа RBF (radial-based function), як і більшість інших нейромереж, призначена для апроксимації функцій, які задані в неявному вигляді набором шаблонів (навчальних образів). Теорема Ковера (Cover) говорить про те, що для набору образів у лінійному просторі роздільна здатність апроксиматора підвищується з підвищенням ступеню активаційної функції апроксиматора.

Створення вхідних даних

```
close all, clear all, clc, format compact

K = 100;
% зміщення кластерів
q = .6;

% визначити 2 групи вхідних даних
A = [rand(1,K)-q rand(1,K)+q;
     rand(1,K)+q rand(1,K)-q];
B = [rand(1,K)+q rand(1,K)-q;
     rand(1,K)+q rand(1,K)-q];

% plot data
plot(A(1,:),A(2,:),'k+',B(1,:),B(2,:),'b*')
grid on
hold on
```



Визначення класів

```
a = -1;
b =  1;
```

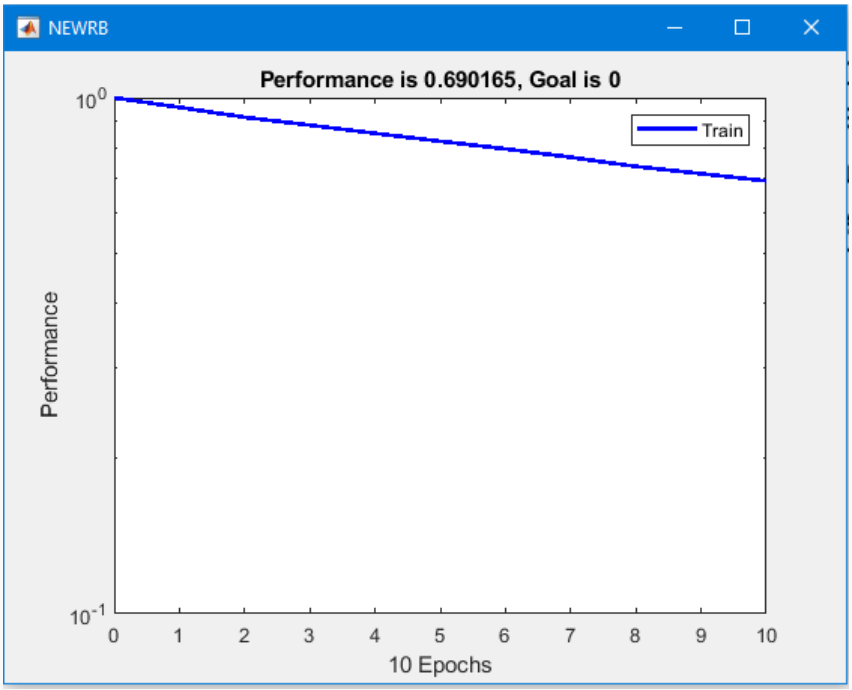
Підготуйте входи та виходи для навчання мережі

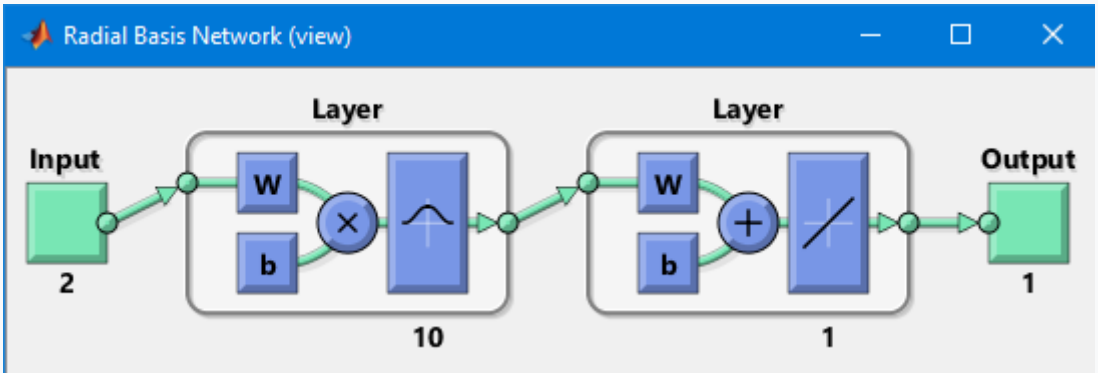
```
% Вхідний вектор
P = [A B];
% Вектор бажаних значень
T = [repmat(a,1,length(A)) repmat(b,1,length(B))];
```

Створення мережі типу RBFN

```
% зміщення
spread = .1;
% кількість нейронів
K      = 10;
% (SSE)
goal   = 0;
% кількість нейронів прихованого шару
Ki     = 2;
% create a neural network
net    = newrb(P,T,goal,spread,K,Ki);
% view network
view(net)
```

Структура мережі та процес навчання



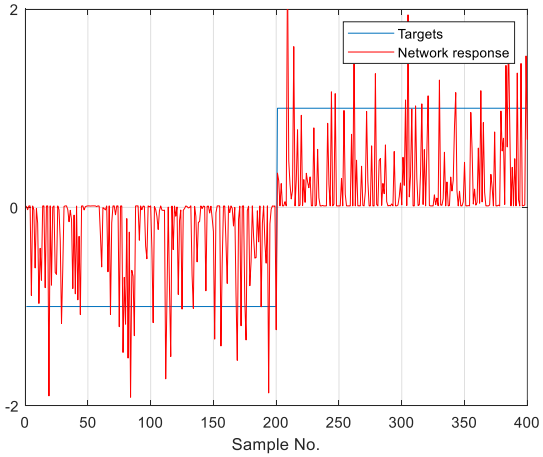


```
NEWRB, neurons = 0, MSE = 1
NEWRB, neurons = 2, MSE = 0.928277
NEWRB, neurons = 4, MSE = 0.855829
NEWRB, neurons = 6, MSE = 0.798564
NEWRB, neurons = 8, MSE = 0.742854
NEWRB, neurons = 10, MSE = 0.690962
```

Оцініть продуктивність мережі

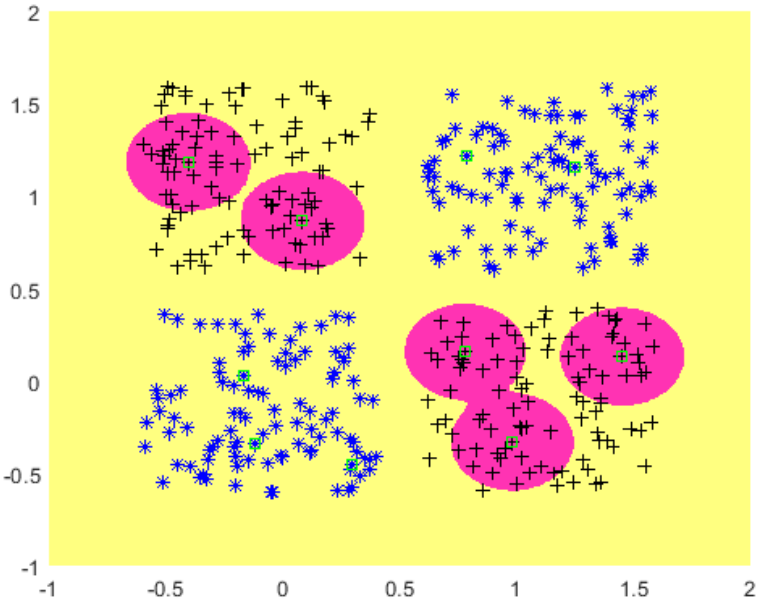
```
% check RBFN spread
actual_spread = net.b{1}
% імітація RBFN на тренувальних даних
Y = net(P);
% обчислити [%] правильних класифікацій
correct = 100 * length(find(T.*Y > 0)) / length(T);
fprintf('\nSpread          = %.2f\n',spread)
fprintf('Num of neurons  = %d\n',net.layers{1}.size)
fprintf('Correct class   = %.2f %%\n',correct)
% побудувати цілі та відповідь мережі, щоб побачити,
наскільки добре мережа вивчає данifigure;
plot(T')
ylim([-2 2])
set(gca,'ytick',[-2 0 2])
hold on
grid on
plot(Y','r')
legend('Targets','Network response')
xlabel('Sample No.')
```

```
Spread          = 0.10
Num of neurons  = 10
Correct class   = 79.75 %
```



Результат класифікації

```
% generate a grid
span    = -1:.025:2;
[P1,P2] = meshgrid(span,span);
pp      = [P1(:) P2(:)]';
% simualte neural network on a grid
aa      = sim(net,pp);
% plot classification regions based on MAX activation
figure(1)
ma = mesh(P1,P2,reshape(-aa,length(span),length(span))-5);
mb = mesh(P1,P2,reshape( aa,length(span),length(span))-5);
set(ma,'facecolor',[1 0.2 .7],'linestyle','none');
set(mb,'facecolor',[1 1.0 .5],'linestyle','none');
view(2)
% plot RBFN centers
plot(net.iw{1}(:,1),net.iw{1}(:,2),'gs')
```



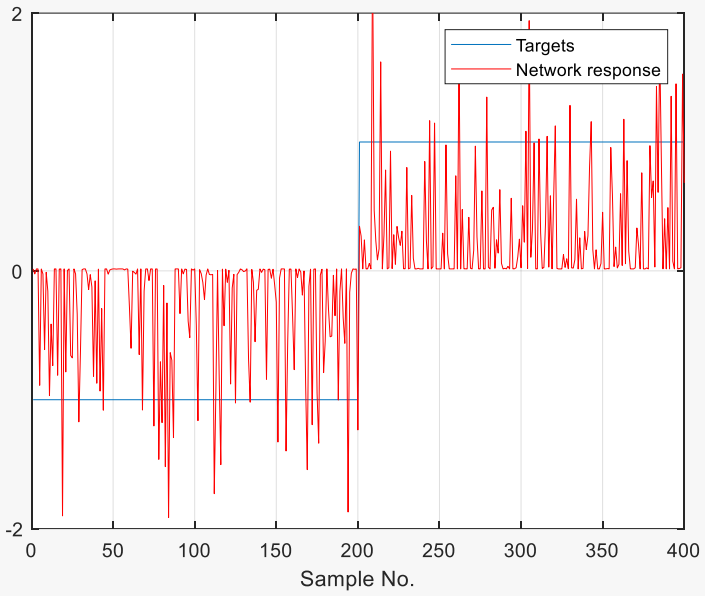
Повторне навчання RBFN за допомогою зворотного поширення помилки за допомогою регуляризації Байєса.

```
% define custom training function: Bayesian regularization
backpropagation
net.trainFcn='trainbr';
```

```
% perform Levenberg-Marquardt training with Bayesian
regularization
net = train(net,P,T);
```

Оцініть продуктивність мережі після байєсівської регуляризації

```
% check new RBFN spread
spread_after_training = net.b{1}
% simulate RBFN on training data
Y = net(P);
% calculate [%] of correct classifications
correct = 100 * length(find(T.*Y > 0)) / length(T);
fprintf('Num of neurons  = %d\n',net.layers{1}.size)
fprintf('Correct class   = %.2f %%\n',correct)
% plot targets and network response
figure;
plot(T')
ylim([-2 2])
set(gca,'ytick',[-2 0 2])
hold on
grid on
plot(Y','r')
legend('Targets','Network response')
xlabel('Sample No.')
```

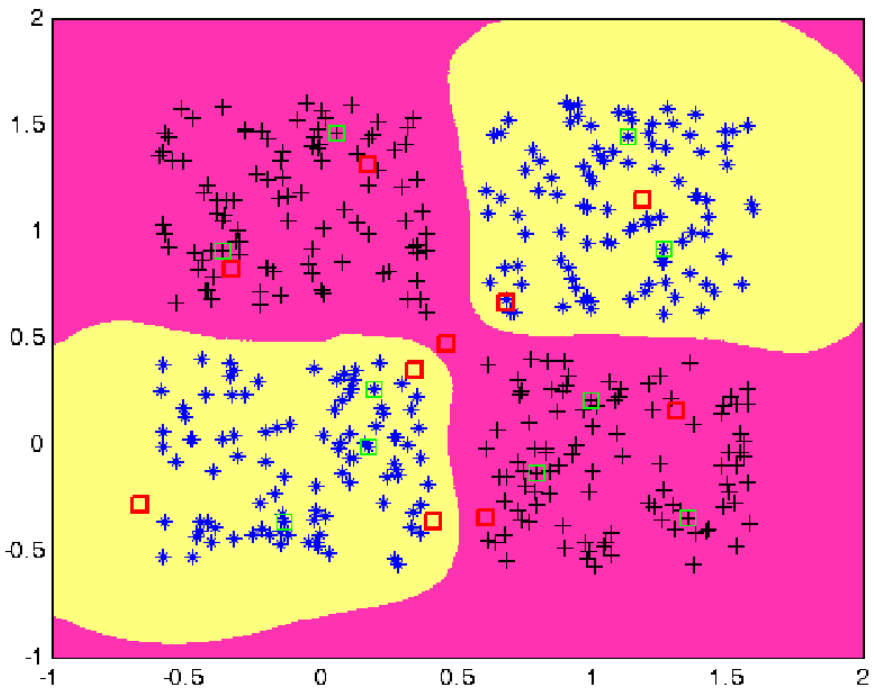



```
spread_after_training =
    8.3255
    8.3255
    8.3255
    8.3255
```

```
            8.3255
            8.3255
            8.3255
            8.3255
            8.3255
            8.3255
        Num of neurons  = 10
        Correct class   = 79.75
```

Результат класифікації після навчання байєсівської регуляризації

```
% simulate neural network on a grid
aa = sim(net,pp);
% plot classification regions based on MAX activation
figure(1)
ma = mesh(P1,P2,reshape(-aa,length(span),length(span))-5);
mb = mesh(P1,P2,reshape( aa,length(span),length(span))-5);
set(ma,'facecolor',[1 0.2 .7],'linestyle','none');
set(mb,'facecolor',[1 1.0 .5],'linestyle','none');
view(2)
% Plot modified RBFN centers
plot(net.iw{1}(:,1),net.iw{1}(:,2),'rs','linewidth',2)
```

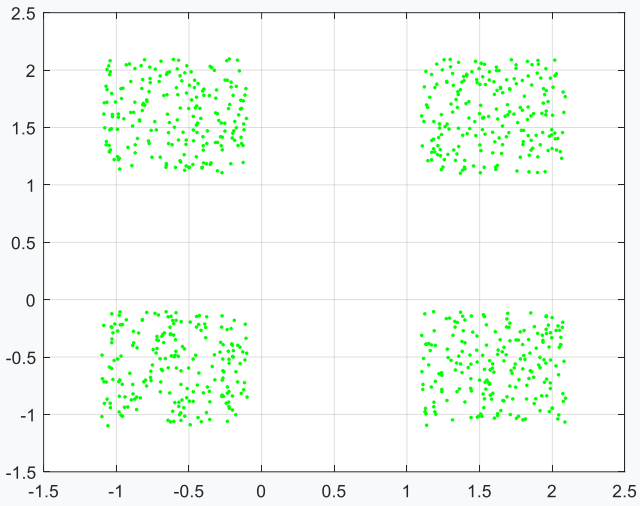


1D і 2D самоорганізована карта

ОПИС ПРОБЛЕМИ: Визначте 1-вимірну та 2-вимірну мережі SOM для представлення 2-вимірного вхідного простору.

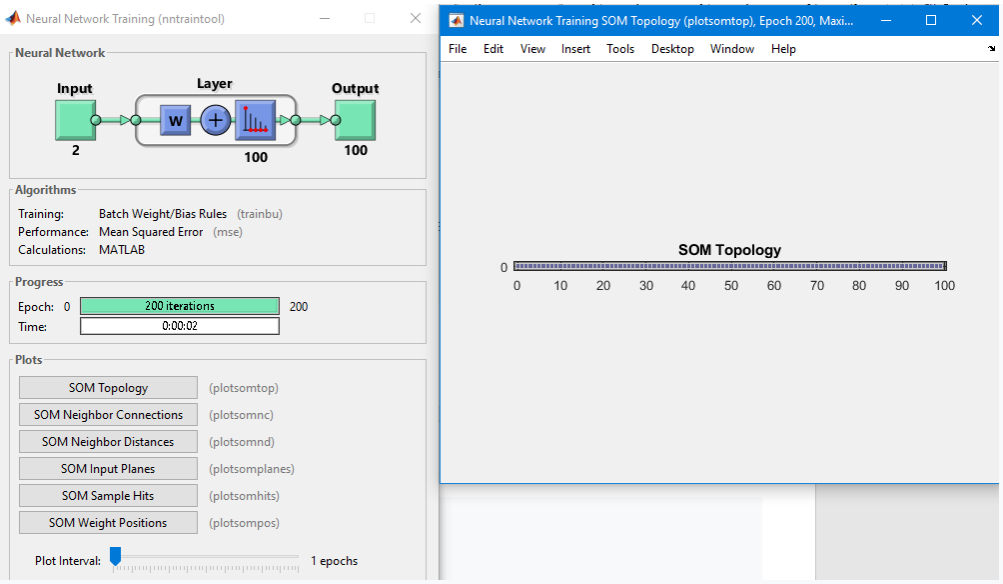
Визначте 4 кластери вхідних даних

```
close all, clear all, clc, format compact
% number of samples of each cluster
K = 200;
% offset of classes
q = 1.1;
% define 4 clusters of input data
P = [rand(1,K)-q rand(1,K)+q rand(1,K)+q rand(1,K)-q;
     rand(1,K)+q rand(1,K)+q rand(1,K)-q rand(1,K)-q];
% plot clusters
plot(P(1,:),P(2,:),'g.')
hold on
grid on
```

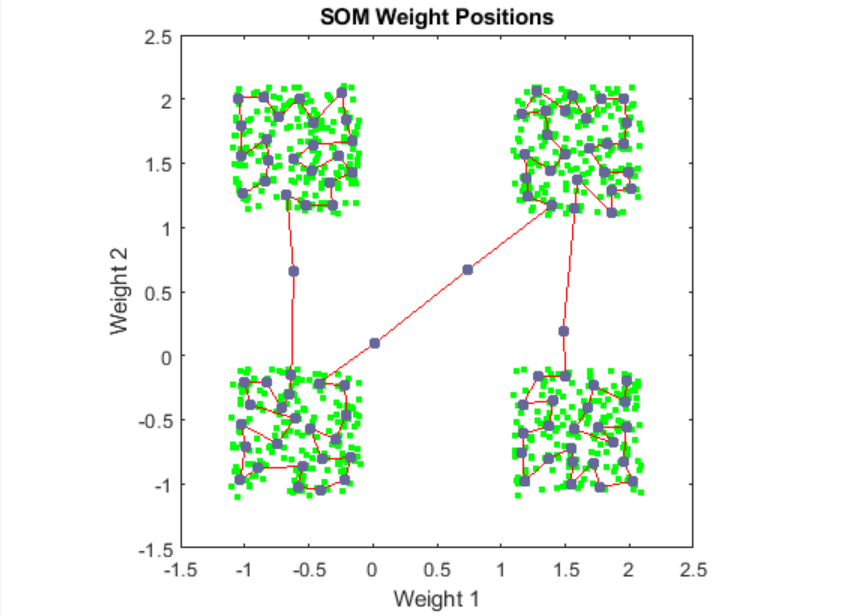


Створення та навчання 1D-SOM

```
% SOM parameters
dimensions   = [100];
coverSteps   = 100;
initNeighbor = 10;
topologyFcn  = 'gridtop';
distanceFcn  = 'linkdist';
% define net
net1 =
selforgmap(dimensions,coverSteps,initNeighbor,topologyFcn,di
stanceFcn);
% train
[net1,Y] = train(net1,P);
```




```
% plot input data and SOM weight positions
plotsompos(net1,P);
grid on
```

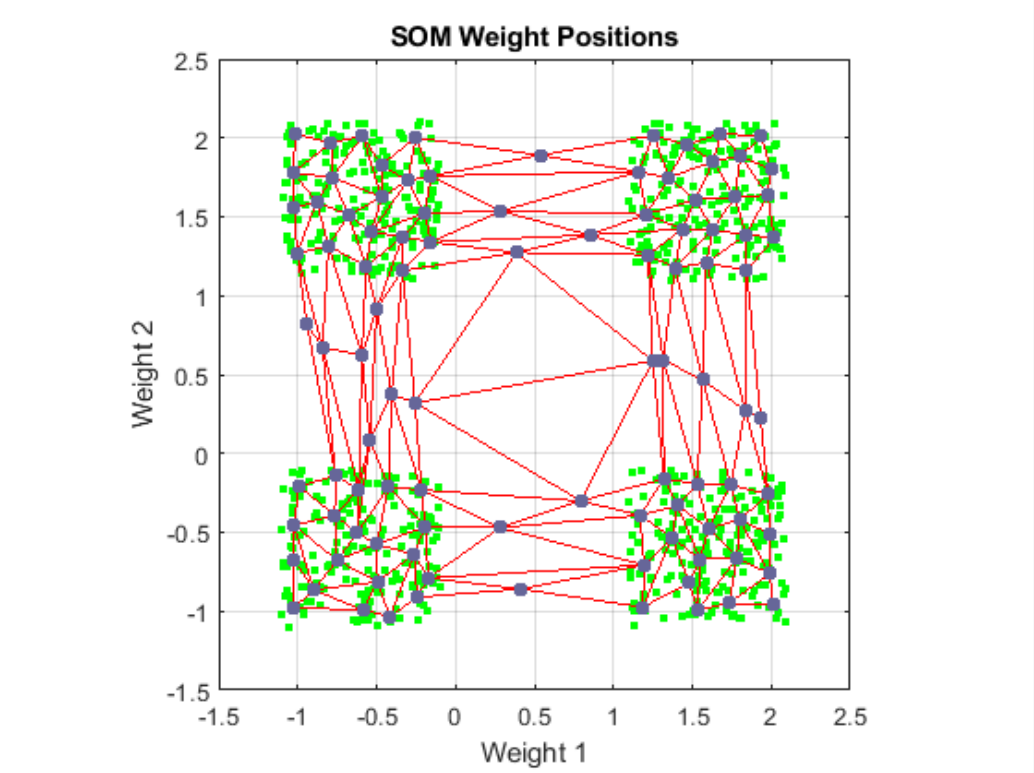


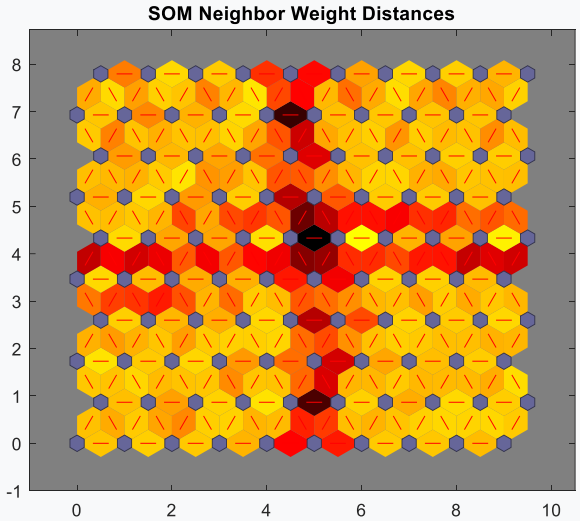
Створення та навчання 2D-SOM

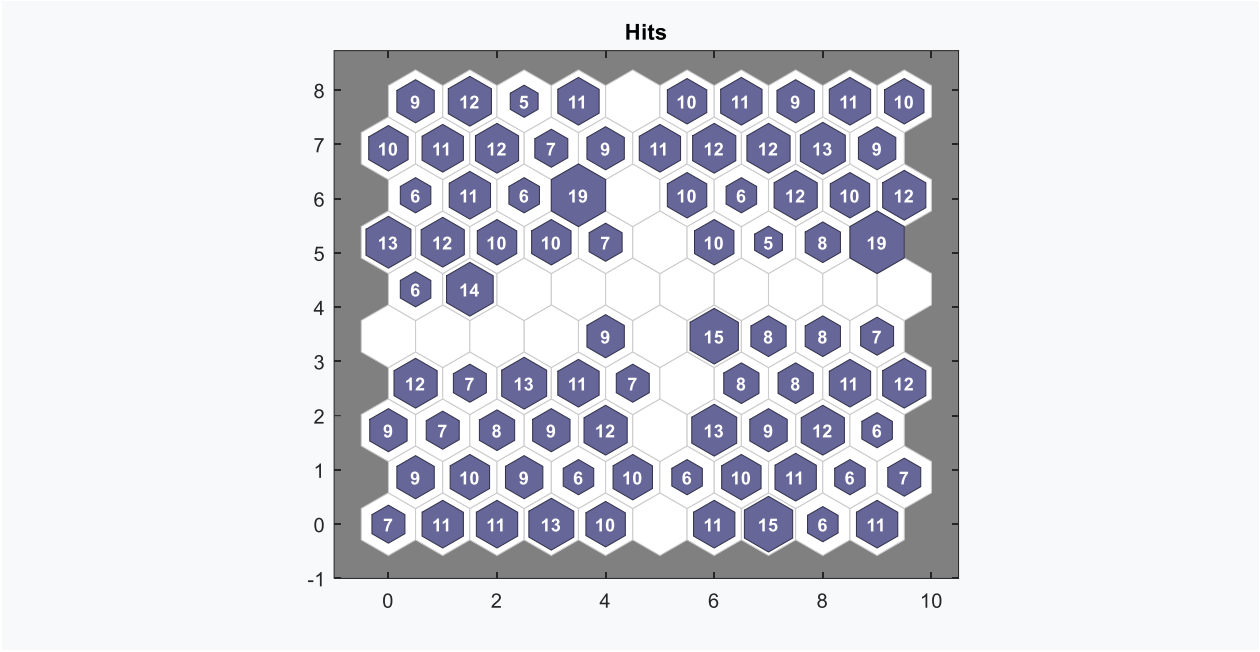
```
% SOM parameters
dimensions   = [10 10];
coverSteps   = 100;
initNeighbor = 4;
topologyFcn  = 'hextop';
distanceFcn  = 'linkdist';
% define net
net2 = selforgmap(dimensions,coverSteps,initNeighbor,topologyFcn,distanceFcn);
% train
[net2,Y] = train(net2,P);
```

побудувати результати 2D-SOM

```
% plot input data and SOM weight positions
plotsompos(net2,P);
grid on
% plot SOM neighbor distances
plotsomnd(net2)
% plot for each SOM neuron the number of input vectors that
it classifies
figure
plotsomhits(net2,P)
```







GoogLeNet

GoogLeNet - це згорткова нейронна мережа, що має глибину 22 шари. Ви можете завантажити попередньо навчену версію мережі, навчену наборам даних ImageNet або Places365. Мережа, навчена ImageNet, класифікує зображення зі 1000 категорій об’єктів, таких як клавіатура, миша, олівець та багато тварин.

Мережа, яка навчається на Places365, подібна до мережі, яка навчається на ImageNet, але класифікує зображення за 365 різними категоріями місць, такими як поле, парк, злітно -посадкова смуга та вестибюль. Ці мережі вивчили різні уявлення функцій для широкого спектра зображень. Обидві попередньо навчені мережі мають розмір вхідного зображення 224 на 224.

Завантажте попередньо навчену мережу GoogLeNet. Для цього кроку необхідний пакет підтримки моделі Deep Learning Toolbox ™ для мережі GoogLeNet. Якщо у вас не встановлено необхідні пакети підтримки, програмне забезпечення надає посилання для завантаження.

Ви також можете завантажити іншу попередньо навчену мережу для класифікації зображень.

Ви можете перенавчити мережу GoogLeNet для виконання нового завдання за допомогою трансферного навчання. Під час виконання трансферного навчання найпоширенішим підходом є використання мереж, попередньо навчених набором даних ImageNet. Якщо нове завдання схоже на класифікацію сцен, то використання мережі, навченої на Places-365, може дати більш високу точність.

Цей приклад демонструє, як використовувати Deep Network Designer, щоб адаптувати попередньо навчену мережу GoogLeNet, щоб класифікувати новий набір зображень. Цей процес називається передачею навчання і зазвичай набагато швидше і легше, ніж навчання нової мережі, тому що можна застосувати вивчені функції до нової задачі за допомогою меншого числа навчальних зображень. Щоб підготувати мережу до передачі навчання в інтерактивному режимі, використовуйте Deep Network Designer.

```
>> net = googlenet; % Завантажуємо GoogLeNet
```

Зображення, яке потрібно класифікувати, має мати той самий розмір, що і вхідний розмір мережі. Для GoogLeNet першим елементом властивості Layers мережі є шар введення зображення. Розмір вхідної мережі - це властивість InputSize шару введення зображення.

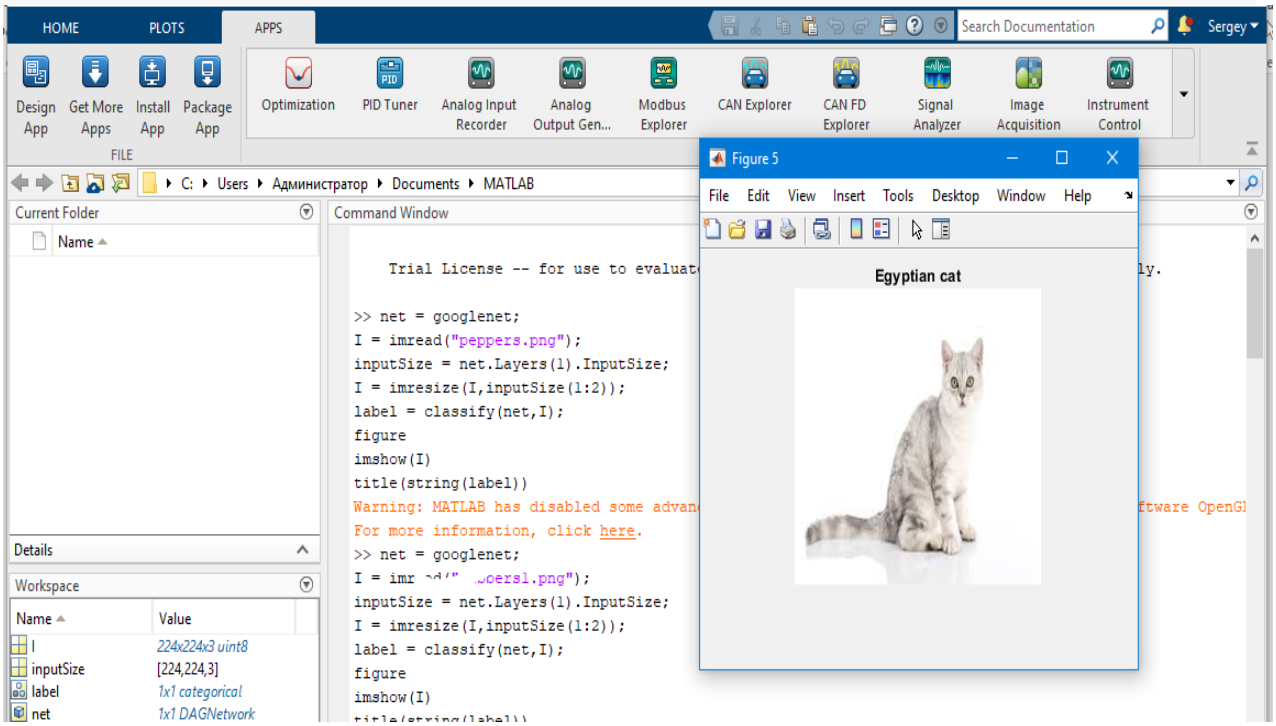
```
I = imread("peppers.png");
inputSize = net.Layers(1).InputSize;
inputSize = 1×3

   224   224     3
I = imresize(I,inputSize(1:2));
```

Остаточним елементом властивості Layers є вихідний шар класифікації. Властивість ClassNames цього шару містить імена класів, вивчених мережею загально. кількістю 1000.

```
label = classify(net,I);
```

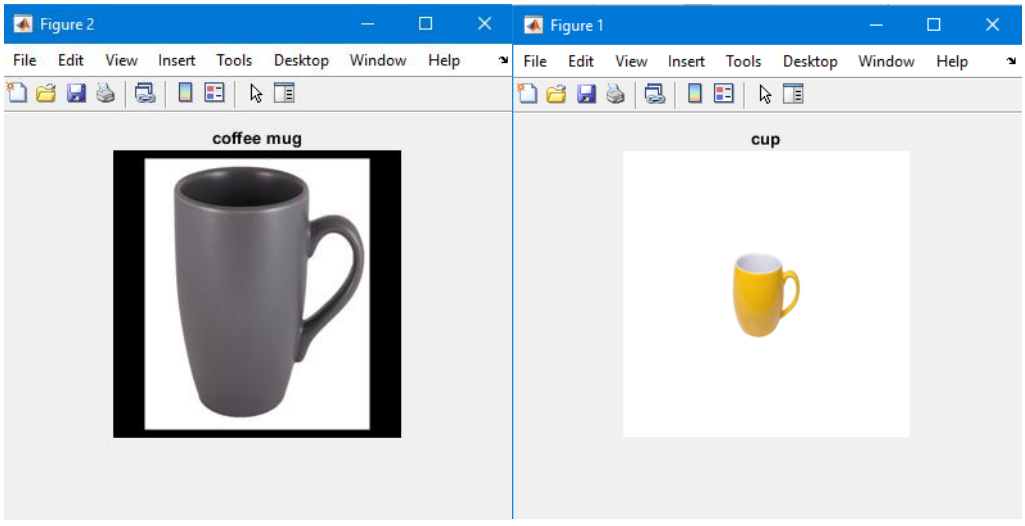
Вивід зображення, яке потрібно класифікувати.

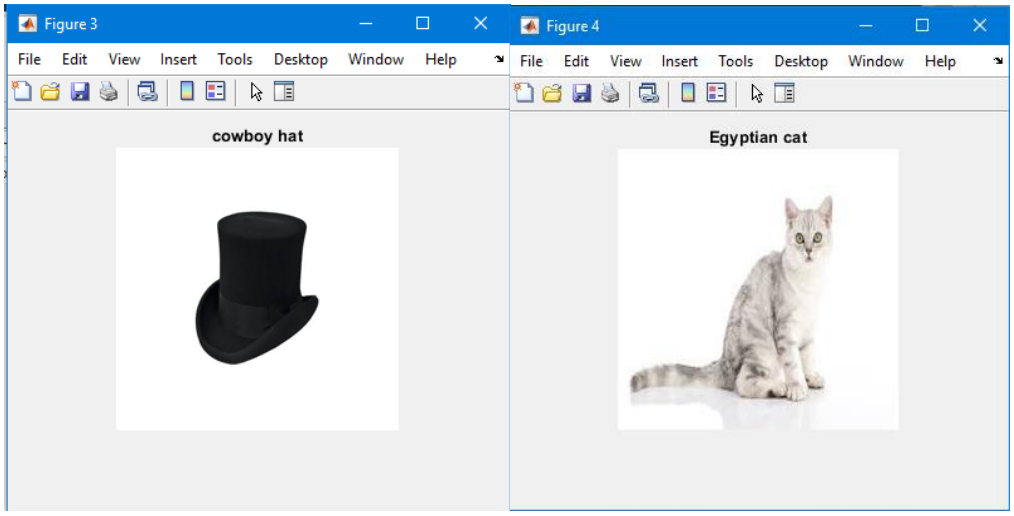
```
figure
imshow(I)
title(string(label))
```



Зображення класифікується та обчислюється ймовірність класу за допомогою класифікації. Мережа правильно класифікує зображення як болгарський перець. Мережа для класифікації навчається виводити одну мітку для кожного вхідного зображення, навіть якщо зображення містить кілька об’єктів.

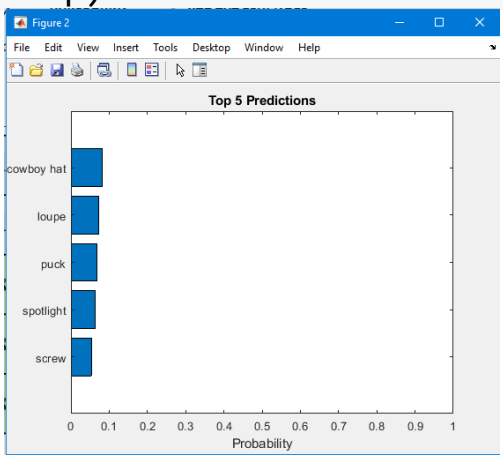
Було вибрано декілька випадкових зображень. Над кожним малюнком записано результат класифікації.





Для виводу на екран ймовірності правильної відповіді виведемо гістограму:

```
[~,idx] = sort(scores,'descend');
idx = idx(5:-1:1);
classNamesTop = net.Layers(end).ClassNames(idx);
scoresTop = scores(idx);

figure
barh(scoresTop)
xlim([0 1])
title('Top 5 Predictions')
xlabel('Probability')
yticklabels(classNamesTop)
```



Трохи ускладнивши програму можна створити нейромережу для розпізнавання об’єктів, які знаходяться поряд за допомогою відеокамери.

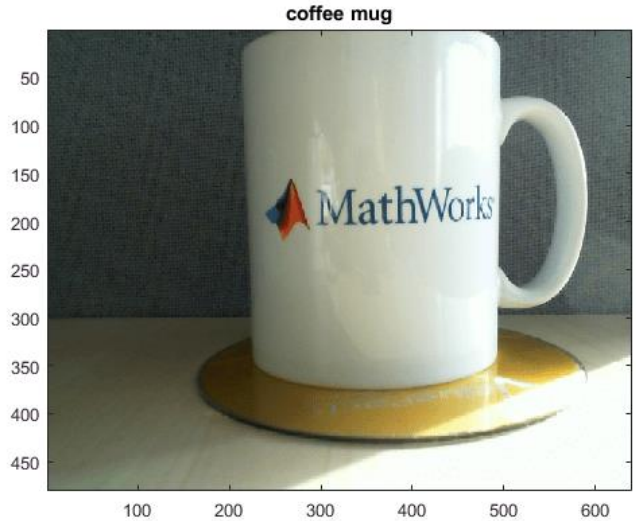
Виконайте ці команди, щоб отримати завантаження, якщо це необхідно, підключитися до веб -камери та отримати попередньо навчену нейронну мережу.

```
camera = webcam; % Connect to the camera
net = alexnet;   % Load the neural network
```

Після встановлення моделі Deep Learning Toolbox ™ для мережі AlexNet можна використовувати її для класифікації зображень. AlexNet - це попередньо навчена згорткова нейронна мережа (CNN), яка пройшла навчання на понад мільйоні зображень і може класифікувати зображення за 1000 категорій об’єктів (наприклад, клавіатура, миша, кухоль для кави, олівець та багато тварин).

Виконайте наступний код, щоб показати та класифікувати живі зображення. Наведіть веб -камеру на об’єкт, і нейромережа повідомляє, який клас об’єктів, на її думку, показує веб -камера. Він буде продовжувати класифікувати зображення, поки ви не натиснете Ctrl+C.

```
while true
    im = snapshot(camera);       % Take a picture
    image(im);                   % Show the picture
    im = imresize(im,[227 227]); % Resize the picture for
alexnet
    label = classify(net,im);    % Classify the picture
    title(char(label));          % Show the class label
    drawnow
end
```



У цьому прикладі мережа правильно класифікує кухоль кави.

Повний лістинг прикладу

```
>> net = googlenet;
I = imread("C:\3.png");
inputSize = net.Layers(1).InputSize;
I = imresize(I,inputSize(1:2));
label = classify(net,I);
figure
imshow(I)
[label,scores] = classify(net,I);
label

label =

  categorical

     cowboy hat

>> figure
imshow(I)
title(string(label) + ", " + num2str(100*scores(classNames
== label),3) + "%");
Unrecognized function or variable 'classNames'.

>> [~,idx] = sort(scores,'descend');
idx = idx(5:-1:1);
classNamesTop = net.Layers(end).ClassNames(idx);
scoresTop = scores(idx);

figure
barh(scoresTop)
xlim([0 1])
title('Top 5 Predictions')
xlabel('Probability')
yticklabels(classNamesTop)
>>
```

Розпізнавання символів

Цей приклад ілюструє, як навчити нейронну мережу, щоб виконати просте розпізнавання символів.

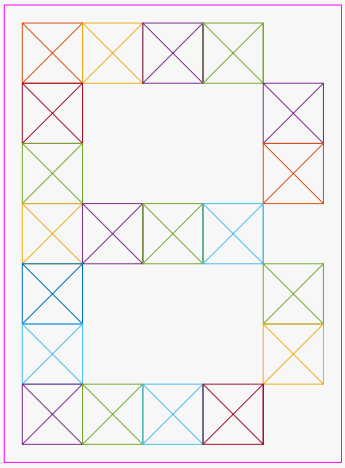
Завдання

Скрипт *prprob* задає матрицю X із 26 стовпцями описуючими літери алфавіту. Кожен стовпець має 35 значень, які можуть бути або 1 або 0. Матриця T 26x26 одинична матриця, яка зіставляє ці 26 векторів вхідних з цими 26 класами.

1	2	3	4	5	6	7	8	9	10	11
0	1	0	1	1	1	0	1	0	1	1
0	1	1	1	1	1	1	0	1	1	0
1	1	1	1	1	1	1	0	1	1	0
0	1	1	1	1	1	1	0	1	1	0
0	0	0	0	1	1	0	1	0	1	1
0	1	1	1	1	1	1	1	0	0	1
1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	0	0	1

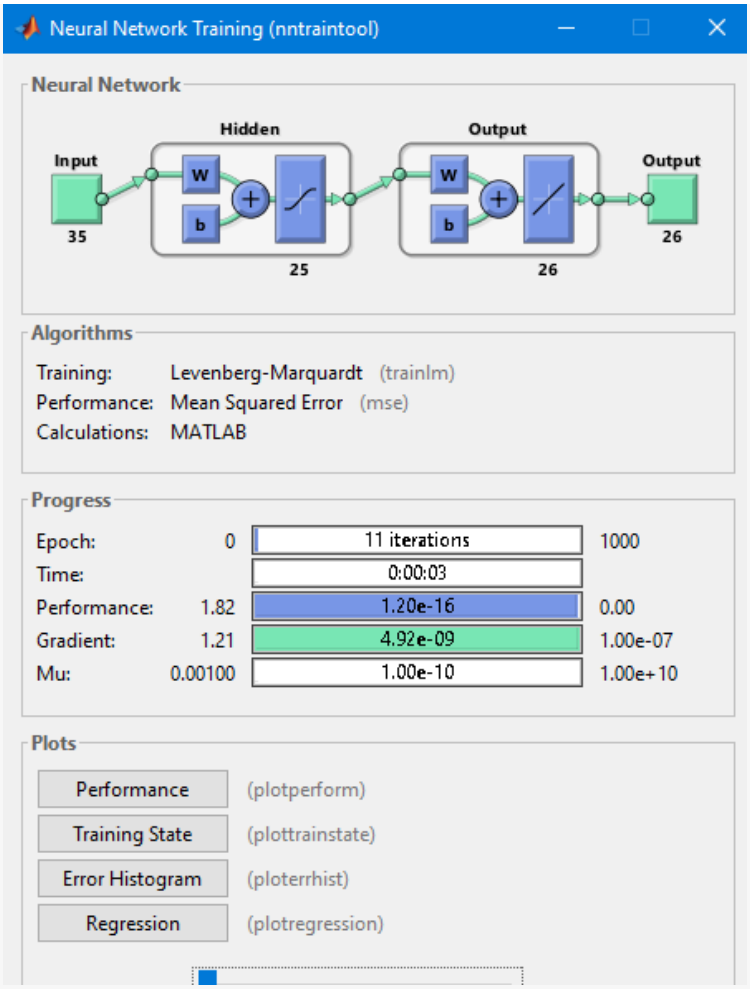
Виведемо другу літеру.

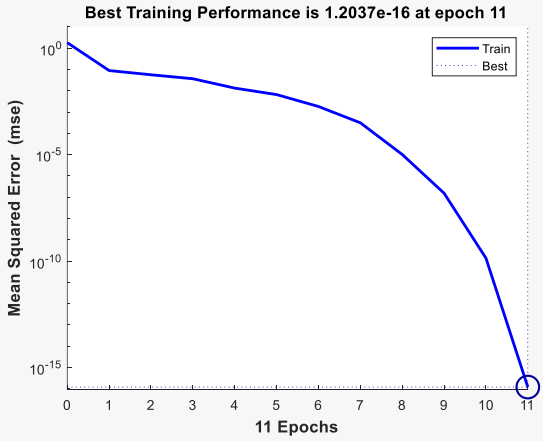
```
[X,T] = prprob;
plotchar(X(:,2))
setdemorandstream(pi);
```

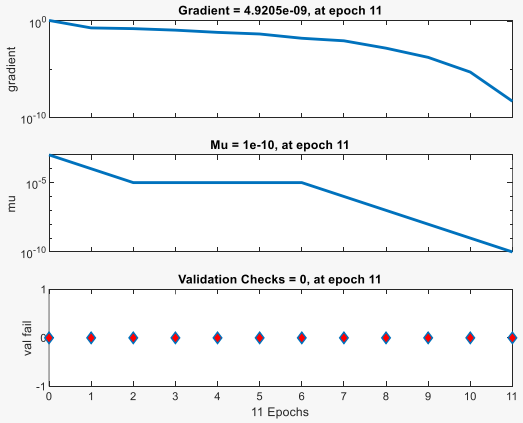


Далі створимо та навчимо нейронну мережу

```
net1 = feedforwardnet(25);
view(net1)
net1.divideFcn = '';
net1 = train(net1,X,T,nnMATLAB);
```

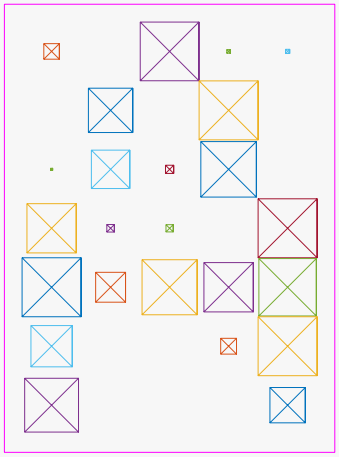






Створимо “зашумлені” версії літер та мережу для їх розпізнавання.

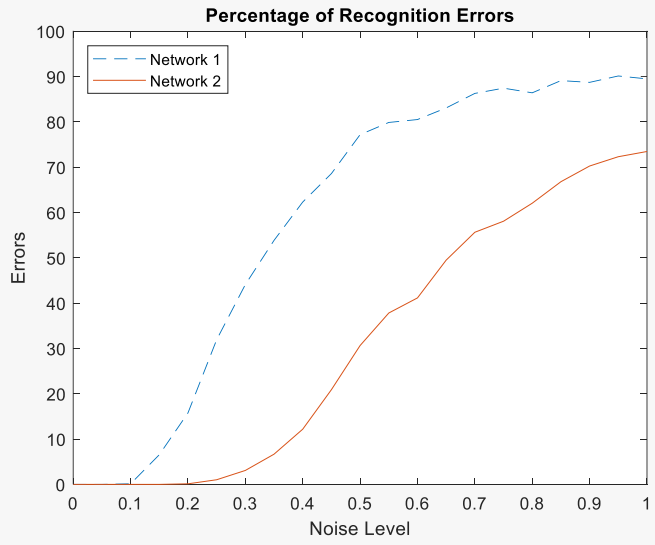
```
numNoise = 30;
Xn = min(max(repmat(X,1,numNoise)+randn(35,26*numNoise)*0.2,0),1);
Tn = repmat(T,1,numNoise);

figure
plotchar(Xn(:,1))
```



```
net2 = feedforwardnet(25);
net2 = train(net2,Xn,Tn,nnMATLAB);
```

Тестування обох нейронних мереж

```
noiseLevels = 0:.05:1;
numLevels = length(noiseLevels);
percError1 = zeros(1,numLevels);
percError2 = zeros(1,numLevels);
for i = 1:numLevels
Xtest =
min(max(repmat(X,1,numNoise)+randn(35,26*numNoise)*noiseLevels(i),0),1);
  Y1 = net1(Xtest);
  percError1(i) = sum(sum(abs(Tn-compet(Y1))))/(26*numNoise*2);
  Y2 = net2(Xtest);
  percError2(i) = sum(sum(abs(Tn-compet(Y2))))/(26*numNoise*2);
end
```



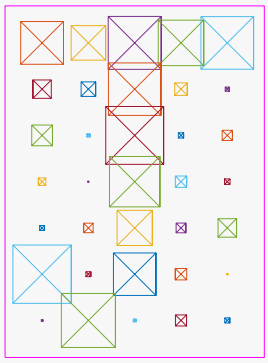

```
figure
```

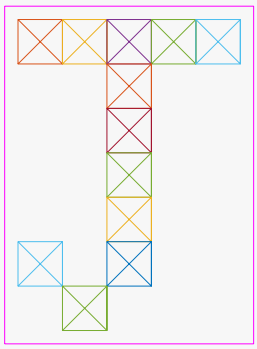
```
plot(noiseLevels,percError1*100,'--
',noiseLevels,percError2*100);
title('Percentage of Recognition Errors');
xlabel('Noise Level');
ylabel('Errors');
legend('Network 1','Network 2','Location','NorthWest')
>>
```

Як бачимо, зашумлена мережа навчається з більшими помилками ніж звичайними. Перевіримо функціонування мережі.

Сформуємо зашумлений вектор для символу J

```
noisyJ = X(:,10)+randn(35,1)*0.2;
plotchar(noisyJ);
A2 = sim(net1,noisyJ);
A2 = compet(A2);
answer = find(compet(A2)) ==1
plotchar(X(:,answer));
```





Після аналізу мережа видає розпізнаний символ.

Обробка фото за допомогою навченої нейронної мережі

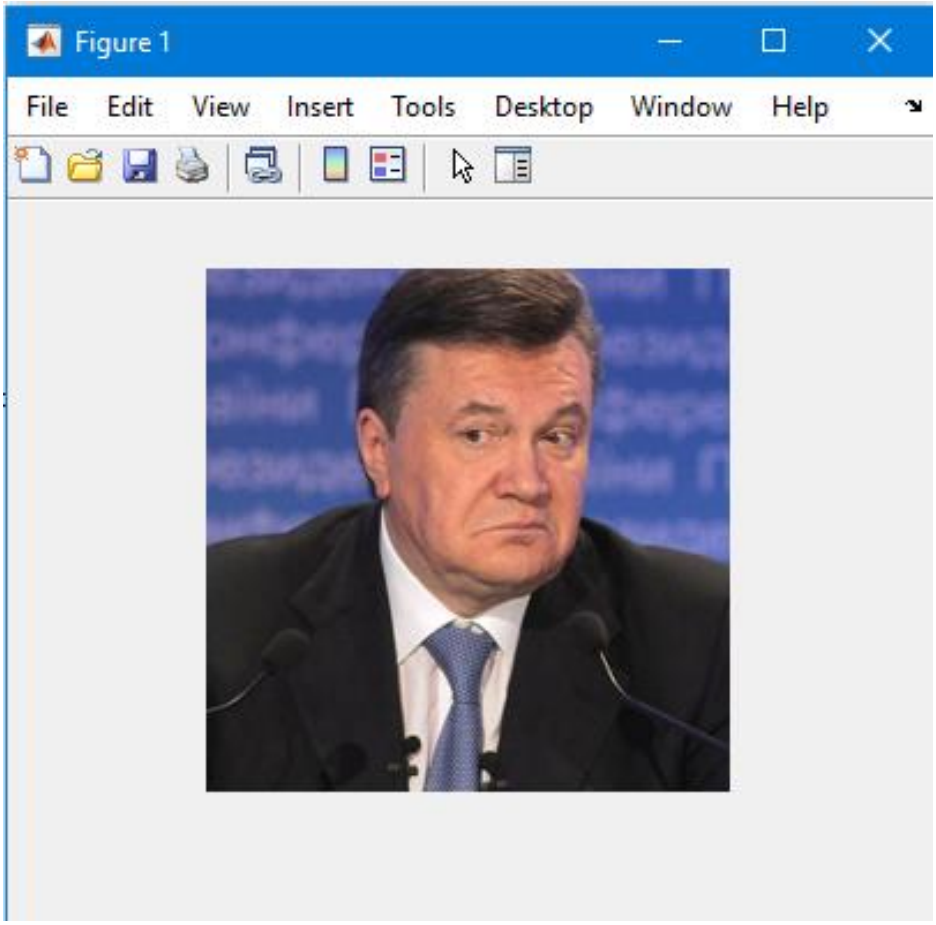
Техніка LIME апроксимує класифікаційну поведінку мережі, використовуючи більш просту та інтерпретовану модель. Створюючи синтетичні дані з вхідних X даних, класифікуючи синтетичні дані за допомогою мережі, а потім використовуючи результати для відповідності простої регресійної моделі, функція imageLIME визначає важливість кожної функції X для оцінки класифікації мережі для класу, заданого міткою.

Використовується вже навчена мережа squeezenet. Підключаємося до мережі.

```
net = squeezenet;
```

Імпортуйте зображення та змініть розмір відповідно до вхідного розміру для мережі.

```
X = imread("C:\face.jpg");
inputSize = net.Layers(1).InputSize(1:2);
X = imresize(X,inputSize);
imshow(X)
```



Класифікуйте зображення, щоб отримати позначку класу.

```
label = classify(net,X)
        label =
          categorical
             suit
```

Використовуйте imageLIME, щоб визначити, які частини зображення важливі для результату класифікації.

```
scoreMap = imageLIME(net,X,label);
figure
imshow(X)
hold on
imagesc(scoreMap,'AlphaData',0.5)
colormap jet
```

Нанесіть результат на вихідне зображення з прозорістю, щоб побачити, які області зображення впливають на оцінку класифікації.

Мережа фокусується переважно на голові та спині, щоб прийняти рішення щодо класифікації. Око та вухо також важливі для результату класифікації.

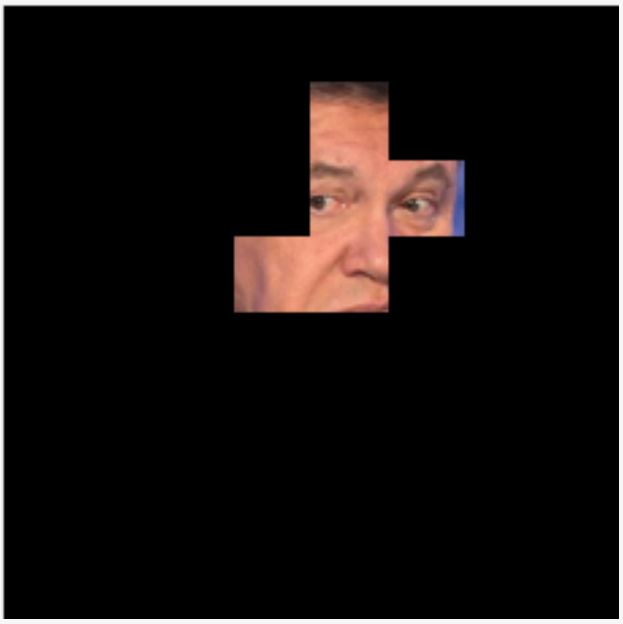
Обчислюється карта важливості сегментів. Встановлюється метод сегментації зображення на "сітку", кількість функцій 64, а кількість синтетичних зображень 3072.

```
[scoreMap,featureMap,featureImportance]  =
imageLIME(net,X,label,'Segmentation','grid','NumFeatures',64
,'NumSamples',3072);
```

Використовується важливість функції, щоб знайти індекси п’яти класів.

```
numTopFeatures = 5;
[~,idx] = maxk(featureImportance,numTopFeatures);
```

```
mask = ismember(featureMap,idx);
maskedImg = uint8(mask).*X;
figure
imshow(maskedImg);
```



Створіть матрицю, що визначає спеціальну карту сегментації, яка ділить зображення на трикутні сегменти. Кожен трикутний сегмент представляє об’єкт. Почніть із визначення матриці, розмір якої дорівнює вхідному розміру зображення.

```
segmentationMap = zeros(inputSize(1));
```

Потім створіть меншу карту сегментації, яка ділить область розміром 56 на 56 пікселів на дві трикутні елементи. Призначте значення 1 і 2 верхньому і нижньому сегментам, що представляють першу і другу ознаки відповідно.

```
blockSize = 56;

segmentationSubset = ones(blockSize);
segmentationSubset = tril(segmentationSubset) +
segmentationSubset;
% Set the diagonal elements to alternate values 1 and 2.
segmentationSubset(1:(blockSize+1):end) = repmat([1 2],1,blockSize/2)';
```

Щоб створити спеціальну карту сегментації для всього зображення, повторіть малу карту сегментації. Кожного разу, коли ви повторюєте меншу

карту, збільшуйте значення індексу об’єкта, щоб пікселі в кожному трикутному сегменті відповідали унікальному об’єкту. У кінцевій матриці значення 1 вказує на першу ознаку, значення 2 – на другу ознаку і так далі для кожного сегмента зображення.

```
blocksPerSide = inputSize(1)/blockSize;
subset = 0;
for i=1:blocksPerSide
    for j=1:blocksPerSide
        xidx = (blockSize*(i-1))+1:(blockSize*i);
        yidx = (blockSize*(j-1))+1:(blockSize*j);
        segmentationMap(xidx,yidx) = segmentationSubset +
2*subset;
        subset = subset + 1;
    end
end

figure
imshow(X)
hold on
imagesc(segmentationMap,'AlphaData',0.8);
title('Custom Segmentation Map')
colormap gray
```


```
scoreMap = imageLIME(net,X,label, ...
    'Segmentation',segmentationMap);
```

Використовуйте imageLIME із спеціальною картою сегментації, щоб визначити, які частини зображення є найважливішими для результату класифікації.

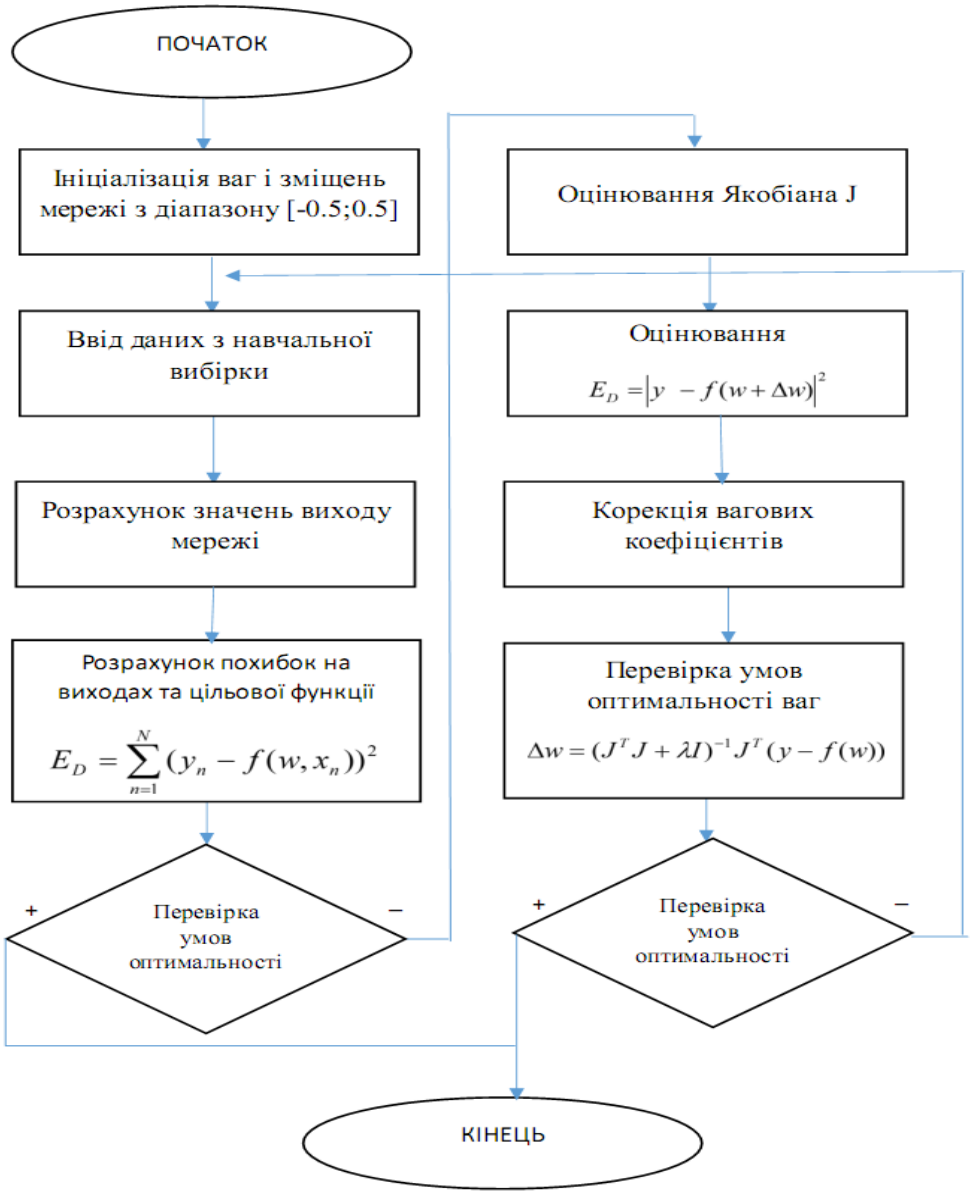
```
figure;
imshow(X)
hold on
title('Image LIME (Golden Retriever)')
colormap jet;
imagesc(scoreMap, "AlphaData", 0.5);
```

Накладення градієнта

```
>> scoreMap = gradCAM(net,X,label);
figure
imshow(X)
hold on
imagesc(scoreMap,'AlphaData',0.5)
colormap jet
```

Апроксимація набору даних C#.

Серед багатьох алгоритмів навчання ШНМ для задач апроксимації використовують алгоритм Левенберга-Марквардта (LM).



Алгоритм Левенберга-Марквардта

Приклад програмної реалізації з використанням C#. Програмна реалізація виконувалась з використанням бібліотек [4]:

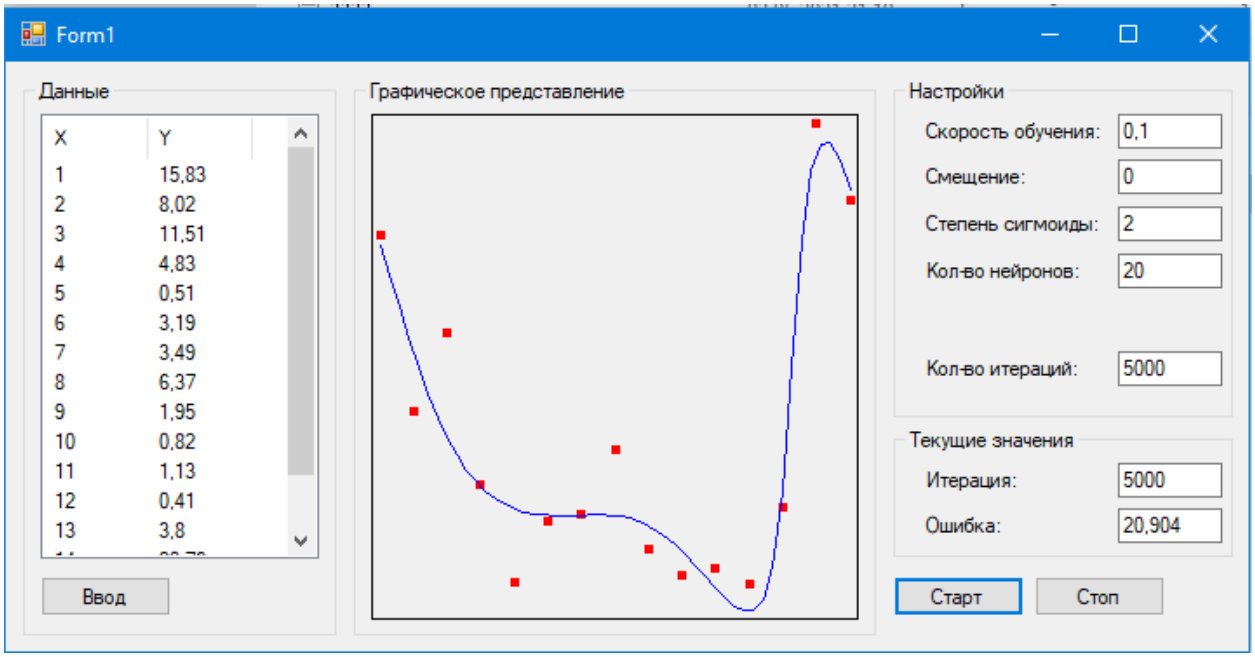
AForge.dll (динамічне компонування);

AForge.Controls.dll (елементи управління);

AForge.Neuro.dll (нейронні мережі).

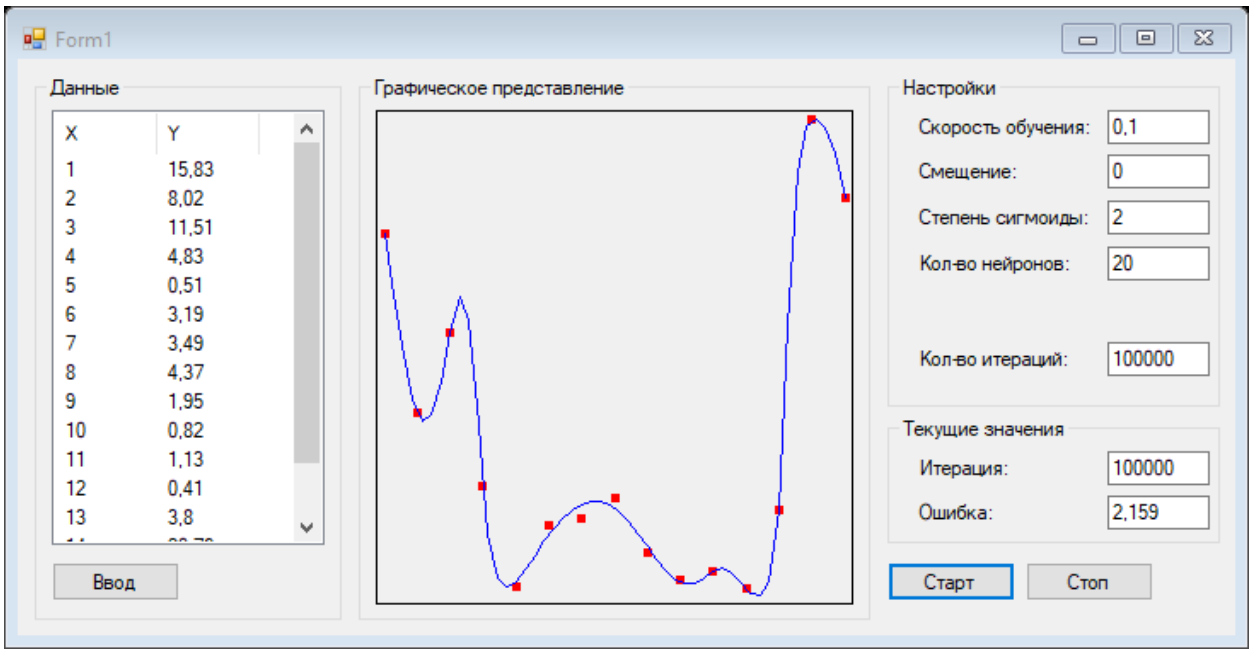
Файл AForge.dll вважається різновидом DLL-файлу. DLL-файли, такі як AForge.dll, по суті є довідником, що зберігають інформацію та інструкції для виконуваних файлів (EXE-файлів), наприклад MpSigStub.exe. Дані файли були створені для того, щоб різні програми (наприклад, Free Webcam Recorder) мали загальний доступ до файлу AForge.dll для більш ефективного розподілу пам'яті, що в свою чергу сприяє підвищенню швидкодії комп'ютера.

AForge.Neuro.dll керований алгоритм навчання нейронної мережі, який базується на генетичних алгоритмах. Для даної нейронної мережі він створює популяцію хромосом DoubleArrayChromosome, які представляють ваги нейронної мережі. Потім, під час процесу навчання, генетична популяція еволюціонує, і ваги, які представлені найкращою хромосомою, встановлюються на вихідну нейронну мережу.

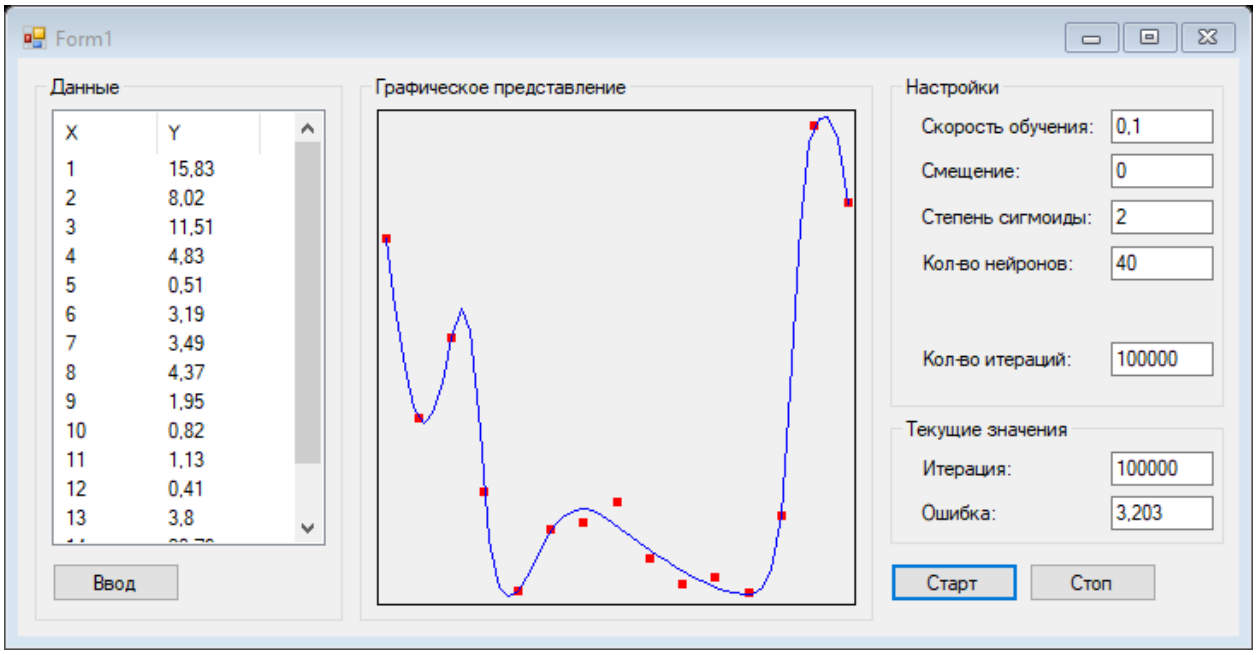


Апроксимація набору даних. Кількість нейронів прихованого шару – 20. Навчання 5000 ітерацій. Помилка 20,904.

В даному прикладі кількість ітерацій недостатня для процесу навчання мережі, що викликає велику помилку апроксимації. Цей недолік ліквідовано в наступних прикладах.



Апроксимація набору даних. Кількість нейронів прихованого шару – 20. Навчання 100000 ітерацій. Помилка 2,159.



Апроксимація набору даних. Кількість нейронів прихованого шару – 40. Навчання 100000 ітерацій. Помилка 3,203.

Можемо звернути увагу на те, що при збільшенні кількості нейронів помилка не зменшується, а навпаки збільшується. Це пояснюється так званим «перенавчанням» мережі. Суть цього поняття ми розглянемо пізніше.

Для порівняння покажемо різницю між апроксимацією в Excel та за допомогою нейромереж.

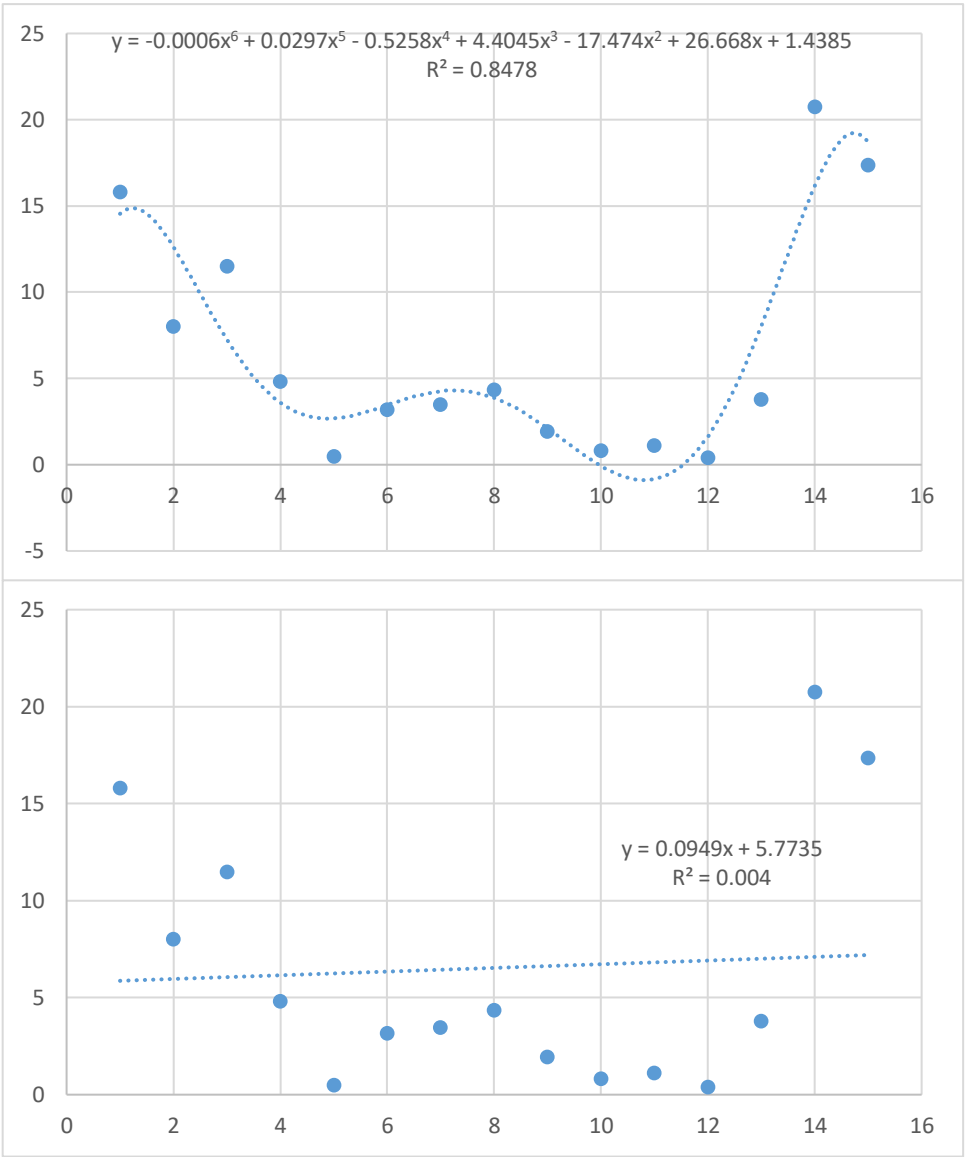


Рис 5. Апроксимація набору даних за допомогою Excel.

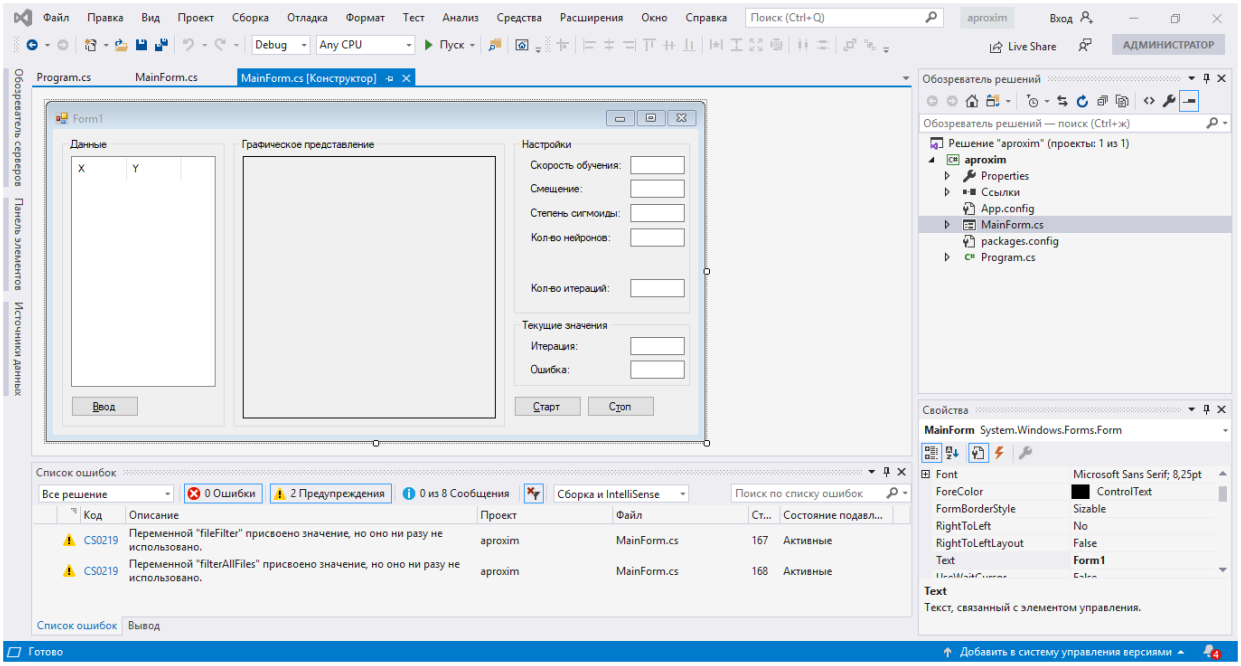
Ми бачимо, що при апроксимації навіть поліномом 6 порядку помилка на порядок вища.

Висновок: було встановлено, що нейронна мережа з одним прихованим й одним вихідним шаром здатна апроксимувати з будь-якою наперед заданою точністю на компактній множині будь-яку неперервну функцію або набір даних.

Програмна реалізація.

```
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Threading.Tasks;
using System.Windows.Forms;

namespace aproxim
{
    static class Program
    {
        /// <summary>
        /// Главная точка входа для приложения.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new MainForm());
        }
    }

}
```




```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using AForge;
using AForge.Neuro;
using AForge.Neuro.Learning;
using AForge.Controls;
using System.Threading;

namespace aproxim
{
    public partial class MainForm : Form
```

```
{
    private double learningRate = 0.1;
    private double momentum = 0.0;
    private double sigmoidAlphaValue = 2.0;
    private int neuronsInFirstLayer = 20;
    private int iterations = 1000;
    private System.Threading.Thread workerThread = null;
    private bool needToStop = false;
    public double[,] data;
    public MainForm()
    {
        InitializeComponent();
        chart.AddDataSeries("data", Color.Red, Chart.SeriesType.Dots, 5);
        chart.AddDataSeries("solution", Color.Blue, Chart.SeriesType.Line, 1);
        UpdateSettings();

    }
    private void EnableControls(bool enable)
    {
        loadDataButton.Enabled              = enable;
        learningRateBox.Enabled             = enable;
        momentumBox.Enabled                 = enable;
        alphaBox.Enabled                    = enable;
        neuronsBox.Enabled                  = enable;
        iterationsBox.Enabled       = enable;
        startButton.Enabled   = enable;
        stopButton.Enabled    = !enable;
    }
    void SearchSolution()
    {
        // number of learning samples

        int samples = data.GetLength(0);
        // data transformation factor
        double yFactor = 1.7 / chart.RangeY.Length;
        double yMin = chart.RangeY.Min;
        double xFactor = 2.0 / chart.RangeX.Length;
        double xMin = chart.RangeX.Min;

        // prepare learning data
        double[][] input = new double[samples][];
        double[][] output = new double[samples][];

        for (int i = 0; i < samples; i++)
        {
            input[i] = new double[1];
            output[i] = new double[1];

            // set input
            input[i][0] = (data[i, 0] - xMin) * xFactor - 1.0;
            // set output
            output[i][0] = (data[i, 1] - yMin) * yFactor - 0.85;
        }

        // create multi-layer neural network
        ActivationNetwork network = new ActivationNetwork(
            new BipolarSigmoidFunction(sigmoidAlphaValue),
            1, neuronsInFirstLayer, 1);
        // create teacher
        BackPropagationLearning teacher = new BackPropagationLearning(network);
        // set learning rate and momentum
        teacher.LearningRate = learningRate;
        teacher.Momentum = momentum;
```

```
            // iterations
            int iteration = 1;

            // solution array
            double[,] solution = new double[50, 2];
            double[] networkInput = new double[1];

            // calculate X values to be used with solution function
            for (int j = 0; j < 50; j++)
            {
                solution[j, 0] = chart.RangeX.Min + (double)j * chart.RangeX.Length / 49;
            }

            // loop
            while (!needToStop)
            {
                // run epoch of learning procedure
                double error = teacher.RunEpoch(input, output) / samples;

                // calculate solution
                for (int j = 0; j < 50; j++)
                {
                    networkInput[0] = (solution[j, 0] - xMin) * xFactor - 1.0;
                    solution[j, 1] = (network.Compute(networkInput)[0] + 0.85) / yFactor
+ yMin;
                }
                chart.UpdateDataSeries("solution", solution);
                // calculate error
                double learningError = 0.0;
                for (int j = 0, k = data.GetLength(0); j < k; j++)
                {
                    networkInput[0] = input[j][0];
                    learningError += Math.Abs(data[j, 1] -
((network.Compute(networkInput)[0] + 0.85) / yFactor + yMin));
                }

                // set current iteration's info
                bool uiMarshal = currentIterationBox.InvokeRequired;
                bool uiMarshall = currentErrorBox.InvokeRequired;
                if (uiMarshal)
                {
                    currentIterationBox.Invoke(new Action(() => {
currentIterationBox.Text = iteration.ToString(); }));
                }
                else
                {
                    currentIterationBox.Text = iteration.ToString();
                }
                if (uiMarshall)
                {
                    currentErrorBox.Invoke(new Action(() => { currentErrorBox.Text =
learningError.ToString("F3"); }));
                }
                else
                {
                    currentErrorBox.Text = learningError.ToString("F3");
                }

                iteration++;

                // check if we need to stop
                if ((iterations != 0) && (iteration > iterations))
                    break;
            }
```

```
    // enable settings controls
    EnableControls(true);
}
private void UpdateSettings()
{

    learningRateBox.Text = learningRate.ToString();
    momentumBox.Text = momentum.ToString();
    alphaBox.Text = sigmoidAlphaValue.ToString();
    neuronsBox.Text = neuronsInFirstLayer.ToString();
    iterationsBox.Text = iterations.ToString();

}
private void loadDataButton_Click(object sender, EventArgs e)
{
    double[,] tempData = new double[50, 2];
    //double[,] data;
    double minX = double.MaxValue;
    double maxX = double.MinValue;
    string filePath = "";
    string fileFilter = "";
    bool filterAllFiles = false;

    try
    {
        //Выбор файла
        OpenFileDialog ofd = new OpenFileDialog();
        if (ofd.ShowDialog() == DialogResult.OK)

        {
            filePath = ofd.FileName;
            String line;
            System.IO.StreamReader file = new StreamReader(filePath);
            int counter = 0;
            //Заполнение массива данными
            while ((line = file.ReadLine()) != null)
            {
                string[] strs = line.Split(';');
                tempData[counter, 0] = double.Parse(strs[0]);
                tempData[counter, 1] = double.Parse(strs[1]);
                // Поиск минимального значения
                if (tempData[counter, 0] < minX)
                    minX = tempData[counter, 0];
                // Поиск максимального значения
                if (tempData[counter, 0] > maxX)
                    maxX = tempData[counter, 0];
                counter++;
            }
            //Заполнение таблицы
            data = new double[counter, 2];
            Array.Copy(tempData, 0, data, 0, counter * 2);
            dataList.Items.Clear();
            for (int i = 0, n = data.GetLength(0); i < n; i++)
            {
                dataList.Items.Add(data[i, 0].ToString());
                dataList.Items[i].SubItems.Add(data[i, 1].ToString());
            }
            // Вывод точек в графический контейнер
            chart.RangeX = new AForge.Range((float)minX, (float)maxX);
            chart.UpdateDataSeries("data", data);
            chart.UpdateDataSeries("solution", null);
        }
```

```
            }
            catch (Exception)
            {
                MessageBox.Show("Не удалось прочитать файл", "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }
            finally
            {
                //  if (filePath != null)
                //filePath.Close();
                //   file.Close();
            }
            //UpdateDataList();

            //return ;
        }//11111111111

        private void startButton_Click(object sender, EventArgs e)
        {

            {
                // get learning rate
                try
                {
                    learningRate = Math.Max(0.00001, Math.Min(1,
double.Parse(learningRateBox.Text)));
                }
                catch
                {
                    learningRate = 0.1;
                }
                // get momentum
                try
                {
                    momentum = Math.Max(0, Math.Min(0.5,
double.Parse(momentumBox.Text)));
                }
                catch
                {
                    momentum = 0;
                }
                // get sigmoid's alpha value
                try
                {
                    sigmoidAlphaValue = Math.Max(0.001, Math.Min(50,
double.Parse(alphaBox.Text)));
                }
                catch
                {
                    sigmoidAlphaValue = 2;
                }
                // get neurons count in first layer
                try
                {
                    neuronsInFirstLayer = Math.Max(5, Math.Min(50,
int.Parse(neuronsBox.Text)));
                }
                catch
                {
                    neuronsInFirstLayer = 20;
                }
                // iterations
                try
```

```
                {
                    iterations = Math.Max(0, int.Parse(iterationsBox.Text));
                }
                catch
                {
                    iterations = 1000;
                }
                // update settings controls
                UpdateSettings();

                // disable all settings controls except "Stop" button
                 EnableControls(false);
                //Application.Exit();
                // run worker thread
                needToStop = false;
                 workerThread = new System.Threading.Thread(new
ThreadStart(SearchSolution));
                workerThread.Start();
            }
        }

        private void stopButton_Click(object sender, EventArgs e)
        {
            needToStop = true;
            //workerThread.Join();
            //workerThread = null;

        }


        //private void EnableControls(bool v)
        //{
        //     throw new NotImplementedException();
        // }
    }
}
```

Задача комівояжера.

Проблема комівояжера складається з продавця та низки міст. Продавець має відвідати кожне з міст, що починаються з певного та повернутися в те саме місто. Проблема полягає в тому, що комівояжер хоче мінімізувати повну тривалість поїздки і знайти найвигідніший шлях. Використовується генетичний алгоритм (метод рою частинок).

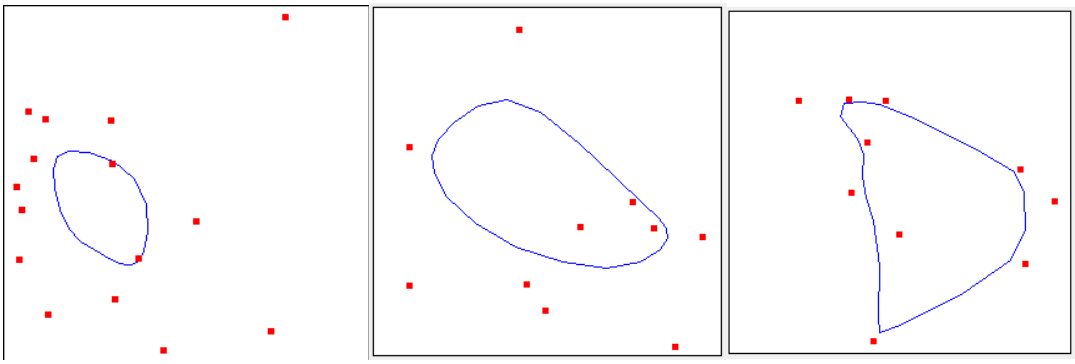
Проблеми пов'язані з використанням математичних задач оптимізації при вирішенні масштабних проблем, внесли свій внесок у розвиток еволюційних обчислювальних алгоритмів. Еволюційні алгоритми імітують природну біологічну еволюцію та соціальну поведінку стохастичних методів пошуку.

Вхідною умовою для розв’язання задачі комівояжера є матриця суміжності орієнтованого повного графу з n вершинами, заповнена відстанями

найкоротших маршрутів між кожною парою вершин. Відповіддю на задачу є перестановка з n вершин, що відповідає порядку обходу вершин знайденого маршруту. Вершини графу будемо називати містами, а будь-яку перестановку, що формує замкнутий маршрут – туром

Починається алгоритм з установки на площині невеликого кола. Воно нерівномірно розширюється, стаючи кільцем, що проходить практично біля всіх міст і встановлюючи таким чином шуканий маршрут. На кожну рухому точку кола впливає дві складові: переміщення точки в бік найближчого міста і зміщення в бік сусідів точки на кільці так, щоб зменшити його довжину. Місто в результаті зв'язується з певною ділянкою кільця в міру розширення. У міру розширення такої еластичної мережі кожне місто виявляється асоційований з певною ділянкою кільця.

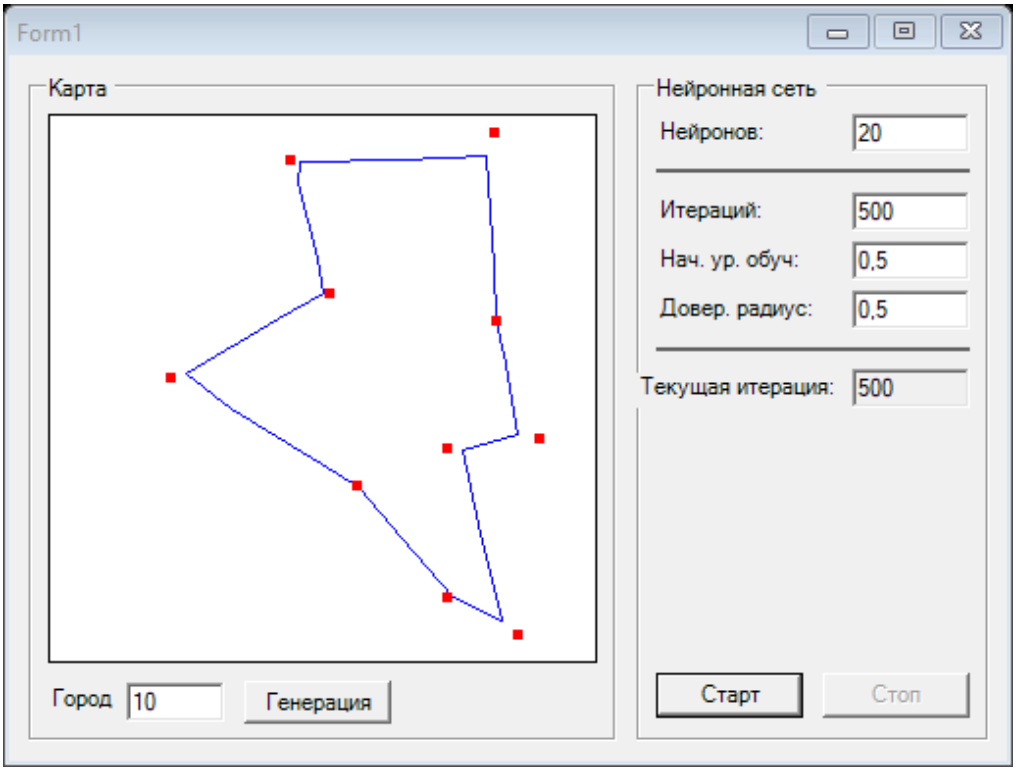
Спочатку все міста надають приблизно однаковий вплив на кожну точку маршруту. В подальшому, великі відстані стають менш впливовими і кожне місто стає більш специфічним для найближчих до нього точок кільця. Таке поступове збільшення специфічності, яке нагадує метод навчання мережі Коханена, контролюється значенням деякого ефективного радіуса.



Дурбін і Уіллшоу показали, що для завдання з 30 містами метод еластичної сітки генерує найкоротший маршрут приблизно за 1000 ітерацій. Для 100 міст знайдений цим методом маршрут лише на 1% перевищував оптимальний.

Програмна реалізація.

Використовувались бібліотеки AForge.dll; AForge.Neuro.dll; AForge. Neuro.Learning.dll; AForge. Controls.dll.




```
/// <summary>
		/// The main entry point for the application.
		/// </summary>
		[STAThread]
		static void Main( )
		{
			Application.Run( new MainForm( ) );
		}

		// On main form closing
		private void MainForm_Closing(object sender,
System.ComponentModel.CancelEventArgs e)
		{
			// check if worker thread is running
			if ( ( workerThread != null ) && ( workerThread.IsAlive ) )
			{
				needToStop = true;
				workerThread.Join( );
			}
		}

		// Update settings controls
		private void UpdateSettings( )
		{
			citiesCountBox.Text  = citiesCount.ToString( );
			neuronsBox.Text            = neurons.ToString( );
			iterationsBox.Text   = iterations.ToString( );
			rateBox.Text         = learningRate.ToString( );
			radiusBox.Text             = learningRadius.ToString( );
		}
```

```
        // Generate new map for the Traivaling Salesman problem
        private void GenerateMap( )
        {
            Random rand = new Random( (int) DateTime.Now.Ticks );

            // create coordinates array
            map = new double[citiesCount, 2];

            for ( int i = 0; i < citiesCount; i++ )
            {
                map[i, 0] = rand.Next( 1001 );
                map[i, 1] = rand.Next( 1001 );
            }

            // set the map
            chart.UpdateDataSeries( "cities", map );
            // erase path if it is
            chart.UpdateDataSeries( "path", null );
        }

        // On "Generate" button click - generate map
        private void generateMapButton_Click(object sender, System.EventArgs e)
        {
            // get cities count
            try
            {
                citiesCount = Math.Max( 5, Math.Min( 50, int.Parse(
citiesCountBox.Text ) ) );
            }
            catch
            {
                citiesCount = 20;
            }
            citiesCountBox.Text = citiesCount.ToString( );

            // regenerate map
            GenerateMap( );
        }

        // Enable/disale controls
        private void EnableControls( bool enable )
        {

        private void startButton_Click(object sender, System.EventArgs e)
        {
            // get network size
            try
            {
                neurons = Math.Max( 5, Math.Min( 50, int.Parse(
neuronsBox.Text ) ) );
            }
            catch
            {
                neurons = 20;
            }
            // get iterations count
            try
            {
                iterations = Math.Max( 10, Math.Min( 1000000, int.Parse(
iterationsBox.Text ) ) );
            }
            catch
            {
                iterations = 500;
```

```
                    }
                    // get learning rate
                    try
                    {
                        learningRate = Math.Max( 0.00001, Math.Min( 1.0, double.Parse(
rateBox.Text ) ) );
                    }
                    catch
                    {
                        learningRate = 0.5;
                    }
                    // get learning radius
                    try
                    {
                        learningRadius = Math.Max( 0.00001, Math.Min( 1.0,
double.Parse( radiusBox.Text ) ) );
                    }
                    catch
                    {
                        learningRadius = 0.5;
                    }
                    // update settings controls
                    UpdateSettings( );

                    // disable all settings controls except "Stop" button
                    EnableControls( false );

                    // run worker thread
                    needToStop = false;
                    workerThread = new Thread( new ThreadStart( SearchSolution ) );
                    workerThread.Start( );
                }

                // On "Stop" button click
                private void stopButton_Click(object sender, System.EventArgs e)
                {
                    // stop worker thread
                    needToStop = true;
                    workerThread.Join( );
                    workerThread = null;
                }

                // Worker thread
                void SearchSolution( )
                {
                    // set random generators range
                    Neuron.RandRange = new DoubleRange( 0, 1000 );

                    // create network
                    DistanceNetwork network = new DistanceNetwork( 2, neurons );

                    // create learning algorithm
                    ElasticNetworkLearning     trainer = new ElasticNetworkLearning(
network );

                    double fixedLearningRate = learningRate / 20;
                    double driftingLearningRate = fixedLearningRate * 19;

                    // path
                    double[,] path = new double[neurons + 1, 2];

                    // input
                    double[] input = new double[2];
```

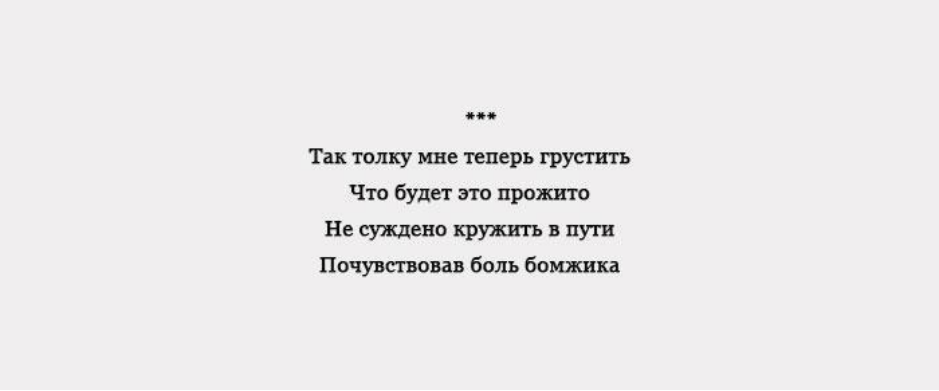
```
                    // iterations
                    int i = 0;

                    // loop
                    while ( !needToStop )
                    {
                        // update learning speed & radius
                        trainer.LearningRate = driftingLearningRate * ( iterations - i
) / iterations + fixedLearningRate;
                        trainer.LearningRadius = learningRadius * ( iterations - i ) /
iterations;

                        // set network input
                        int currentCity = rand.Next( citiesCount );
                        input[0] = map[currentCity, 0];
                        input[1] = map[currentCity, 1];

                        // run one training iteration
                        trainer.Run( input );

                        // show current path
                        for ( int j = 0; j < neurons; j++ )
                        {
                            path[j, 0] = network[0][j][0];
                            path[j, 1] = network[0][j][1];
                        }
                        path[neurons, 0] = network[0][0][0];
                        path[neurons, 1] = network[0][0][1];

                        chart.UpdateDataSeries( "path", path );

                        // increase current iteration
                        i++;

                        // set current iteration's info
                        //currentIterationBox.Text = i.ToString( );
                        currentIterationBox.Invoke((MethodInvoker)(() =>
currentIterationBox.Text = i.ToString()));

                        bool uiMarshall = currentIterationBox.InvokeRequired;
                        if (uiMarshall)
                            if ( i >= iterations )
                            break;
                    }

                    // enable settings controls
                    EnableControls( true );
                }
            }
}
```

НЕЙРОННІ МЕРЕЖІ В НАШ ЧАС

Сьогодні нейронні мережі використовують як альтернативу всім існуючим алгоритмам для машинного перекладу, розпізнавання мови та музики, обробки зображень, визначення об'єктів на фото та відео. Глибоке навчання (Deep learning) – метод машинного навчання, заснований, у першу чергу, на нейронних мережах, хоча можна застосовувати й інші методи. У сучасній реальності практично у всьому, що стосується Deep Learning, використовують нейронні мережі.

Нейронні мережі використовують практично у всіх завданнях, де людина намагається застосувати штучний інтелект. Розглянемо буквально кілька прикладів із посиланнями на більш докладні статті в залежності від типу нейромережі.

Генерація віршів – RNN. Цей вірш було насправді написано RNN. Цікавою особливістю даних мереж є їхнє вміння створювати власні унікальні слова, яких не було в словнику, на якому їх навчали.



Розпізнавання образів – CNN

CNN або Convolutional neural network – одні з найвпливовіших інновацій в області комп'ютерного зору. Застосовуються скрізь, де необхідно розпізнати і/або класифікувати образи/обличчя. Зі складних задач, наприклад, для забезпечення безпеки в аеропортах, на вокзалах, в системах допуску корпорацій із високим рівнем секретності. У щоденному використанні – для більш зручного способу здійснювати покупки.

Фотореалістичні зображення – GAN

GAN або Generative Adversarial Nets використовуються, наприклад, у криміналістиці, коли потрібно створити фоторобот злочинця за описом, в дизайні – для створення предметів одягу або інтер'єру, виходячи з їхнього призначення, в кіновиробництві – коли потрібно поміняти освітлення, настрій кадру, зістарити/омолодити героя.

Прийняття рішень і боти в іграх – DQN

Нейронні мережі типу DQN або Deep Q Learning використовують для прийняття рішень ШІ на підстав аналізу поточної ситуації. Тобто система сама збирає дані, сама їх аналізує, прогнозує найбільш ймовірний результат у тій чи іншій ситуації, приймає максимально вигідне рішення на підставі всіх факторів. Роботу таких нейронних мереж демонструють безпілотні автомобілі, трейдингові боти, чат-боти та ін.

ВИСНОВКИ

З кожним роком людство ставить перед собою все більш складні задачі. Для вирішення цих задач створюються все більш складні та потужні алгоритми, які зможуть це зробити набагато швидше та ефективніше, ніж людина. Але, часом, перед нами постають нелінійні задачі, коли набір вхідних даних кожен раз різний, на вхід приходять дані, які потрібно спочатку фільтрувати, а тільки потім вирішувати поставлену задачу. Для такого типу задач необхідно розробити алгоритм, якій вміє сумніватися, якій розуміє, що не може бути однозначної відповіді і який спроможний знаходити цю відповідь. Наприклад, це можуть бути задачі відстеження фроду при банківських операціях, аномальні показники лічильників у літаку або на атомних станціях. Це можуть бути задачі по розпізнаванню образів з фото або відео даних. Для таких класів задач створюють алгоритмі на основі так званого штучного інтелекту. Ці алгоритмі спроможні працювати кожен раз з різним набором даним, і так само ефективно при цьому вирішувати поставлену задачу. Штучні нейронні мережі, які зазвичай називають просто нейронними мережами, на сьогодні є одним з найбільш відомих та водночас загадкових засобів інтелектуального аналізу даних, що розвивається завдяки досягненням в галузях теорії нейронних мереж та інформатики. Оскільки бурхливий розвиток комп'ютерної техніки створює передумови для появи нейрокомп'ютерів (тобто комп'ютерів 6-го покоління), які, за прогнозами фахівців в галузі штучного інтелекту, будуть переробляти інформацію за тими ж принципами, що й людський мозок, то зацікавленість нейромережевими технологіями поступово охоплює все більш широке коло користувачів.

СПИСОК ЛІТЕРАТУРИ

1. Xabier Basogain Olabe. Redes Neuronales Artificiales y sus Aplicaciones Formato Impreso: Publicaciones de la Escuela de Ingenieros, 1998 – 79 p.

2. Минский М., Пейперт С. Персептроны. М.: Мир, 1971 - 261 с.

3. Терехов В.А., Ефимов Д.В., Тюкин И.Ю. Нейросетевые системы управления: Учеб. Пособие для вузов - М.: Высш. шк. 2002. - 183 с.: ил.

4. Хайкин С. Нейронные сети: полный курс, 2-е изд., испр. : Пер. с англ. - М. : ООО "И.Д. Вильямс", 2006. - 1104 с.

5. Дивеев А.И., Софронова Е.А. ―Основы генетического программирования Учебно-методическое пособие‖ - М.: Изд-во РУДН, 2006;

6. Васенков Д.В. Методы обучения искусственных нейронных сетей //Компьютерные инструменты в образовании. - СПб.: Изд-во ЦПО "Информатизация образования", 2007, №1, С. 20-29.

7. Круг П.Г. Нейронные сети и нейрокомпьютеры: Учебное пособие по курсу «Микропроцессоры». - М.: Издательство МЭИ, 2002. – 176 с.

8. Каллан, Р. Основные концепции нейронных сетей : Пер. с англ. - М. : Издательский дом "Вильямс"', 2001;

9. Круглов В.В., Борисов В.В. Искусственные нейронные сети. Теория и практика. - М.: Горячая линия - Телеком, 2001. - С. 382.

10. Мочалов И.А. Искусственные нейронные сети в задачах управления и обработки информации Ч.1 - М.: 2004. -145 с.

11. Осовский С. Нейронные сети для обработки информации – М.: Финансы и статистика, 2002. – 344 с.

12. Пупков К.А., Егупов Н.Д. «Методы классической и современной теории автоматического управления»: Учебник в 5-и тт.; 2-е изд., перераб. и доп. Т.3: Синтез регуляторов систем автоматического управления / Под ред. 76 К.А. Пупкова и Н.Д. Егупова. - М.: Издательство МГТУ им. Н.Э. Баумана, 2004. - 616 с.; ил.

13. Кочладзе З.Ю., Оганезов А.Л. Об одном возможном подходе к проблеме распознаванияплоских фигур, Университетский журнал. Тбилиси, 2006.

14. Купарадзе М.Р. Структура и функции нервной системы, Москва, Сборник докладов, 1962

15. Чумбуридзе И.Ш., Нуцубидзе М.А., Микашавидзе Г.Н., Береникашвили Н.Н. К вопросу об организации нейронных сетей, способных к обу ению классификации событий. Тезисы докладов V Всесоюзной конференции по нейрокибернетике, РГУ Ростов на-Дону, 1973.

16. Аркадьев А. Г., Браверманн Э. М. Обучение машины классификации объектов, Москва, ―Наука‖, 1971.

Навчальне видання

ТКАЛІЧЕНКО Сергій Володимирович

ШТУЧНІ НЕЙРОННІ МЕРЕЖІ