Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр
(назва освітнього ступеня)

на тему: РОЗРОБКА МЕТОДІВ ДОСЛІДЖЕННЯ НЕЙРОННИХ МЕРЕЖ ВИКОРИСТАННЯМ СЕРЕДОВИЩА WOLFRAM MATHEMATICA ТА МОВИ ПРОГРАМУВАННЯ C++

Виконав(ла): студент(ка) VI курсу, групи СПм
спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

	(підпис)	Лавренів А. Д. (прізвище та ініціали)
Керівник	(підпис)	Бойко І. В. (прізвище та ініціали)
Нормоконтроль	(підпис)	Стоянов Ю. М. (прізвище та ініціали)
Завідувач кафедри	(підпис)	Петрик М. Р. (прізвище та ініціали)
Рецензент	(підпис)	Михайлишин М. С. (прізвище та ініціали)

Тернопіль
2021

АНОТАЦІЯ

Лавренів А.Д. Розробка методів дослідження нейронних мереж з використанням середовища Wolfram Mathematica та мови програмування с++**. – Рукопис.**

Кваліфікаційна робота здобуття освітнього ступеня магістр за спеціальністю 121 – Інженерія програмного забезпечення. – Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра програмної інженерії, група СПм-61 // м.Тернопіль, 2021 // С. , рис. – 73, табл. – 2, додат. – , бібліогр. – 25.

Мета, яка поставлена в магістерській роботі досягається шляхом виконання таких завдань для середовища Wolfram Mathematica: дослідження основних видів нейронних мереж та способів їх подання; навчання нейронної мережі; формування репозиторію нейронної мережі Wolfram; способи отримання необхідної інформації у нейронній мережі;розробка самостійної LeNet архітектури та MXNet фреймфорку для нейронних мереж.

Практичним застосуванням є нейроні мережі та методи роботи з ними у системі Wolfram Mathematica.

Технічні вимоги: розроблені методи роботи з нейронними мережами застосовні до окремих ПК так і для хмарних ресурсів й розподілених систем.

Ключові слова: НЕЙРОННА МЕРЕЖА, ХМАРНІ ТЕХЛОГОЇ, ГРАФИ, ПЕРСЕПТРОН, МАШИННЕ НАВЧАННЯ.

ABSTRACT

Lavreniv A.D. Development of methods for research of neural networks using Wolfram Mathematica and C ++ programming language. – Manuscript.

Qualifying work for obtaining a master's degree in specialty 121 — Software Engineering. – Ternopil Ivan Pul'ui National Technical University, Faculty of Computer Information Systems and Software Engineering, Software Engineering Department, group SPm-61 // Ternopil, 2021 // Pages. – , pictures. – 73, tables. – 2 , supp. – , bibl.ref. – 25.

The main goal set in the master's work is achieved by performing the following tasks for the Wolfram Mathematica environment: researching the main types of neural networks and ways of representing them; neural network training; formation of the Wolfram neural network repository; ways to obtain the necessary information in a neural network; development of an independent LeNet architecture and MXNet framework for neural networks.

Practical application - neural networks and methods of working with them in the Wolfram Mathematica system.

Technical requirements: the developed methods of working with neural networks are applicable to individual PCs and for cloud resources and distributed systems.

Keywords: NEURAL NETWORK, CLOUD TECHNOLOGIES, GRAPHS, PERSEPTRON, MACHINE LEARNING.

ЗМІСТ

ВСТУП

Чому наука про дані (data science) важлива в наші дні? Наука про дані - активна тема, яка розвивається кожен день; нові методи, нові техніки і нові дані створюються кожен день. Наука про дані - це міждисциплінарна область, яка включає в себе наукові методи, алгоритми і систематичні процедури для отримання наборів даних і, таким чином, кращого розуміння даних в їх різних дискретних структурах. Це продовження деяких теоретичних областей аналізу даних, таких як статистика, інтелектуальний аналіз даних, машинне навчання і аналіз шаблонів[1]. З унікальною метою наука про дані витягує цінну кількісну і якісну інформацію з даних, які збираються з різних джерел, що дозволяє об'єктивно підраховувати події для прийняття рішень, розробки продукту, виявлення закономірностей або виявлення нових сфер бізнесу.

Наука про дані виконує ряд процесів для вирішення проблеми, включаючи збір даних, обробку даних, побудова моделі, передачу результатів, а також моніторинг даних або поліпшення моделі. Перший крок – формалізація процесу дослідження. Від об'єкта дослідження ми можемо перейти до джерел отримання наших даних. Цей крок безпосередньо є спрямований на пошук правильних джерел даних. Продуктом цього шляху зазвичай є необроблені дані, які необхідно обробити перед кінцевою роботою з ними.

Обробка даних включає перетворення даних з необробленої форми в стан, в якому вони можуть бути відтворені для побудови математичної моделі. Побудова моделі - це етап, який призначений для отримання інформації шляхом прогнозування відповідно до умов, які були створені на ранніх етапах. Тут використовуються відповідні методи і інструменти, що відносяться до різних дисциплін. Мета полягає в тому, щоб отримати модель, що забезпечує найкращі

результати. Наступним кроком є подання результату дослідження, який складається з повідомлення отриманих результатів та їх відповідності встановленої мети дослідження[2]. В загальному випадку, моніторинг даних має на меті підтримувати дані в актуальному стані, тому що дані можуть змінюватися постійно і по-різному.

В сучасній науці про дані та нейронні мережі, розробка програмного забезпечення ведеться в основному із застосуванням мов Python та R та спеціалізованих середовищ розробки, що орієнтовані під ці мови. Це зумовлено насамперед доступністю цих програмних засобів та можливістю їх застосування в режимі відкритого доступу. Проте разом з простотою написання програмного коду часто доводиться стикатися з рядом проблем зумовлених необхідністю застосування методів об'єктно-орієнтованого програмування та обмеженістю доступної оперативної пам’яті для виконання обчислень і роботи з даними. Як вирішення цих проблем пропонується використовувати додаткові бібліотеки, що в свою чергу ще більше ускладнюють роботу на персональних ПК. Як альтернатива даному програмному забезпеченню, що тільки починає розвиватися є застосування пакетів символьної математики, таких як Wolfram Mathematica, Matlab, Maple. Як перевага над Python та R ці програмні середовища повністю реалізують об’єктно-орієнтований метод програмування та мають усі елементи для виконання розподілених та паралельних обчислень, дозволяють використовувати методи паралельного програмування. Хмарні версії цих програмних систем є повністю доступними для користувачів з вільно розповсюджуваним кодом. У випадку системи Wolfram Mathematica це код, що відповідає стандартам та структурі мови C++[3]. Дані факти дозволяють зробити висновок у необхідності розробки методів роботи з масивами інформації, машинним навчанням та нейронними мережами, які могли б бути реалізованими у середовищі системи Wolfram Mathematica.

Таким чином, безпосередньою метою магістерської роботи є розробка і дослідження методів побудови, навчання та роботи з нейронними мережами з використаннями системи Wolfram Mathematica.

Мета, яка поставлена в даній магістерській роботі досягається шляхом виконання таких завдань для середовища Wolfram Mathematica:

- дослідження основних видів нейронних мереж та способів їх подання;
- навчання нейронної мережі;
- формування репозиторію нейронної мережі Wolfram;
- способи отримання необхідної інформації у нейронній мережі;
- розробка самостійної LeNet архітектури та MXNet фреймфорку для нейронних мереж.

Об’єктом дослідження в магістерській роботі є нейроні мережі та методи роботи з ними у системі Wolfram Mathematica.

Предметом дослідження є нейронні мережі в перспективі їх використання для розв’язання практичних задач з використанням програмного середовища Wolfram Mathematica та типової семантики програмування мови програмування C++.

Безпосередня ідея магістерської роботи полягає у розробці програмних засобів для роботи з нейронними мережами у системі Wolfram Mathematica. Також в роботі розв’язується задача із порівняння ефективності роботи основних типів нейронних мереж та встановлюються особливості подання і отримання даних в них.

Наукова новизна отриманих в дипломній роботі результатів полягає у тому, що вперше було послідовно розроблено програмні засоби для побудови, аналізу та застосування нейронних мереж з використанням середовища та мови програмування Wolfram Mathematica. Встановлено, що нейронні мережі, які працюють у рамках системи Mathematica мають перспективу застосування до широкого кола прикладних та наукових задач[25].

Для розв’язання завдань поставлених у магістерській роботі, застосовувались сучасні методи і моделі нейронних мереж та засоби об’єктно-орієнтованого й спеціалізованого програмного забезпечення віднесеного до предметної області розподілених у хмарних обчислень у інженерії програмного забезпечення.

Найбільш прискіплива увага в роботі присвячена дослідженню сучасних типів нейронних мереж та роботі з ними в порівнянні одна з одною при їх застосуванні до вирішення типових задач і завдань змішаного характеру.

Результати, які було отримано в результаті виконаних завдань та досліджень, що викладені у магістерській роботі доповідались та обговорювались на Міжнародній науково-практичній конференції молодих учених та студентів "АКТУАЛЬНІ ЗАДАЧІ СУЧАСНИХ ТЕХНОЛОГІЙ".

1 АНАЛІТИЧНА ЧАСТИНА

1.1 Нейронні мережі їх характеристика, структура та основні принципи роботи з ними

Насамперед необхідно почати з формулювання основних засад нейронної мережі на мові Wolfram Language. Глава починається з концепцій шарів, того, як використовувати команди для різних шарів, а також з найбільш поширених шарів. Сформулюємо принципи того, як вводити дані в шари через мережевий порт, а також різні форми еквівалентного вираження шарів. Також слід встановити, як розрізняти різні шари за їх символом. Буде показано, що рівні можуть мати кілька опцій, які дозволяють шару мати різні специфікації, особливо з точки зору перегляду поняття і концепції шару нейронної мережі в схемі мови Wolfram Language, а також в результаті порівняння різних шарів, які мають різні призначення і які виконують різні обчислення[4]. Така можливість досягається шляхом дослідження різних функції активації, підтримувані мовою Wolfram Language та вивчення графіків кожної з функцій разом з різними формами синтаксису. Делі перенесемо в нашу систему основні концепції кодерів і декодерів і то, як ці інструменти використовуються для побудови моделі нейронної мережі, в залежності від розв'язуваної задачі. Далі встановимо, як ці кодери та декодери використовуються для перетворення різних типів даних в числові масиви, а також як перетворювати числові масиви назад у вихідні дані. Перенесемо у систему Wolfram Mathematica концепцію контейнера для існуючих його типів і значенням для створюваних моделей. Розвинемо методи того як обробляти і створювати контейнери з різними командами і як графічно візуалізувати створену модель.

Безпосередньо дослідимо як Wolfram Neural Net Framework підтримує операції, пов'язані з MXNet, та операцію експортування мережі в формат операції MXNet.

1.1.1 Шари.

Щоб побудувати нейронну мережу на мові Wolfram Language, необхідно розуміти, що вона побудована з шарів. Шар - це термін, який можна застосовувати до набору вузлів, які працюють разом на певному рівні в нейронної мережі. Шар є важливим і простим елементом, який існує для побудови нейронної мережі.

1.1.2 Вхідні дані.

Дані, що обробляються шарами, відносяться до числового типу, а не до іншого типу. Вхідні змінні можуть бути: вектором, одновимірним списком; матрицями, двовимірними списками і масиви, списками списків або будь-який інший числовим тензором. Ці вхідні змінні можуть бути об'єктами або атрибутами досліджуваного набору даних з відомою формою або багатовимірної формою. Ці типи вхідних атрибутів пов'язані з вхідним шаром, для якого розмір об'єкта, в свою чергу, має дорівнювати вхідному розміром шару, але не кожен шар отримує однакові вхідні дані і повертає однакові вихідні дані; кожен вхід різниться в залежності від типу використовуваного шару. Це визначення є однією із самих

основних ідей нейронних мереж, оскільки вони є важливим компонентом всієї структури, що визначає сам термін як нейронні мережі.

1.1.3 Лінійний шар.

Лінійний шар є найбільш поширеним і широко використовуваним шаром в нейронної мережі. Для створення найпростішого шару в мові Wolfram Language будемо використовувати команду LinearLayer.

```
In[1]:= LinearLayer["Input"→ 1,"Output"→ 2]
Out[1]=
```

Рис. 1.1. Об’єкт, що відповідає лінійному шару LinearLayer

На Рис. 1.1 представлений об'єкт LinearLayer створений за допомогою середовища мови Wolfram Language. При натисканні на піктограму плюса відображаються внутрішні параметри, в тому числі відомості про вхідних і вихідних даних порту шару, а також ранг масиву ваг і зміщень лінійного шару. Це показано на Рис. 1.2.

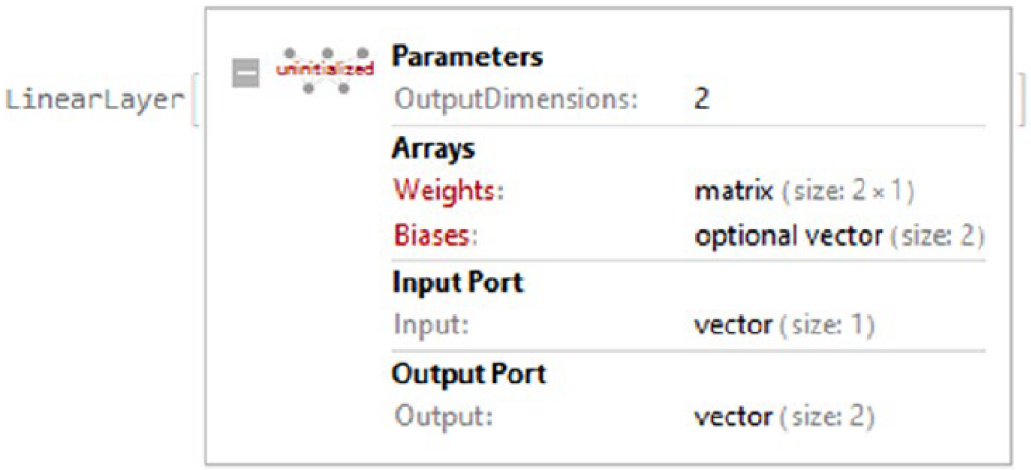


Рис. 1.2. Розширений об'єкт LinearLayer

Отже, кожен створений нами шар має порт введення і порт виводу. Кожен порт має пов'язаний з ним розмір того, що входить в шар, а що виходить. В останньому випадку входить вектор розміру один, а шар повертає вектор розміру два.

Загальний вигляд лінійного шару задається наступним виразом скалярного добутку $\boldsymbol{w}\cdot\boldsymbol{x} + \boldsymbol{b}$, де $\boldsymbol{x}$ - вектор даних, $\boldsymbol{w}$ - матриця ваг, а $\boldsymbol{b}$ - вектор зміщення[5]. Лінійні шари мають інші пов'язані імена, наприклад, повністю зв'язаний шар, як в структурі MXnet. На вхід шарів в Wolfram Language на вхід надходять числові тензори, тобто вони діють тільки на числові масиви.

Щоб явно вказати розмір введення і виведення, ми пишемо форму порту введення і порту виводу, за якими слідують різні параметри: «Вхід» або «Висновок» → {розмір, Параметри.}. Варіанти включають визначення дійсного числа (Real), вектора форми n (одиночне число n), масиву ({n1 * n2 * n3} ...) або NetEncoder, що ми побачимо пізніше. Нижче наведені деякі еквівалентні способи запису шарів, як показано на Рис. 1.3.

```
In[2]:= LinearLayer[“Input"->{2,"Real"},"Output"->{2,1}]
Out[2]=
```

Рис. 1.3. LinearLayer з різними масивами рангів введення і виведення

Як ми бачимо на Рис. 3, шар отримує вектор розміру два (список довжиною 2), який складається з дійсних чисел, а на виході виходить матриця форми 3×2. Коли вказано дійсне число в рамках структури нейронної мережі Wolfram вона працює з точністю Real32. Якщо до шару не додано аргументи, буде виведена форма введення і виведення.

Щоб визначити ваги і зміщення вручну, запишемо їх в формі «Ваги» → число, «Зміщення» → число; крім того немає значень ваг або зсувів. Це показано в наступному прикладі, де ваги і зміщення встановлені на фіксовані значення 1 і 2 (Рис. 1.4).

```
In[3]:= LinearLayer["Input"→ 1,"Output"→ 1,"Weights"→ 1,"Biases"→ 2]
Out[3]=
```

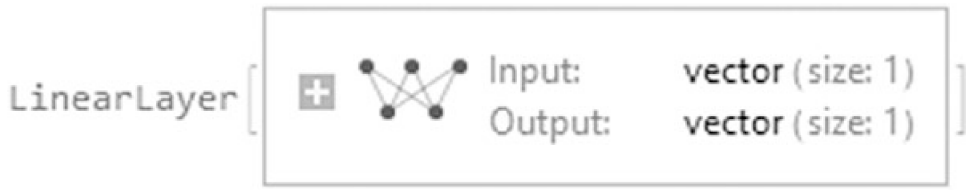


Рис. 1.4. Ініціалізований лінійний шар з фіксованими зміщеннями і вагами

1.1.4 Ініціалізація шару.

Крім можливості форматувати шар самостійно, є ще одна команда, яка дозволяє нам форматувати шар випадковими значеннями; це NetInitialize. Отже, щоб встановити утримувані значення ваг або зсувів, ми також можемо використовувати параметр LearningRateMultipliers (Рис. 5). Крім цього LearningRateMultipliers також відзначає швидкість, з якою рівень навчається на етапі навчання.

```
In[4]:= NetInitialize[LinearLayer["Input"→ "Real","Output"→ "Real",
LearningRateMultipliers→{"Biases"→1}]]
Out[4]=
```

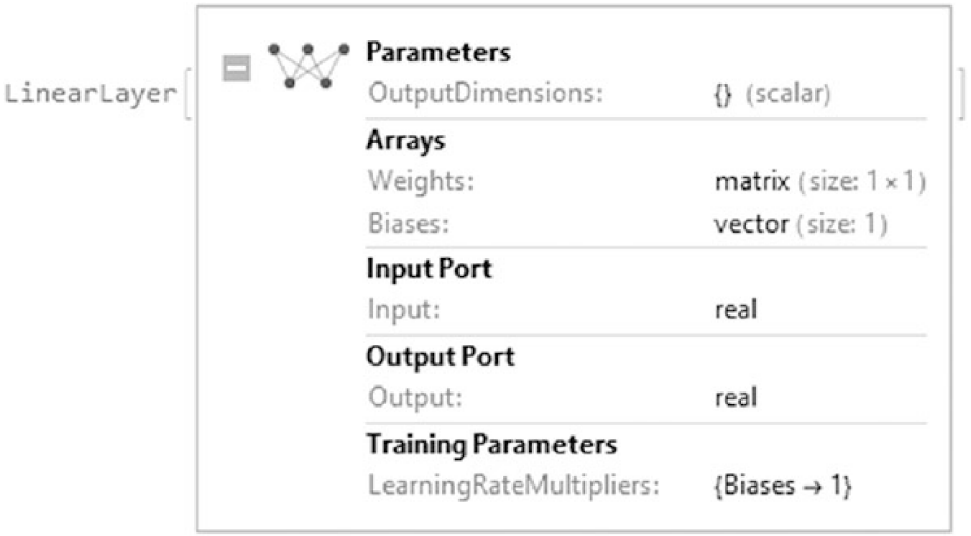


Рис. 1.5. Об’єкт LinearLayer з параметрами навчання

Коли шар ініціалузується, неініціалізований текст зникає. Якщо ми спостерігаємо за властивостями нового шару, в параметрах навчання виявиться, що були встановлені фіксовані зміщення і швидкість навчання.

Для NetInitialize доступні наступні параметри: Method і RandomSeeding. Із таких методів: Kaiming, Xavier, Orthogonal (ортогональні ваги) і Random (вибір ваг з розподілу). Наприклад, ми можемо використовувати вибірку ініціалізації Xavier з нормального розподілу, як показано на Рис. 6.

```
In[5]:= NetInitialize[LinearLayer["Input"→ "Real","Output"→ "Real",
LearningRateMultipliers→{"Biases"→1}], Method→ {"Xavier","Distribution"→"Normal"},
RandomSeeding→888]
Out[5]=
```

Рис. 1.6. LinearLayer ініціалізований методом Xavier

Незважаючи на можливість встановлювати ваги і зміщення вручну, будемо починати з шару з випадковими значеннями, щоб підтримувати певний рівень складності в загальній структурі моделі, оскільки, навпаки, це може вплинути на створення моделі нейронної мережі, яка не дає точних прогнозів нелінійного поводження.

1.1.5 Отримання даних.

NetExtract використовується для отримання значень ваг і зміщень в формі NetExtract [net, {level1, level2, ...}. Параметри ваг і зміщень лінійних шарів упаковані в об'єкти NumericArray (Рис. 1.7.). Цей об'єкт буде мати значення,

розміри і тип значень в шарі. NetExtract також дозволяє отримувати кількість шарів багатошарової мережі. NumericArrays використовуються в мові Wolfram Language для скорочення споживання пам'яті та часу обчислень.

```
In[6]:= LinearL=NetInitialize[LinearLayer[2, "Input"→ 1],RandomSeeding→888];
NetExtract[LinearL,#]&/@{"Weights","Biases"}//TableForm
Out[6]=
```

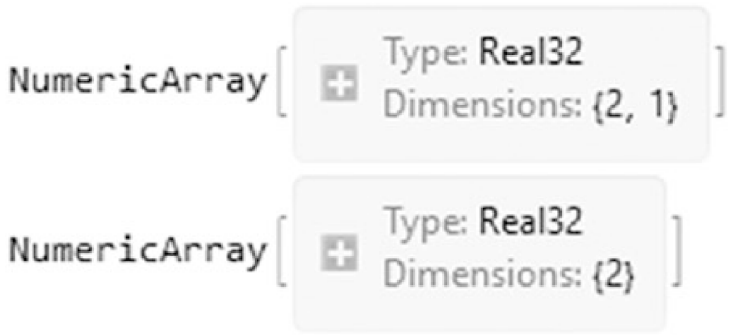


Рис. 1.7. Ваги і зміщення лінійного шару. За допомогою Normal ми конвертуємо їх в списки.

```
In[7]:= TableForm[SetPrecision[{{Normal[NetExtract[LinearL,"Weights"]]
},{Normal[NetExtract[LinearL,"Biases"]]}},3],TableHeadings→{{"Weights→",
"Biases →"},None}]
Out[7]//TableForm=
```

$$\text{Weights} \to \begin{matrix} -0.779 \\ 0.0435 \end{matrix}$$

$$\text{Biases} \to \begin{matrix} 0. \\ 0. \end{matrix}$$

Наприклад, шар може отримати вектор довжини 1 для створення вихідного вектора розміру 2.

```
In[8]:= LinearL[4]
Out[8]= {-3.11505,0.174007}
```

Якщо введення не подано у відповідній формі, шар не буде виконуватися.

```
In[9]:= LinearL[{88,99}]
```

```
Out[9]= LinearLayer::invindata1: Data supplied to port "Input" was a
length-2 vector of real numbers, but expected a length-1 vector of real numbers.
        $Failed
```

Ваги і зміщення - це параметри, з яких модель повинна навчатися, які можуть бути адаптовані на основі вхідних даних, які отримує модель, тому вона ініціалізується випадковим чином, оскільки, якщо ми спробуємо отримати ці значення без ініціалізації, ми не будемо мати можливість, оскільки вони не визначені.

Шари мають властивість диференціювання. Це досягається за допомогою опції NetPortGradient, який може представляти градієнт виведення мережі щодо порту або параметра. Наприклад, вкажемо похідну виходу по відношенню до входу для певного вхідного значення.

```
In[10]:= LinearL[2,NetPortGradient["Input"]]
Out[10]= {-0.735261}
```

1.1.6 Середньоквадратичний шар.

Досі ми розглядали лінійний шар, який має різні властивості. Шари із позначкою з'єднаного ромба (Рис. 1.8), навпаки, не містять ніяких навчаваних параметрів, таких як MeanSquaredLossLayer, AppendLayer, SumptingLayer, DotLayer, ContrastiveLossLayer і SoftmaxLayer.

```
In[11]:= MeanSquaredLossLayer[]
Out[11]:=
```

Рис. 1.8. MeanSquaredLossLayer

MeanSquaredLossLayer [], має більше одного входу; це тому, що цей шар обчислює середньоквадратичне відхилення, яке задається наступним виразом $\frac{1}{n}\sum(\text{Input}-\text{Target})^2$ і має властивість, що порівнює два числових масиви. При використанні MeanSquaredLossLayer розміри вхідних/вихідних портів вводяться в тій же формі, що і лінійний шар, а значення вхідних і цільових значень вводяться як Associations

```
In[1]:= MeanSquaredLossLayer["Input"→{5,6},"Target"→{1,5}][Association["
Input"→{{2,3},{2,3},{4,3}},"Target"→{{3,3},{3,3},{5,4}}]]
Out[2]= 2.7767
```

В останньому прикладі обчислюється MeanSquaredLossLayer для вхідних/вихідних вимірів трьох рядків і двох стовпців або шляхом визначення спочатку шару, а потім застосування шару до даних.

Параметри вхідного порту і цільового порту аналогічні установкам лінійного шару з різними формами: Input → «Real», n (форма вектора n), { $n1\times n2\times n3$ } ... (n – вимірний масив), «Varying» (вектор або змінювальна форма) або NetEncoder, але за винятком того, що вхід і ціль повинні мати однакові розміри. Кілька форм шарів показані на Рис. 1.9.

```
In[16]:= {MeanSquaredLossLayer["Input"→"Varying","Target"→"Varying"],Mean
SquaredLossLayer["Input"→ NetEncoder["Image"],"Target"→ NetEncoder["Image
"]],MeanSquaredLossLayer["Input"→1,"Target"→1]}//Dataset
Out[16]=
```

Рис. 1.9. Шари втрат з різними вхідними і цільовими формами

1.2 Аналіз предметної області

Функції активації - важлива частина побудови нейронної мережі. Роль функції активації - повертати вихід з встановленого діапазону при заданому вході. У Wolfram Language функції активації розглядаються як шари. Шар, який широко використовується в структурі нейронної мережі Wolfram Language, - це ElementwiseLayer.

За допомогою цього шару ми можемо представляти шари, які можуть застосовувати унарну функцію до елементів вхідних даних - іншими словами, функцію, яка отримує тільки один аргумент. Ці функції також відомі як функції активації. Наприклад, однією з найбільш часто використовуваних функцій є гіперболічний тангенс (Tanh[x]), який показаний на малюнку Рис. 1.10.

```
In[17]:= ElementwiseLayer[Tanh[#]&](* Altnernate form
ElementwiseLayer[Tanh]*)
Out[17]=
```

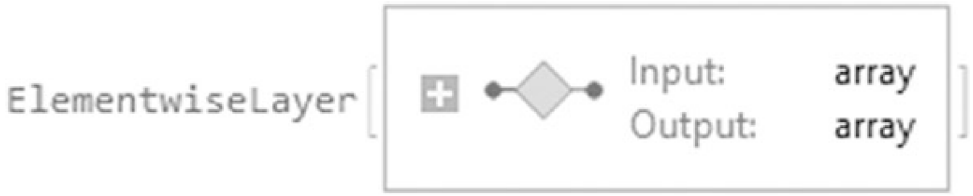


Рис. 1.10. Шар Tanh[x]

Поелементні шари не мають обучуваних параметрів. Використовується чиста функція, тому що шари не можуть отримувати символи. Якщо клацнути значок плюса, відображається детальна інформація про порти, а також параметри відповідної функції, якої в даному випадку є Tanh. Визначивши ElementWiseLayer, він може отримувати значення, як і інші шари.

```
In[18]:= In[52]:= ElementwiseLayer[Tanh[#]&];
Table[%[i],{i,-5,5}]
Out[18]= {-0.999909,-0.999329,-0.995055,-0.964028,-
0.761594,0.,0.761594,0.964028,0.995055,0.999329,0.999909}
```

Якщо форма введення або виведення не задана, шар буде визначати тип даних, які він отримає або поверне. Наприклад, вказавши тільки введення як дійсний, Mathematica зробить висновок, що висновок буде дійсним (Рис. 1.11).

```
In[19]:= TanhLayer=ElementwiseLayer[Tanh,"Input"→ "Real"]
Out[19]=
```

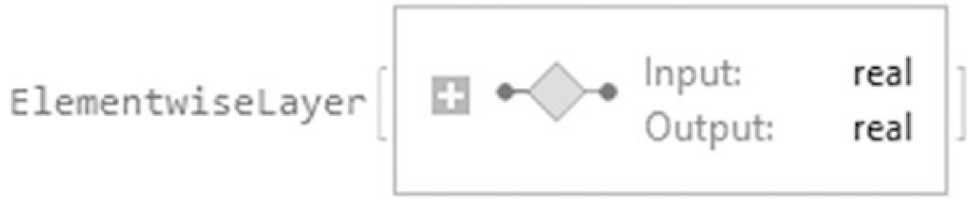


Рис. 1.11. ElementWiseLayer з тим же виходом, що і вхід

Кожен рівень в мові Wolfram Language можна запустити через графічний процесор (ГП) або центральний процесор (ЦП), вказавши параметр TargetDevice.

Наприклад, давайте побудуємо раніше створені шари за допомогою TargetDevice на ЦП (Рис. 1.12)

```
In[21]:= GraphicsRow@{Plot[TanhLayer[x,TargetDevice→
"CPU"],{x,-12,12},PlotLabel→"Hiperbolic
Tangent",AxesLabel→{Style["x",Bold],Style["f(x)",Italic]},PlotStyle→
ColorData[97,25],Frame→True],Plot[RampLayer[x,TargetDevice→ "CPU"],{x,-
12,12},PlotLabel→"ReLU",AxesLabel→{Style["x",Bold],Style["f(x)",Italic]},
PlotStyle→ColorData[97,25],Frame→True]}
Out[21]=
```

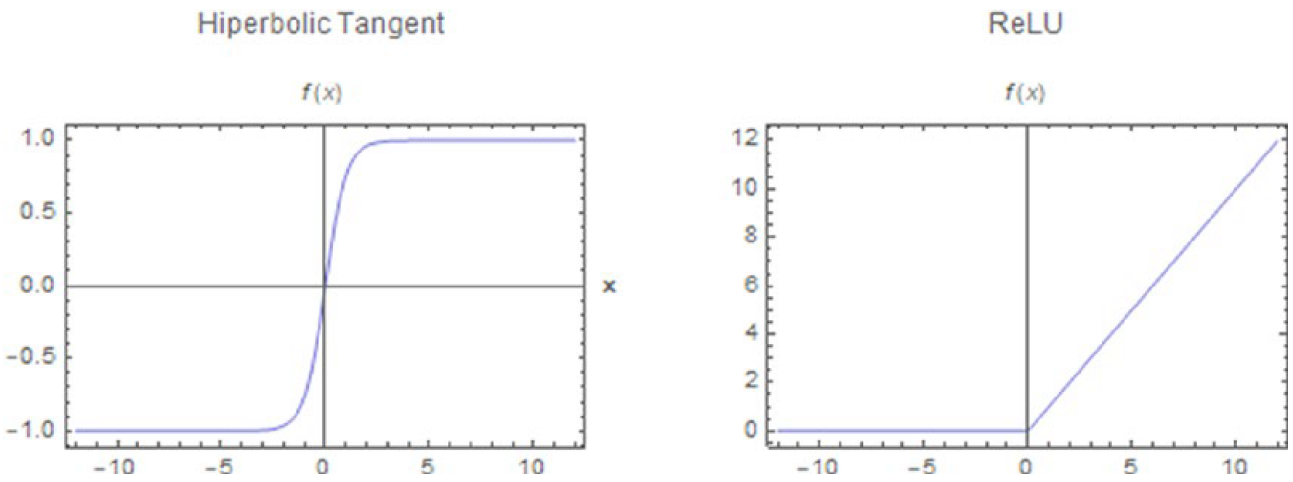


Рис. 1.12. Функції активації Tanh[x] і Ramp[x].

Інші функції можуть використовуватися по їх імені або синтаксису Wolfram Language - наприклад, функція SoftPlus.

```
In[22]:= GraphicsRow@{Plot[ElementwiseLayer["SoftPlus"][x,TargetDevice→
"CPU"],{x,-12,12},PlotLabel→#1,AxesLabel→#2,PlotStyle→#3,Frame→#4],
Plot[ElementwiseLayer[Log[Exp[#]+1]&][x,TargetDevice→ "CPU"],{x,-12,12},
PlotLabel→#1,AxesLabel→#2,PlotStyle→#3,Frame→#4]}&["SoftPlus",
{Style["x",Bold],Style["f(x)",Italic]},ColorData[97,25],True]
Out[22]=
```

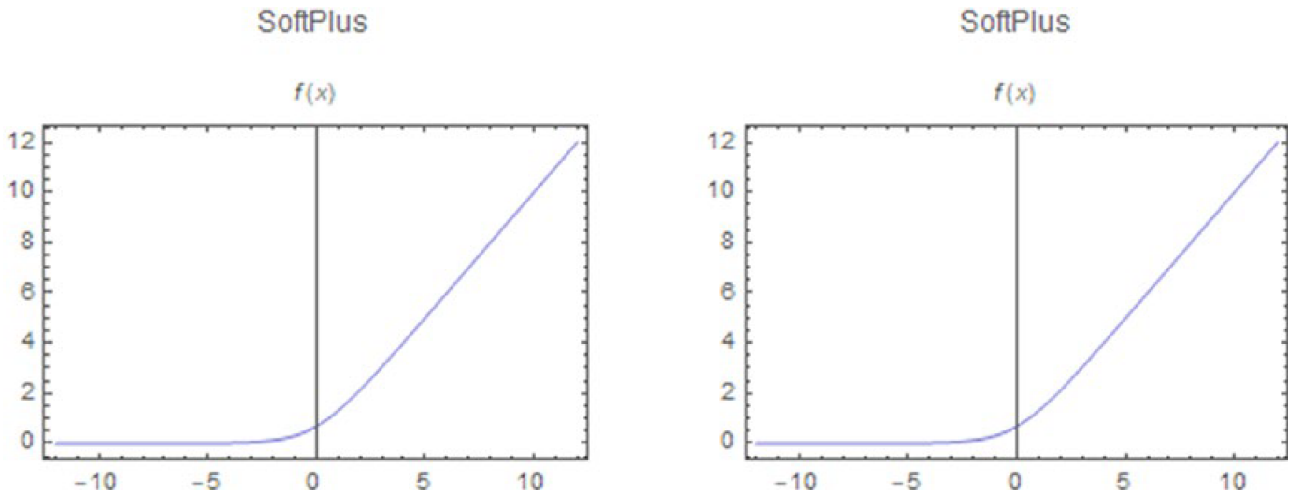


Рис. 1.13. Функція SoftPlus, що генерується асоційованим ім'ям і чистою функцією

Інші загальні функції показані на наступних графіках, такі як масштабована експоненціальна лінійна одиниця, сигмовидна, жорстка сигмоїдальна і жорстка гіперболічна дотична (Рис. 1.14).

```
In[23]:=
GraphicsGrid@{{Plot[ElementwiseLayer["ScaledExponentialLinearUnit"]
[i,TargetDevice→ #1],#2,AxesLabel→#3,PlotStyle→#4,Frame→#5,PlotLabel→
"ScaledExponentialLinearUnit"],
Plot[ElementwiseLayer[LogisticSigmoid][i,TargetDevice→ #1],#2,AxesLabel→
#3,PlotStyle→#4,Frame→#5,PlotLabel→"LogisticSigmoid"]},
{Plot[ElementwiseLayer["HardSigmoid"][i,TargetDevice→ #1],#2,AxesLabel→
#3,PlotStyle→#4,Frame→#5,PlotLabel→"HardSigmoid"],
Plot[ElementwiseLayer["HardTanh"][i,TargetDevice→ #1],#2,AxesLabel→#3,
PlotStyle→#4,Frame→#5,PlotLabel→"HardTanh"]}}&["CPU",{i,-10,10},
{Style["x",Bold],Style["f(x)",Italic]},ColorData[97,25],True]
Out[23]=
```

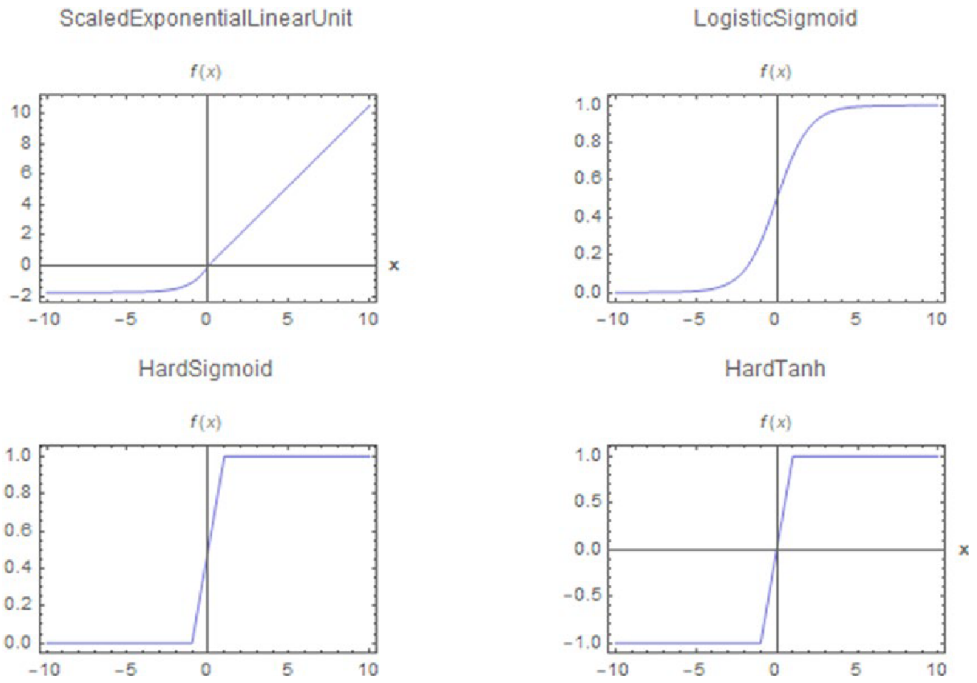


Рис. 1.14. Графіки чотирьох різних функцій активації

1.2.1 SoftmaxLayer.

SoftmaxLayer - це шар, що використовує вираз $S(x_i) = \exp(x_i) / \sum_{i=1}^{n} \exp(x_i)$, де x є вектор, а xi - компоненти вектора. Цей вираз відомий як функція Softmax. Функціональні можливості цього рівня полягають у перетворенні вектора на нормалізований вектор, який складається зі значень в діапазоні від 0 до 1. Цей рівень зазвичай використовується для представлення розбиття класів на основі ймовірностей кожного з них, і це використовується для завдань, пов'язаних із класифікацією.

Форми введення та виведення в SoftmaxLayer можуть бути введені як інші загальні шари, за винятком форми Real.

In[24]:= SFL=SoftmaxLayer["Input"→ 4,"Output"→ 4];

In[25]:= SetAccuracy[SFL[{9,8,7,6}],3]

Out[25]= {0.64,0.24,0.09,0.03}

Сума останнього дорівнює 1. SoftmaxLayer дозволяє вказати глибину нормалізації. Це видно у властивостях параметра шару. Рівень –1 призводить до нормалізації плоского списку. Крім того, SoftmaxLayer може отримувати багатовимірні масиви, а не лише нормалізовані списки.

In[26]:= SoftmaxLayer[1,"Input"→{3,2}];

InitaalPrecision[%[{{6,7},{8,7},{7,8}}],3]//MatrixForm

Out[26]//MatrixForm=

$$\begin{pmatrix} 0.212 & 0.422 \\ 0.576 & 0.155 \\ 0.212 & 0.422 \end{pmatrix}$$

Підсумовування елементів перших стовпців дає те саме для другого стовбця Інший практичний шар називається CrossEntropyLossLayer. Цей шар широко використовується як функція втрат для завдань класифікації. Цей шар втрат вимірює наскільки добре працює модель класифікації. При введенні рядка «Ймовірності» як аргумент шару втрат обчислюється крос-ентропійна втрата шляхом порівняння ймовірності вхідного класу з ймовірністю цільового класу.

In[27]:= CrossEntropyLossLayer["Probabilities","Input"→3];

Тепер цільову форму встановлено на ймовірності класів; вхідні дані та цілі вводяться так само, як і для MeansSquaredLoss.

In[28]:= %[<|"Input"→{0.2,0.5,0.3},"Target"→{0.3,0.5,0.2}|>]

Out[28]= 1.0702

Встановлення двійкового аргументу шарі використовується, коли ймовірності становлять двійкову альтернативу.

In[29]:=CrossEntropyLossLayer["Binary","Input"→ 1];

%[<|"Input"→ 0.1,"Target"→ 0.9|>]

Out[29]= 2.08286

Узагальнені властивості шарів у мові Wolfram Language, вхідні та вихідні дані шарів завжди є скалярами та числовими матрицями. Шари оцінюються з використанням нижчої числової точності, наприклад чисел з одинарною точністю. Шари мають властивість диференціації, це допомагає моделі виконувати ефективне навчання, оскільки деякі методи навчання дозволяють вирішити проблеми випуклої оптимізації[6].

2. ТЕОРЕТИЧНА ЧАСТИНА

2.1 Кодери та декодери

2.1.1 Кодери

Якщо передбачається використання аудіо, зображень або інших типів змінних, цей тип даних необхідно перетворити на числовий масив, щоб їх можна було ввести як вхідні дані в шар. Шари повинні мати NetEncoder, підключений до входу, щоб виконати правильну побудову. NetEncoders інтерпретують зображення, звук тощо. буд. Дані в числове значення, щоб їх можна було використовувати у мережній моделі. З типом кодування пов'язані різні імена. Найбільш поширеними є: Boolean (True або False, кодування як 1 або 0), Characters (рядкові символи як поточне векторне кодування), Class (мітки класів як ціле чисельне кодування), Function (кодування функцій користувача), Image (2D-зображення кодування як масив рангу 3) та Image3D (кодування тривимірного зображення як масив рангу 4).

Аргументами кодера є ім'я або ім'я та відповідні характеристики кодера (Рис. 2.1).

```
In[30]:= NetEncoder["Boolean"]
Out[30]=
```

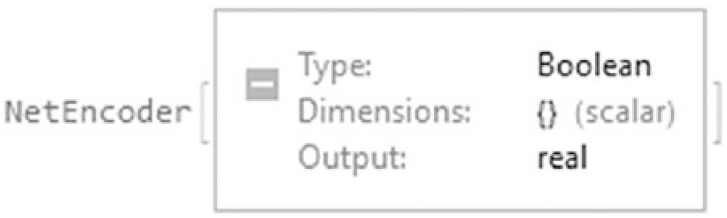


Рис. 2.1. Логічний тип NetEncoder

Для тестування кодувальника ми використовуємо таке.

```
In[31]:= Print["Booleans:",{%[True],%[False]}]
Booleans:{1,0}
```

NetEncoder може мати класи з різними індексними мітками. Подібно до класифікації класу X і класу Y, це буде відповідати індексу в діапазоні від 1 до 2 (Рис. 2.2).

```
In[32]:= NetEncoder[{"Class",{"Class X","Class Y"}}]
Out[32]=
```

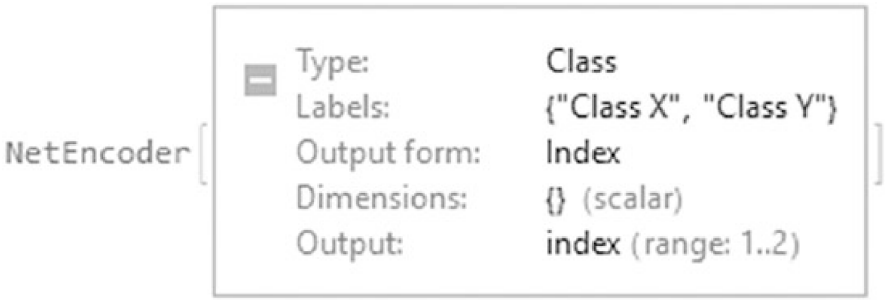


Рис. 2.2. Тип класу NetEncoder

```
In[33]:= Print["Classes:",%[Table[RandomChoice[{"Class X","Class
Y"}],{i,10}]]]
Classes:{2,1,2,2,1,1,2,1,1,1}
```

Наступний код використовується для одиничного вектора.

```
In[34]:= NetEncoder[{"Class",{"Class X","Class Y","Class Z"},"UnitVector"}];
Print["Unit Vector:",%[Table[RandomChoice[{"Class X","Class Y","Class
Z"}],{i,5}]]]
Print["MatrixForm:",%%[Table[RandomChoice[{"Class X","Class Y","Class
Z"}],{i,5}]]//MatrixForm[#]&]
Unit Vector:{{0,1,0},{0,0,1},{0,1,0},{1,0,0},{1,0,0}}
```

MatrixForm:
$$\begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

Залежно від імені, що використовується всередині NetEncoder, властивості, пов'язані з кодувальником, можуть відрізнятися. Це відображається в різних об'єктах кодувальника. Щоб приєднати NetEncoder до шару, кодери вводяться в порт введення - наприклад, для ElementwiseLayer (Рис. 2.3.). І тут вхідний порт шару має ім'я Boolean; Рівень розпізнає, що це NetEncoder логічного типу. Клік по імені Boolean покаже відповідні властивості.

In[35]:= ElementwiseLayer[Sin,"Input"→NetEncoder["Boolean"]]

Out[35]//Short=

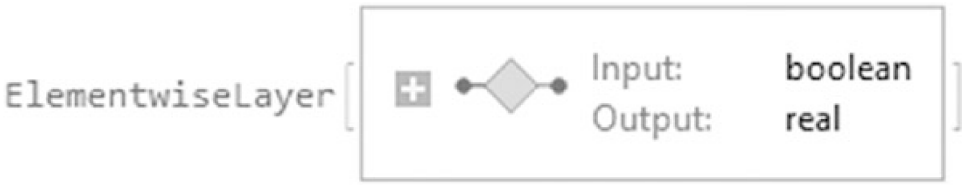


Рис. 2.3. Шар із кодувальником, прикріпленим до вхідного порту

Для LinearLayer використовуємо:

In[36]:= LinearLayer["Input"→NetEncoder[{"Class",{"Class X","Class Y"}}],
"Output"→ "Scalar"]

Out[36]=

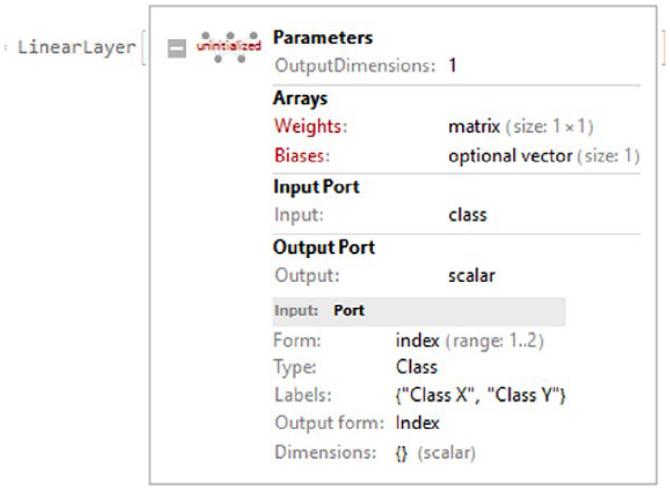


Рис. 2.4. Кодувальник класів, прикріплений до лінійного шару

При натисканні по вхідному порту відобразяться характеристики кодувальника, як показано на Рис. 2.4. NetEncoder також використовується для перетворення зображень у числові матриці або масиви шляхом вказівки класу, розміру або ширини та висоти вихідних розмірів, а також колірного простору, який може бути відтінками сірого, RGB, CMYK або HSB (відтінок, насиченість та яскравість) - наприклад, кодування зображення, яке створює масив 1 x 28 x 28 у відтінках сірого або масив 3 x 28 x 28 у шкалі RGB (рис. 9-21), незалежно від розміру вхідного зображення. Перший ранг масиву представляє колірний канал, а два інших - просторові виміри.

In[37]:= Table[NetEncoder[{"Image",{28,28},"ColorSpace"→ Color}],{Color,
{"Grayscale","RGB"}}]

Out[37]=

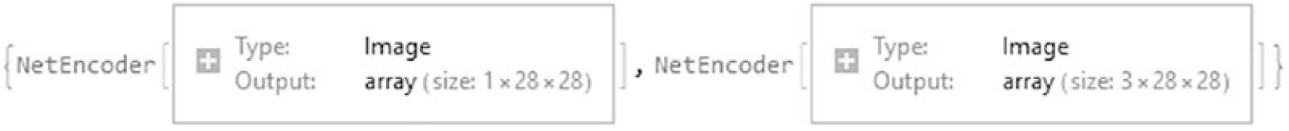


Рис. 2.5. NetEncoders для зображень у градаціях сірого та в масштабі RGB

Після того, як кодувальник встановлений, його можна застосувати до бажаного зображення, потім кодувальник створює числову матрицю із зазначеним розміром. Створення NetEncoder для зображення покаже, серед іншого, відповідні властивості, такі як тип, розмір вхідного зображення та колірний простір. Застосування кодувальника створить матрицю встановленого розміру.

```
In[38]:= ImgEncoder=NetEncoder[{"Image",{3,3},"ColorSpace"→ "CMYK"}];
Print["Numeric Matrix:",%[ExampleData[{"TestImage","House"}]]//MatrixForm]
```

$$\text{Numeric Matrix:}\begin{pmatrix} \begin{pmatrix}0.255\\0.168\\0.255\end{pmatrix} & \begin{pmatrix}0.145\\0.00392\\0.0235\end{pmatrix} & \begin{pmatrix}0.0784\\0.0118\\0.0274\end{pmatrix} \\ \begin{pmatrix}0.153\\0.196\\0.129\end{pmatrix} & \begin{pmatrix}0.2\\0.31\\0.255\end{pmatrix} & \begin{pmatrix}0.259\\0.349\\0.306\end{pmatrix} \\ \begin{pmatrix}0.047\\0.102\\0.00784\end{pmatrix} & \begin{pmatrix}0.164\\0.321\\0.164\end{pmatrix} & \begin{pmatrix}0.262\\0.384\\0.176\end{pmatrix} \\ \begin{pmatrix}0.16\\0.262\\0.184\end{pmatrix} & \begin{pmatrix}0.255\\0.408\\0.478\end{pmatrix} & \begin{pmatrix}0.325\\0.388\\0.569\end{pmatrix} \end{pmatrix}$$

Згенерований висновок є числову матрицю, яка тепер готова до реалізації в мережевій моделі. Якщо форма вхідного зображення знаходиться в іншому колірному просторі, кодувальник змінить форму і перетворить зображення на встановлений колірний простір. Використання зображення у цьому прикладі виходить із ExampleData [{«TestImage», «House»}].

2.1.2 Рівень об'єднання

Кодери можуть бути додані до портів окремих рівнів або контейнерів, вказавши кодувальник для порту, наприклад PoolingLayer. Ці шари використовуються в основному в згорткових нейронних мережах (Рис. 2.6).

```
In[39]:= PoolLayer=PoolingLayer[{3,3},{2,2},PaddingSize→0,"Function"→
Max,"Input"→ NetEncoder[{"Image",{3,3},"ColorSpace"→ "CMYK"}](*Or
ImgEncoder*)]
Out[39]=
```

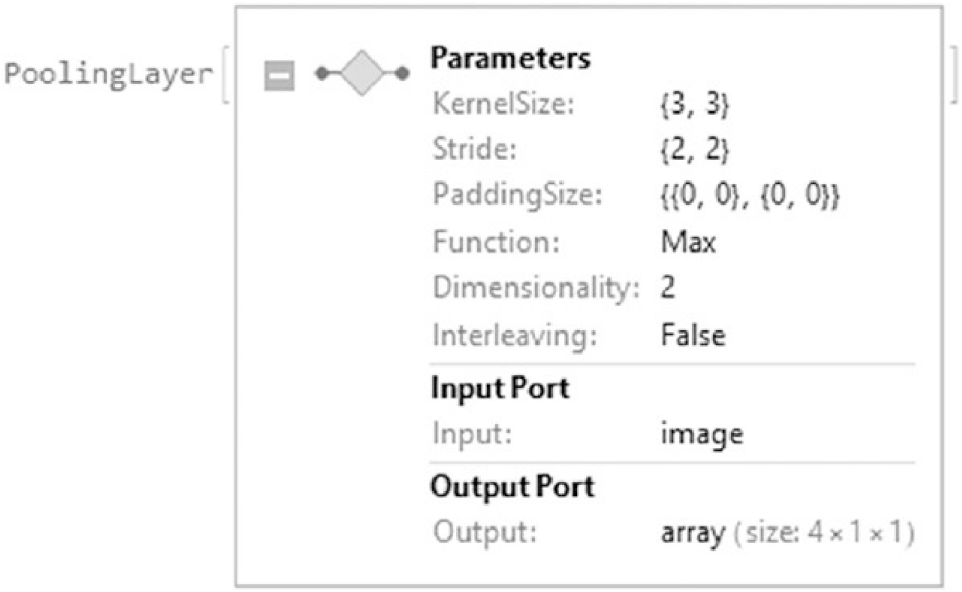


Рис. 2.6. PoolingLayer з NetEncoder

Останній рівень має специфікацію для двовимірного PoolingLayer з розміром ядра 3 x 3 і кроком 2 x 2, який є розміром кроку між додатками ядра. PaddingSize додає елементи на початок і кінець матриці. Це використовується для того, щоб поділ між матрицею та розміром ядра було цілим числом, щоб уникнути втрати інформації між шарами. Функція вказує функцію операції об'єднання, що

дорівнює Max; при цьому він обчислює максимальне значення в кожному фрагменті кожного фільтра, середнє для середнього значення фільтра і суму для підсумовування значень фільтра. Іноді вони можуть бути відомі як максимальні та середні рівні об'єднання.

In[40]:= PoolLayer[ExampleData[{"TestImage","House"}]]//MatrixForm

Out[40]//MatrixForm=

$$\begin{pmatrix} (0.255) \\ (0.349) \\ (0.384) \\ (0.569) \end{pmatrix}$$

2.1.3 Декодери

Як тільки операції мережі будуть завершені, вона поверне числові вирази. З іншого боку, у деяких завданнях нам не потрібні числові вирази, наприклад, у задачах класифікації[7,8], де класи можуть бути задані як вихідні дані, де модель може сказати, що певний об'єкт належить класу A, а інший об'єкт належить до класу B, тому векторний або числовий масив може представляти ймовірність кожного класу. Для перетворення числових масивів на інші форми даних використовується NetDecoder (Рис. 2.7).

In[41]:= Decoder=NetDecoder[{"Class",CharacterRange["W","Z"]}]

Out[41]=

Рис. 2.7. NetDecoder для чотирьох різних класів

Розмір декодера дорівнює конструкції класу. Ми можемо застосувати вектор ймовірностей і декодер інтерпретує його і повідомить нам клас, до якого він належить. Він також відобразить ймовірність класів.

```
In[42]:= Decoder@{0.3,0.2,0.1,0.4}(*This is the same as Decoder[{0.3,0.2,0.
1,0,4},"Decision"] *)
Out[42]= Z
```

TopDecisions, TopProbabilites та невизначеність розподілу ймовірностей відображаються так.

```
In[43]:= TableForm[{Decoder[{0.3,0.2,0.1,0.4},"TopDecisions"→ 4](* or
{"TopDecisions", 4} the same is for TopProbabilities*),
Decoder[{0.3,0.2,0.1,0.4},"TopProbabilities"→ 4],
Decoder[{0.3,0.2,0.1,0.4},"Entropy"]},TableDirections→Column,TableHeadings
→{{Style["TopDecisions",Italic],Style["TopProbabilities",Italic],
Style["Entropy",Italic]},None}]
Out[45]//TableForm=
```

TopDecisions	Z	W	X	Y
TopProbabilities	Z→0.4	W→0.3	X→0.2	Y→0.1
Entropy	1.27985			

Глибина введення додається, щоб визначити рівень застосування класу з урахуванням списку значень.

```
In[44]:= NetDecoder[{"Class",CharacterRange["X","Z"],"InputDepth"→2}];
```

Застосування декодера до вкладеного списку значень призведе до наступного.

```
In[45]:= TableForm[{%[{{0.1,0.3,0.6},{0.3,0.4,0.3}},"TopDecisions"→ 3]
(* or {"TopDecisions", 4} the same is for TopProbabilities*),
%[{{0.1,0.3,0.6},{0.3,0.4,0.3}},"TopProbabilities"→ 3],
%[{{0.1,0.3,0.6},{0.3,0.4,0.3}},"Entropy"]},TableDirections→Column,Table
Headings→{{Style["TopDecisions",Italic],Style["TopProbabilities",Italic],
Style["Entropy",Italic]},None}]
Out[45]//TableForm=
```

TopDecisions	Z Y X	Y X Z
TopProbabilities	Z→0.6 Y→0.3 X→0.1	Y→0.4 X→0.3 Z→0.3
Entropy	0.897946	1.0889

Декодер додається до вихідного порту шару, контейнера або мережної моделі.

```
In[46]:=SoftmaxLayer["Output"→NetDecoder[{"Class",{"X","Y","Z"}}]];
```

Застосування шару до даних дає дає ймовірності реалізації кожного класу.

```
In[47]:= {%@{1,3,5},%[{1,3,5},"AllProbabilities"],%[{1,3,5},"Decision"]}
Out[47]={Z,<|X→0.0158762,Y→0.11731,Z→0.866813|>,Z}
```

2.2 Застосування кодерів та декодерів

Далі ми будемо реалізовувати весь процес кодування та декодування, показаний на Рис. 2.8. Спочатку зображення буде змінено на 200 пікселів шириною, щоб показати, як виглядає вихідне зображення до кодування.

```
In[48]:= Img=ImageResize[ExampleData[{"TestImage","House"}],200]
Out[48]=
```

Рис. 2.8. Приклад зображення будинку

Потім визначаються кодер та декодер.

```
In[49]:=
Encoder=NetEncoder[{"Image",{100,100},"ColorSpace"→ "RGB"}];
Decoder=NetDecoder[{"Image",ColorSpace→ "Grayscale"}];
```

Потім кодер застосовується до зображення, а декодер застосовується до числової матриці. Розміри декодованого зображення перевіряються на відповідність вихідним розмірам кодера.

In[50]:=

Encoder[Img];

Decoder[%]

Рис. 2.9. Приклад декодованого зображення будинку

Як бачимо на Рис. 2.9, зображення було перетворено на зображення у відтінках сірого з новими розмірами.

In[51]:= ImageDimensions[%]

Out[51]= {100,100}

Як видно, розмір зображення було змінено. Послідовно розберемось з етапами процесу, такі як перегляд числової матриці та об'єктів, що відповідають кодувальнику та декодеру. Використання кодувальників та декодерів залежить від типу даних, які ви використовуєте, тому що кожна мережна модель отримує різні вхідні дані та генерує різні вихідні дані[9].

2.3 Мережеві ланцюжки та графи

2.3.1 Контейнери

Нейронні мережі складаються з різних шарів, а не окремих шарів. Для побудови складніших структур, що з кількох шарів, використовується команда NetChain чи NetGraph. Ці контейнери важливі для правильної роботи та побудови нейронних мереж мовою Wolfram Language. У мові Wolfram Language контейнери – це структури, які збирають інфраструктуру моделі нейронної мережі. Контейнери можуть мати декілька форм. NetChain корисний для створення мереж лінійних та нелінійних структур. Це допомагає моделі вивчати нелінійні закономірності[10]. Ми можемо думати, що кожен рівень, що існує в мережі, матиме рівень абстракції, який служить для виявлення складної поведінки, яку не можна було б розпізнати, якби ми працювали лише з одним-єдиним шаром. В результаті ми можемо будувати мережі звичайним чином, починаючи з трьох шарів: вхідного шару, прихованого шару та вихідного шару.

NetChain може поєднувати дві операції. Їх можна записати як чисту функцію, а не просто ім'я функції (Рис. 2.10).

```
In[52]:= NetChain[{ElementwiseLayer[LogisticSigmoid@#&],ElementwiseLayer
[Sin@#&]}]
Out[52]=
```

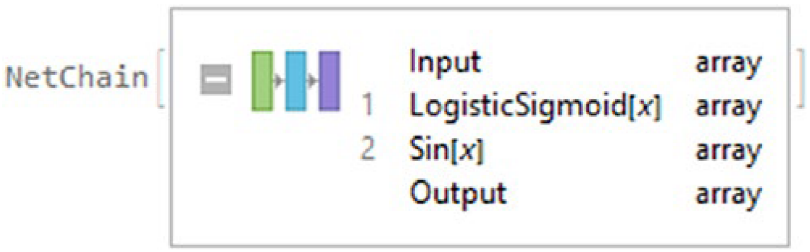


Рис. 2.10. NetChain, що містить два поелементні шари

Повернутий об'єкт є NetChain, і з'являється значок у вигляді трьох кольорових прямокутників. Це означає, що створений (NetChain) або згаданий об'єкт є ланцюжком з'єднань і містить шари. Якщо ланцюжок досліджено, він покаже вхідний, перший (LogisticSigmoid), другий (Sin) та вихідний шари. Операції виконуються у порядку появи: спочатку наноситься перший шар, потім другий. У NetChain підтримуються параметри введення та виведення інших рівнів, такі як одне дійсне число (Real), ціле число (Integer), вектор довжини n та багатовимірний масив.

In[53]:= NetInitialize@NetChain[{3,4,12,Tanh},"Input"→1]

Out[53]=

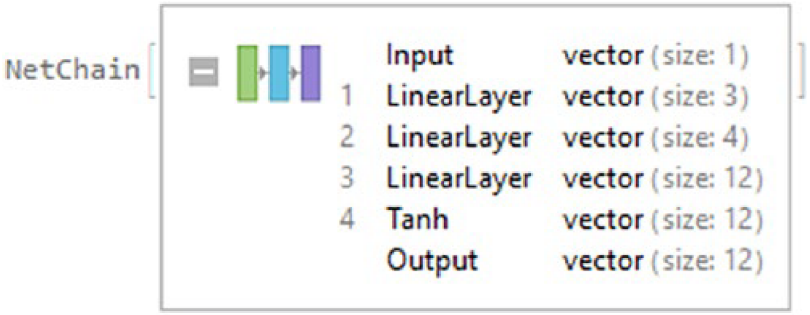


Рис. 2.11. NetChain з кількома рівнями

NetChain розпізнає імена функцій мови Wolfram Language і пов'язує їх з відповідними рівнями, такими як 3, 4 і 12. Вони є лінійним шаром з виходами розміром 3, 4 і 12 (Рис. 2.11). Функція Tanh є поелементний шар.

Давайте додамо шар у ланцюжок, створений за допомогою NetAppend (Рис. 2.12) або NetPrepend. Якщо ви помітили, багато вихідних команд мови Wolfram Language мають те саме значення - наприклад, для видалення в ланцюжку буде NetDelete [net_name, #_of_layer].

In[54]:= NetInitialize@NetChain[{1,ElementwiseLayer[LogisticSigmoid@#&]},

"Input"→ 1];

NetCH2=NetInitialize@NetAppend[%,{1,ElementwiseLayer[Cos[#]&]}]

Out[54]

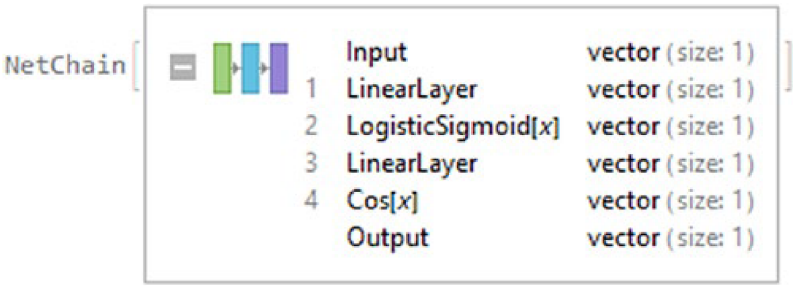


Рис. 2.12. Об'єкт NetChain з доданими шарами

Коли мережа застосовується до даних, доступні різні параметри, такі як NetEvaluationMode (режим оцінки, навчальний чи тестовий), TargetDevice та WorkingPrecision (числова точність).

In[55]:= NetCH2[{{0},{2},{44}},NetEvaluationMode→ "Train",TargetDevice→ "CPU",WorkingPrecision→ "Real64",RandomSeeding→ 8888](*use N@Cos[Sin [LogisticSigmoid[{0,2,44}]]] to check results*)

Out[55]= {{0.919044},{0.991062},{1.}}

Інша форма - запровадити явні імена шарів у ланцюжку. Це тип асоціації (Рис. 2.13).

In[56]:=NetInitialize@NetChain[<|"Linear Layer 1"→LinearLayer[3], "Ramp"→ Ramp,"Linear Layer 2"→LinearLayer[4],"Logistic"→ ElementwiseLayer[Logisti cSigmoid]|>,"Input"→ 3]

Out[56]=

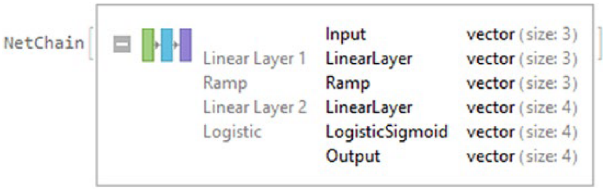


Рис. 2.13. Об'єкт NetChain з іменами кастомних шарів

Перевірка вмісту шару повинна з'явитися після натискання назви шару або шару.

Якщо шар отримається, NetExtract використовується разом з ім'ям відповідного шару.

In[57]:=NetExtract[%,"Logistic"];

Щоб витягти всі шари в один рядок коду, Normal виконає свою роботу (Рис. 2.14).

In[58]:= Normal[NetCH2]//Column

Out[58]=

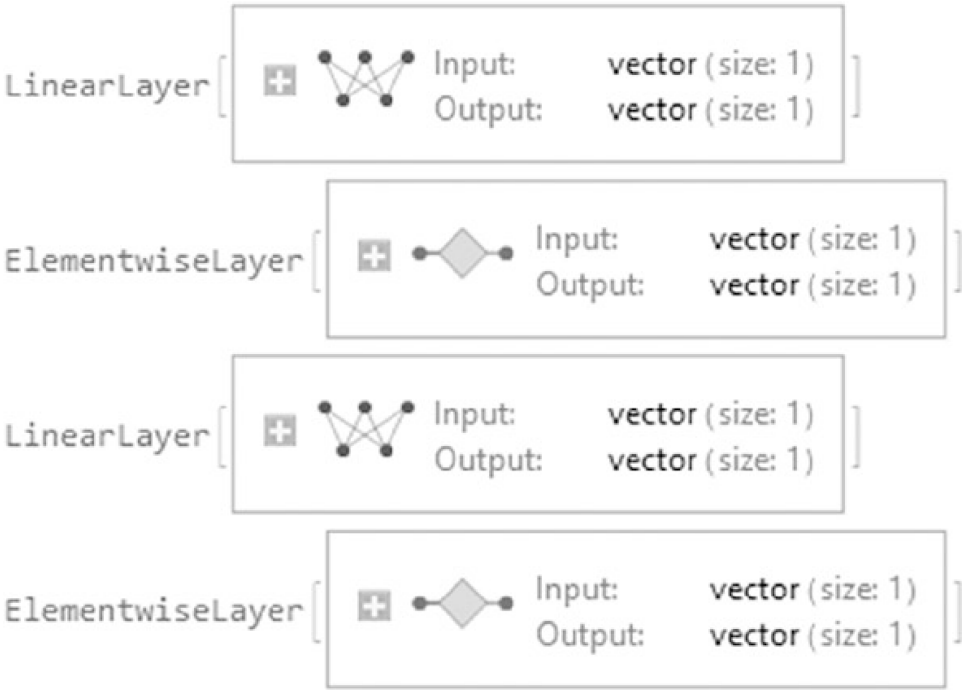


Рис. 2.14. Шари NetChain NetCH2

2.3.2 Декілька ланцюжків

Ланцюжки можна з'єднувати разом, як у випадку вкладеного ланцюжка (Рис. 2.15).

```
In[59]:=
chain1=NetChain[{12,SoftmaxLayer[]}];
chain2=NetChain[{1,ElementwiseLayer[Cos[#]&]}];
NestedChain=NetInitialize@NetChain[{chain1,chain2},"Input"→ 12]
Out[59]=
```

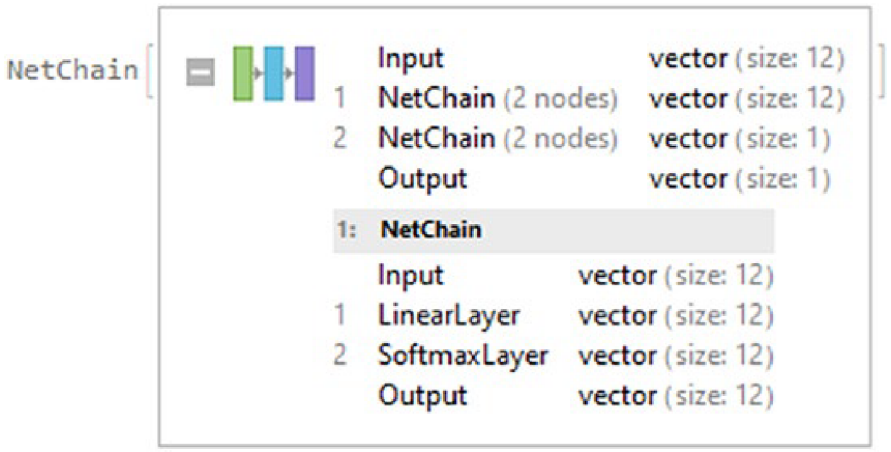


Рис. 2.15. Ланцюжок 1 вибраний з двох доступних ланцюжків

Цей ланцюжок розділений на дві NetChains, кожна з яких є окремим ланцюжком. У цьому випадку ми бачимо ланцюжки 1 і 2, і кожен ланцюжок показує відповідні вузли. Щоб налаштувати ланцюги, використовуємо NetFlatten (Рис. 2.16).

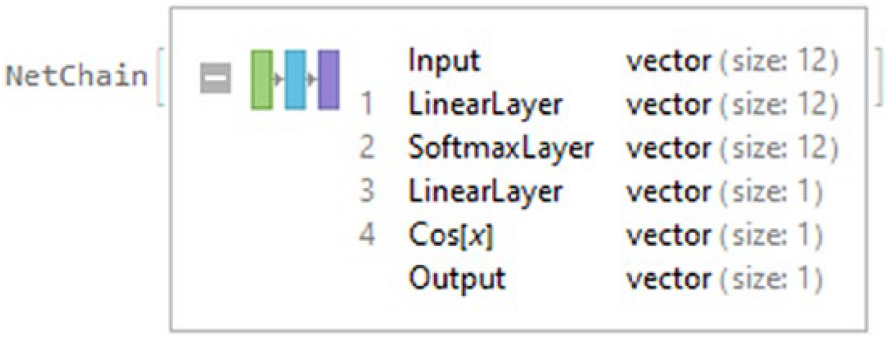


Рис. 2.16. Стиснутий ланцюжок

2.4 NetGraphs

Команда NetChain поєднує лише шари, в яких вихід одного шару пов'язаний із входом наступного шару. NetChain не працює при підключенні входів або виходів до інших рівнів; він працює лише з одним шаром. Щоб обійти цю проблему, потрібно використовувати NetGraph. Крім надання більшої кількості вхідних даних та рівнів, NetGraph представляє структуру та процес нейронної мережі у вигляді графіка (Рис. 2.17).

```
In[62]:= NetInitialize@NetGraph[{ LinearLayer["Output"→ 1,"Input"→
1],Cos,SummationLayer[]},{}]
Out[61]=
```

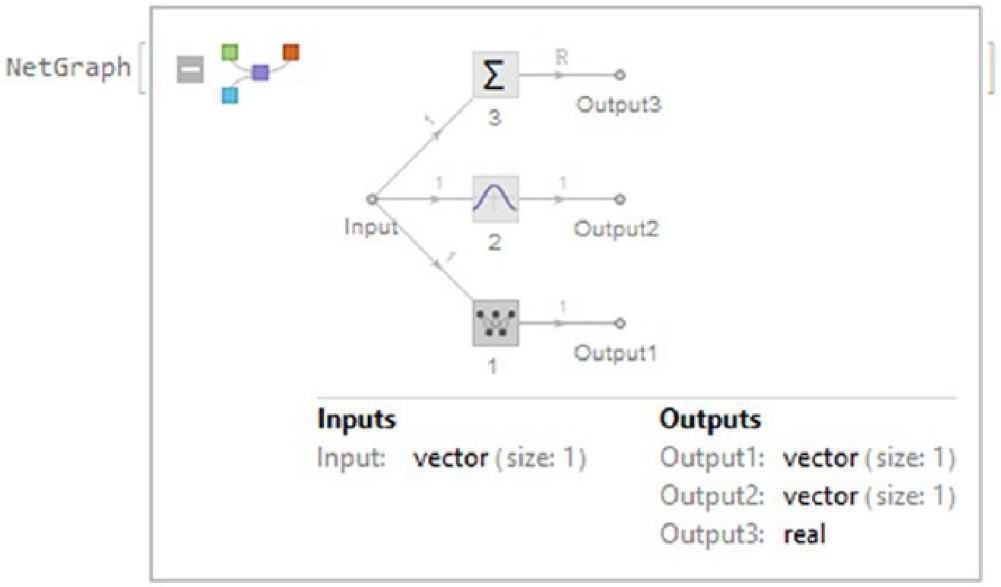


Рис. 2.17. Розширений NetGraph

Створений об'єкт - це NetGraph, і він представлений фігурою квадратів, що з'єднуються. Як показано на Рис. 2.17. Вхідний сигнал надходить на три різні шари, і кожен шар має свій вихід. NetGraph приймає два аргументи: перший призначений для шарів чи ланцюжків, а другий – для визначення вершин графа чи зв'язності мережі[11]. Наприклад, в останньому коді мережа має три виходи, тому що вершини не вказані. SumutationLayer - це шар, який підсумовує всі вхідні дані.

```
In[62]:= Net1=NetInitialize@NetGraph[{ LinearLayer["Output"→ 2,"Input"→
1],Cos,SummationLayer[]},{1→ 2→ 3}]
Out[62]=
```

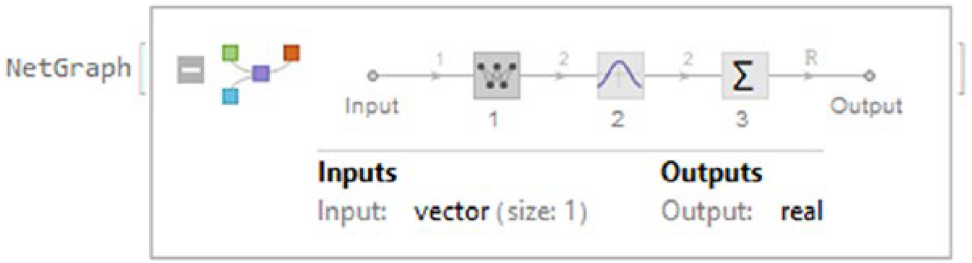


Рис. 2.18. Односпрямований NetGraph

Позначення вершини означає, що вихід одного шару передається іншому шару і т. д. Іншими словами, 1 → 2 → 3 означає, що вихідні дані лінійного шару передаються на наступний рівень доти, доки не будуть остаточно підсумовані в останньому шарі за допомогою SumutationLayer. Таким чином, зберігаючи порядок появи шарів, але ми можемо змінити порядок кожної вершини. Мережа можна змінити так, щоб виходи могли переходити на інші рівні мережі, наприклад, з 1 до 3, а потім на 2. З NetGraph шари та ланцюжки можна вводити у вигляді списку чи асоціації. Вершини набираються як списку правил.

```
In[63]:= Net2=NetInitialize@NetGraph[{ LinearLayer["Output"→ 2,"Input"→ 1],
Cos,SummationLayer[]},{1→ 3→2}]
Out[63]=
```

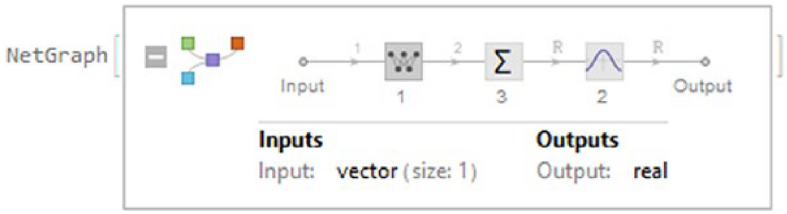


Рис. 2.19. Структура NetGraph

Входи та виходи кожного шару відмічені підказкою, що з'являється при наведенні курсору на лінії або вершини графа. У тому сенсі, що введення та виведення не вказані, NetGraph визначить тип даних у порту введення та виведення; така справа з великою R у вхідних і вихідних даних ElementwiseLayer, що означає real. З NetGraph шари можна вводити як список чи асоціацію. З'єднання набираються у вигляді списку правил (Рис. 2.20).

```
In[64]:= NetInitialize@NetGraph[<|"Layer 1"→ LinearLayer[2,"Input"→
1],"Layer 2"→ Cos,"Layer 3"→ SummationLayer[]|>,{"Layer 2"→ "Layer 1"→
"Layer 3"}]
Out[64]=
```

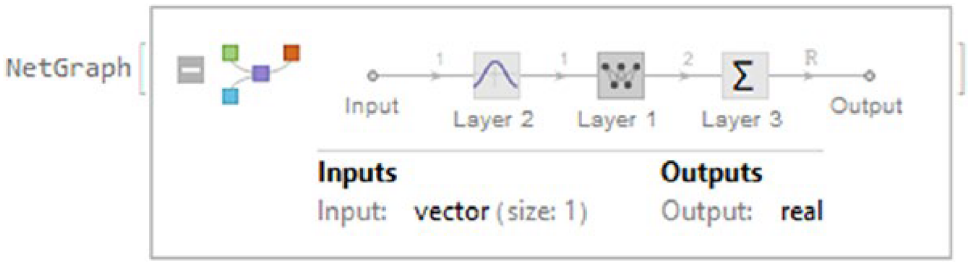


Рис. 2.20. NetGraph з іменованими шарами

Тепер можна вказати, скільки входів та виходів може мати структура за допомогою команди NetPort.

```
In[65]:= NetInitialize@NetGraph[{ LinearLayer[3,"Input"→ 1],
LinearLayer[3,"Input"→ 2], LinearLayer[3,"Input"→ 1],
TotalLayer[]},{NetPort["1st Input"]→ 1, NetPort["2nd Input"]→ 2,
NetPort["3rd Input"]→ 3,{1,2,3}→ 4}](*Or NetInitialize@NetGraph[<|"L1"→
LinearLayer[3,"Input"→,"L2"→ LinearLayer[3,"Input"→1], "L3"\[Rule]
LinearLayer[3,"Input"→ 1],"Tot L"→TotalLayer[]|>,{NetPort["1st Input"] →
"L1", NetPort["2nd Input"] → "L2",NetPort["3rd Input"] → "L3",
{"L1","L2","L3"} → "Tot L"}]*)
Out[65]=
```

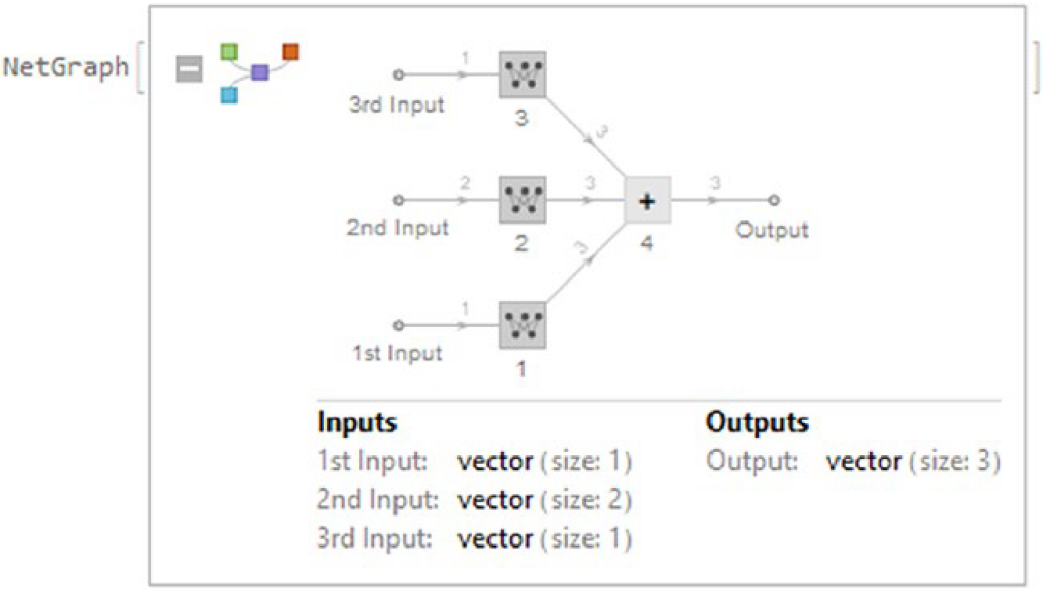


Рис. 2.21. NetGraph з кількома входами

Контейнери NetChain можна як шари за допомогою NetGraph (Рис. 2.22). Деякі рівні, такі як CatenateLayer, не приймають жодних аргументів.

```
In[69]= NetInitialize@NetGraph[{
LinearLayer[1,"Input"→ 1],
NetChain[{LinearLayer[1,"Input"→ 1],ElementwiseLayer[LogisticSigmoid[#]&]}],
NetChain[{LinearLayer[1,"Input"→ 1],Ramp}],
ElementwiseLayer["ExponentialLinearUnit"],
CatenateLayer[]
},{1→4,2→5,3→ 5,4→ 5}]
Ou[69]=
```

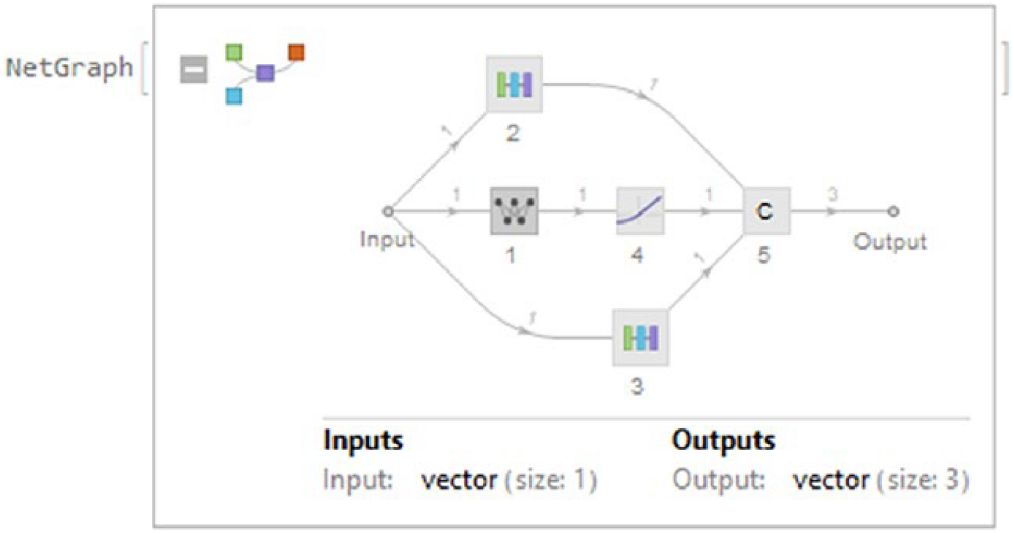


Рис. 2.22. NetGraph з кількома контейнерами

Клацання по ланцюжку або шару відобразить відповідну інформацію, а клацання по шару всередині ланцюжка надасть інформацію про шар, що міститься у вибраному ланцюжку.

За допомогою NetGraph, NetChains і NetGraphs можуть бути вкладені для формування різних структур, як показано в наступному прикладі, де за NetChain може слідувати NetGraph і навпаки. З графіка на Рис. 2.23 видно, що вхідні дані

надходять до NetGraph, а вихідні - у разі, якщо NetGraph переходить до NetChain. NetChain або NetGraph, які не були ініціалізовані, відображатимуться червоним кольором. Основною якістю контейнерів (NetChain, NetGraph) є те, що вони можуть поводитися як шар. Маючи це на увазі, ми можемо створювати вкладені контейнери, що включають лише NetChains, NetGraphs або те й інше.

Як демонстрацію з допомогою NetGraph можна створювати складніші структури, як показано на рис. Рис. 2.23. Після створення мережевої структури можна витягти властивості кожного шару чи ланцюжка. Наприклад, за допомогою «SummaryGraphic» можна отримати графіку мережного графіка.

```
In[71]:=Net=NetInitialize@NetGraph[{LinearLayer[10],Ramp,10,SoftmaxLayer[],
TotalLayer[],ThreadingLayer[Times]},{1→2→3→4,{1,2,3}→5,{1,5}→6},"Input
"→"Real"];
Information[Net,"SummaryGraphic"]
```


```
In[71]:=Net=NetInitialize@NetGraph[{LinearLayer[10],Ramp,10,SoftmaxLayer[],
TotalLayer[],ThreadingLayer[Times]},{1→2→3→4,{1,2,3}→5,{1,5}→6},"Input
"→"Real"];
Information[Net,"SummaryGraphic"]
```


```
Out[71]=
```

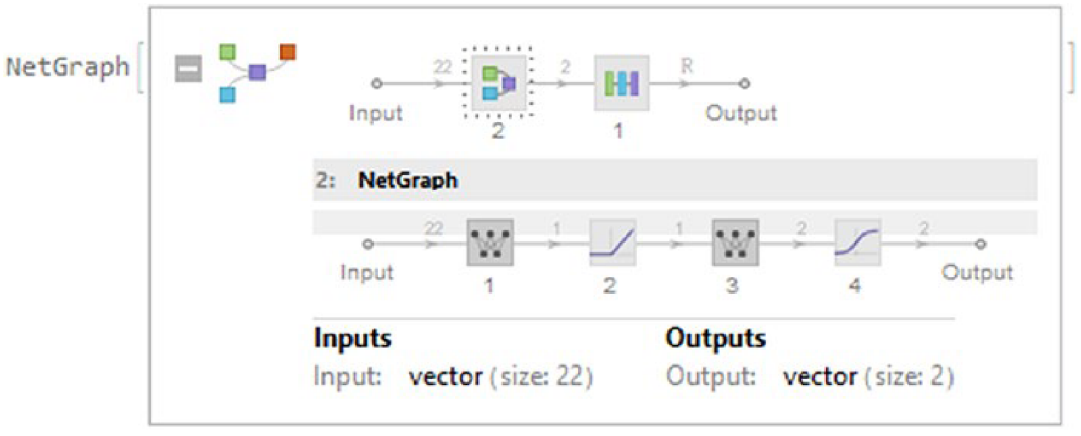


Рис. 2.23.1. Складова частина нейронної мережі

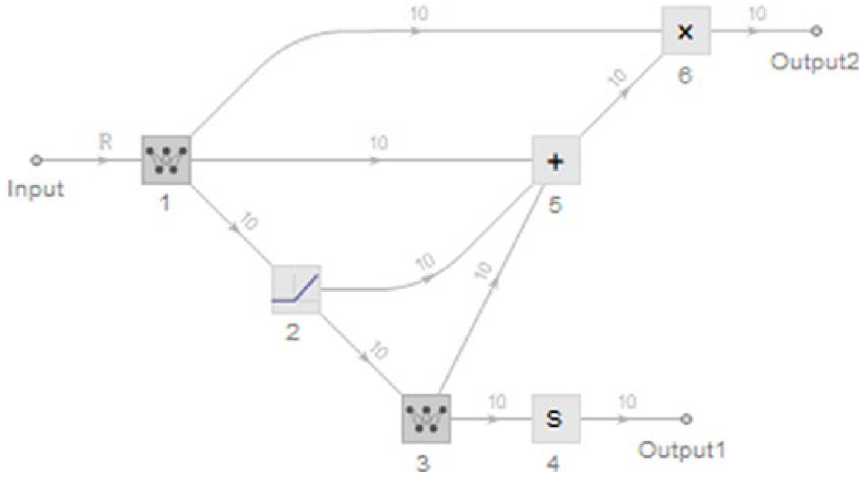


Рис. 2.23.2. Складова частина нейронної мережі

2.5 Властивості нейронної мережі

Властивості, пов'язані з числовими масивами мережі: Arrays (дає кожен масив у мережі), ArraysCount (кількість масивів у мережі), ArraysDimensions (розміри кожного масиву в мережі) та ArraysPositionList (положення кожного масиву в мережі).

```
In[72]:= {Dataset@Information[Net,"Arrays"],Dataset@Information[Net,"Arrays
Dimensions"],Dataset@Information[Net,"ArraysPositionList"]}//Dataset
Out[72]=
```

1	Biases	NumericArray [Type: Real32 Dimensions: {10}]
1	Weights	NumericArray [Type: Real32 Dimensions: {10, 1}]
3	Biases	NumericArray [Type: Real32 Dimensions: {10}]
3	Weights	NumericArray [Type: Real32 Dimensions: {10, 10}]

1	Biases	{10}
1	Weights	{10, 1}
3	Biases	{10}
3	Weights	{10, 10}

1	Biases
1	Weights
3	Biases
3	Weights

Табл. 1. Dataset, що містить різні властивості

Інформація, що стосується типу змінної у вхідних та вихідних портах, відображається за допомогою InputPorts та OutputPorts.

```
In[73]:= {Information[Net,"InputPorts"],Information[Net,"OutputPorts"]}
Out[73]= {<|Input→Real|>,<|Output1→10,Output2→10|>}
```

Ми бачимо, що вхідні дані є дійсними числами, а мережа має два вихідні вектори розміром 10. Найчастіше використовувані властивості, пов'язані з шарами, - це Layers (повертає кожен шар мережі), LayerTypeCounts (кількість входжень шару в мережу), LayersCount (кількість шарів у мережі), LayersList (список всіх шарів у мережі) і LayerTypeCounts (кількість входжень шару до мережі).

In[74]:=

```
Dataset@{Information[Net,"Layers"],Information[Net,"LayerTypeCounts"]}
```

Out[74]=

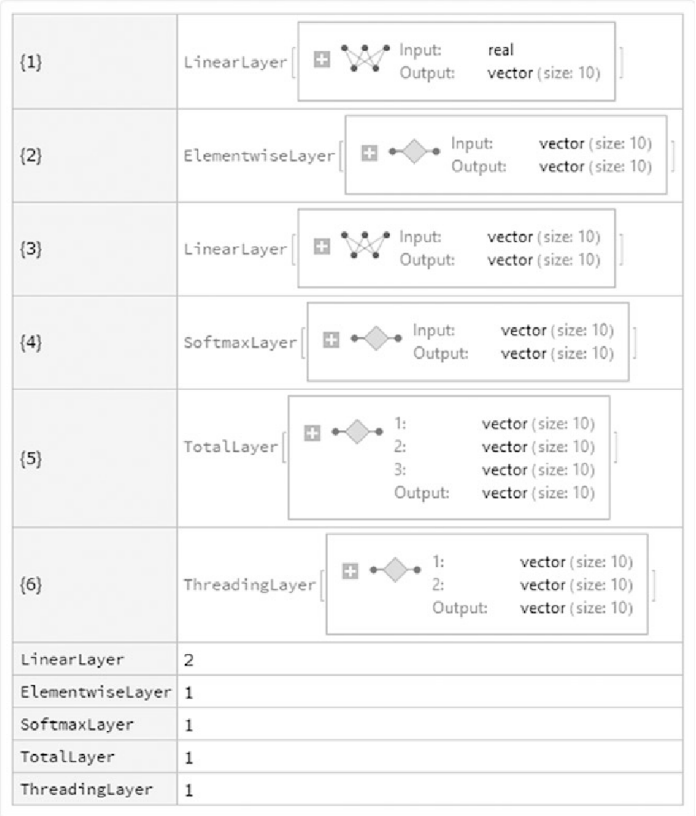


Рис. 2.24. Інформація про шари, що містяться в символі Net

Візуалізація мережевої структури (Рис. 2.24) досягається за допомогою властивостей LayerGraph (графік, що показує зв'язність шарів), SummaryGraphics (графічне зображення мережевої структури), MXNetNodeGraph (операції з необробленим графом MXNeT) та MXNetNodeGraphPlot (аннотації). MXNet - це фреймворк глибокого навчання з відкритим вихідним кодом, який підтримує безліч мов програмування, і однією з них є мова Wolfram Language. Крім того, Wolfram Neural Network Framework працює зі структурою MXNet як внутрішній підтримці.

In[75]:=

```
Grid[{{Style["Layers Connection",Italic,20,ColorData[105,4]],
Style["NetGraph",Italic,20,ColorData[105,4]]},{Information[Net,"LayersGraph
```

"],Information[Net,"SummaryGraphic"]},

{Style["MXNet Layer Graph",Italic,20,ColorData[105,4]],Style["MXNet Ops Graph",Italic,20,ColorData[105,4]]},

{Information[Net,"MXNetNodeGraph"],Information[Net,"MXNetNodeGraphPlot"]}},

Dividers→All,Background→{{{None,None}},{{Opacity[1,Gray],None}}}]

Normal@Keys@%

Out[75]=

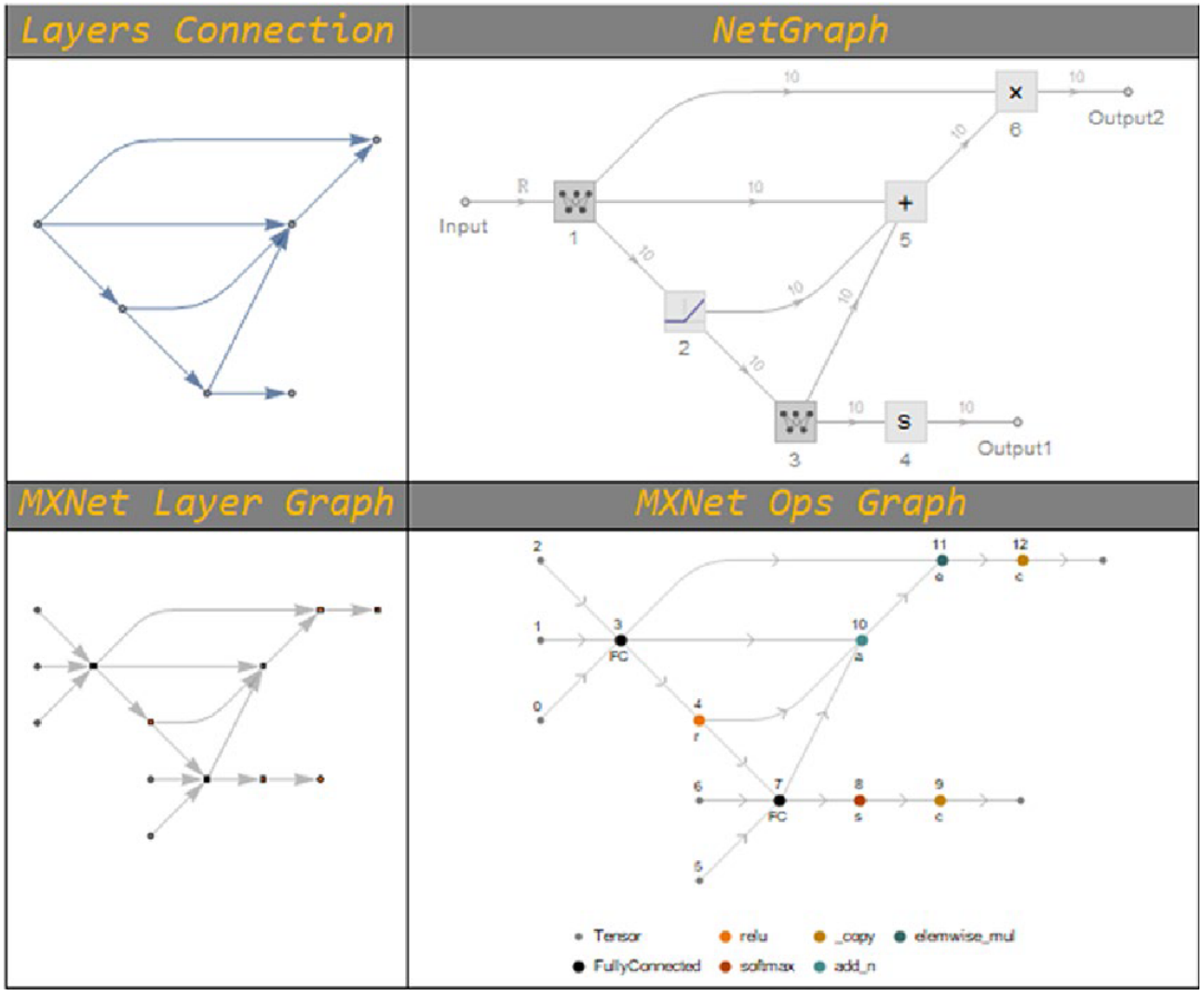


Рис. 2.25. Мережа з кількома графами

При наведенні курсору на шар або вузол на графіку символів MXNet відображається спливаюча підказка, яка показує властивості символів MXNet, такі як ідентифікатор, ім'я, параметри, атрибути та вхідні дані.

Завдяки сумісності мов Wolfram Language та MXNet, мова Wolfram Language підтримує імпорт та експорт нейронних мереж, ініціалізованих або неініціалізованих. Для цього ми створюємо на робочому столі папку з ім'ям MXNet Nets. Ми експортуємо мережу знайдену у змінній Net.

Для імпорту мережі MXNet рекомендується, щоб файл JSON та .params знаходилися в одній папці, оскільки мова Wolfram Language передбачає, що певний файл JSON буде збігатися із шаблоном файлу .params. Існують різні способи імпорту мережі, включаючи Імпорт [ім'я_файлу.json, «MXNet»] та Імпорт [ім'я_файлу.json, {«MXNet», елемент}] (те саме, що і з файлами .param).

Остання мережа була автоматично імпортована із файлом .params. Щоб імпортувати мережу без настройок, використовуйте для параметра ArrayPath значення None. Імпорт мережевих параметрів може бути виконаний за допомогою таких опцій: список (ArrayList), імена (ArrayNames) або як асоціація (ArrayAssociation). Це показано на Рис. 2.25.

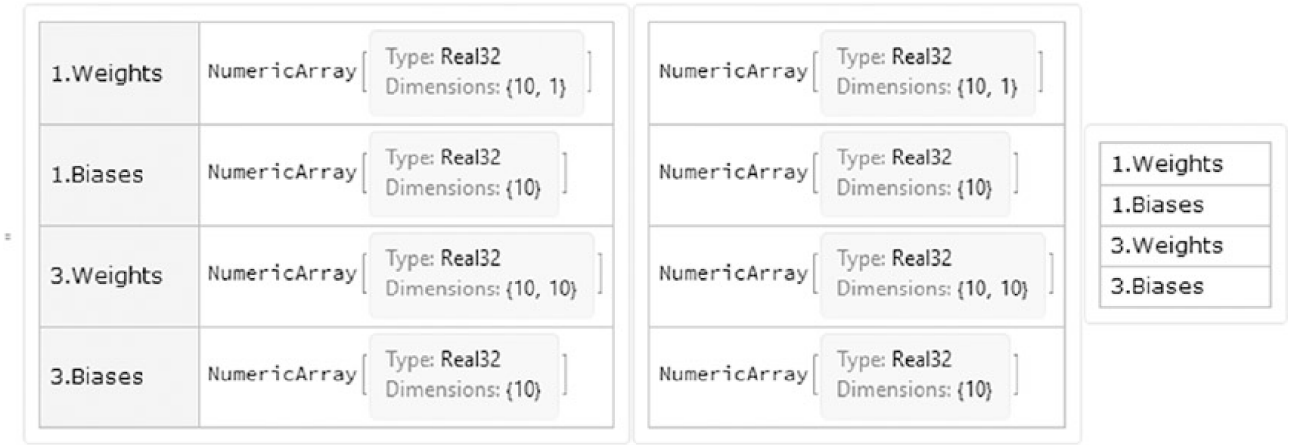


Рис. 2.26. Різні варіанти імпорту формату MXNet

Імпортовані елементи мережі: InputNames, LayerAssociation, Net (імпортувати мережу як NetGraph або NetChain), NodeDataset (набір даних вузлів MXNet), NodeGraph (граф вузлів MXNet), NodeGraphPlot (графік вузлів MXNet) та

UninitializedNet (те ж, що Array . У наступному наборі даних показано кілька параметрів, що перераховані перед Рис. 2.24.

	op	attrs	inputs
Input	null		
1.Weights	null		
1.Biases	null		
1	FullyConnected	2 total ›	{0, 0, 0}
			3 total ›
2$0	relu		{3, 0, 0}
3.Weights	null		
3.Biases	null		
3	FullyConnected	2 total ›	{4, 0, 0}
			3 total ›
4$0	softmax	1 total ›	{7, 0, 0}
5	ElementWiseSum	1 total ›	{3, 0, 0}
			3 total ›
6$0	_Mul		{3, 0, 0}
			2 total ›
Output1	identity		{8, 0, 0}
Output2	identity		{10, 0, 0}

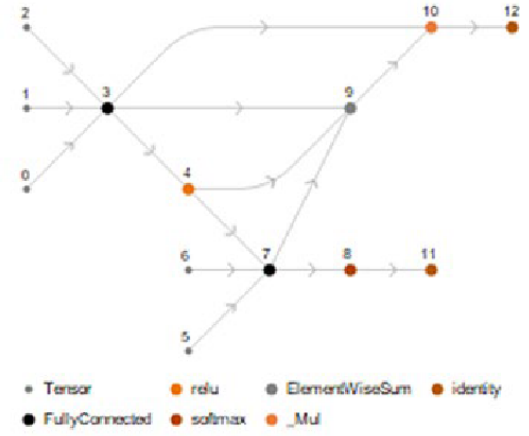


Рис. 2.27. Набір даних вузлів і граф операцій MXNet

In[80]= {Import[FileNameJoin[{FileDirectory,"MxNet.json"}],{"MXNet",
"NodeDataset"}],Import[FileNameJoin[{FileDirectory,"MxNet.json"}],
{"MXNet","NodeGraphPlot"}]}//Row
Out[80]=

Деякі операції між Wolfram Language та MXNet незворотні. Якщо ви звернете увагу, мережний вхід, експортований у формат MXNet, був заданий як дійсне число, на відміну від входу, імпортованого у форматі MXNet, який зазначає, що вхід є масивом без вказівки розмірів. При побудові нейронної мережі немає обмежень кількість ланцюжків ланцюгів чи мережевих графів, які може мати мережу. Наприклад, наступний приклад - нейронна мережа з репозиторію

нейронних мереж Wolfram, яка має глибше розуміння конструкції. Ця мережа називається CapsNet і використовується для оцінки карти глибини зображення. Щоб звернутися до мережі, введіть NetModel ["CapsNet, навчена даними MNIST", "DocumentationLink"] для веб-сторінки документації; для записника у хмарі Wolfram введіть NetModel [«CapsNet, навчена даними MNIST», «ExampleNotebookObject»] або просто ExampleNotebook для настільної версії.

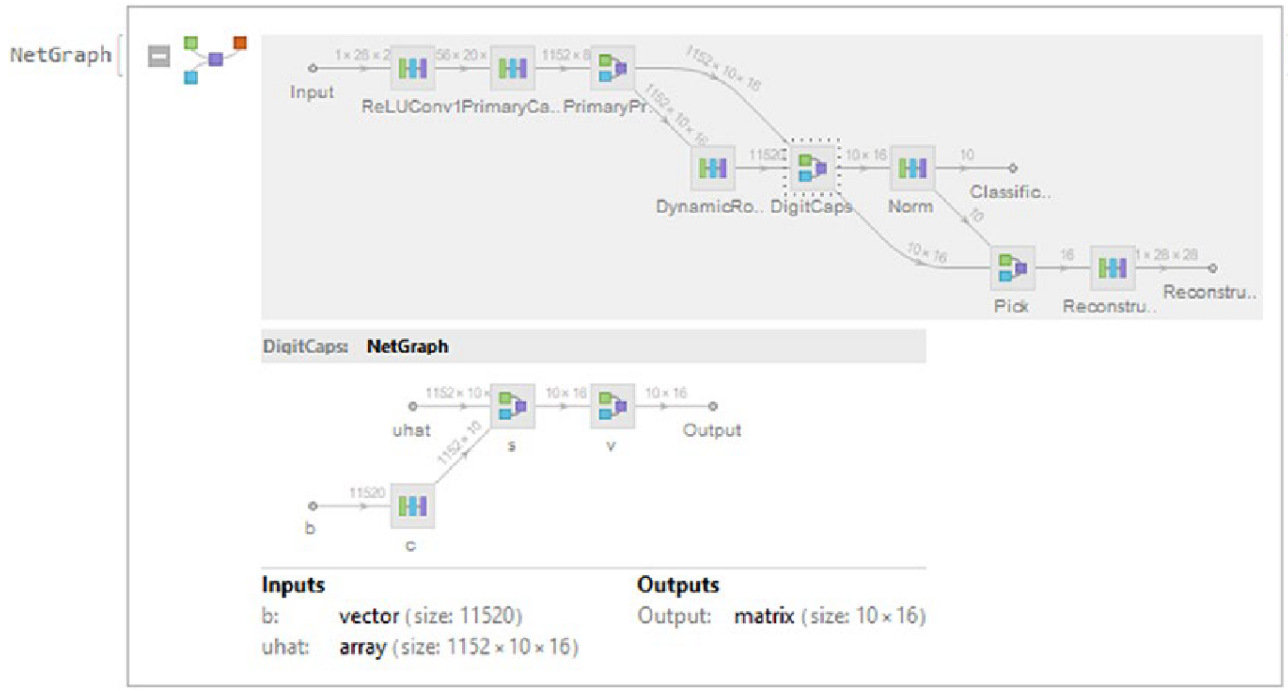


Рис. 2.28. Модель нейронної мережі CapsNet

3 ПРАКТИЧНА ЧАСТИНА

3.1 Розробка фреймворку нейронної мережі

У цьому розділі ми розробимо безпосередні методи того, як навчити модель нейронної мережі на мові Wolfram Language та як отримати доступ до результатів і самої навченої мережі. Ми розглянемо основні команди для експорту та імпорту мережевої моделі. Далі ми дослідимо структуру сховищ нейронних мереж Wolfram і окремо розглянемо мережеву модель LeNet.

Мова Wolfram Language містить дуже корисну команду, що автоматизує процес навчання моделі нейронної мережі. Це команда NetTrain. Навчання нейронної мережі полягає в точній настройці внутрішніх параметрів нейронної мережі. Вся справа в тому, що параметри можна дізнатися в процесі навчання мережі. Цей загальний процес виконується алгоритмом оптимізації, який називається градієнтним спуском. Він, в свою чергу, прораховується за допомогою алгоритму зворотного поширення помилки.

3.1.1 Введення даних

З NetTrain дані можна вводити в різних формах. Спочатку в якості першого аргументу йде мережева модель, за якою слідує Input → Target, {Inputs, ...} → {Targets, ...} або ім'я даних або набору даних. Після визначення мережевої моделі наступним аргументом є дані, за якими слід необов'язковий аргумент All.

Параметр All створить об'єкт NetTrainResultsObject, який використовується для відображення панелі результатів NetTrain після обчислення і для зберігання всієї необхідної інформації про навченої моделі. Опції для навчання моделі вводяться в якості останніх аргументів. Загальні параметри, які використовуються в шарах і контейнерах, доступні в NetTrain.

У наступному прикладі ми будемо використовувати модель персептрона для побудови лінійного класифікатора. Класифіковані дані подані в наступній діаграмі.

In[1]:=

```
Plt=ListPlot[{{{-1.8,-1.5},{-1,-1.7},{-1.5,-1},{-1,-1},{-0.5,-1.2},
{-1,-0.7}},{{1,1},{1.7,1},{0.5,2},{0.1,0.3},{0.5,1},{0.6,1.3}}},
PlotMarkers→"OpenMarkers",Frame→True, PlotStyle→{Green,Red}]
```

Out[1]=

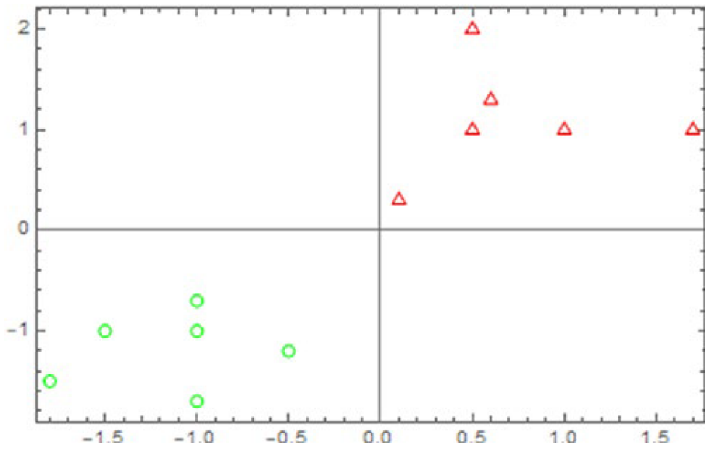


Рис. 3.1. ListPlot, що показує дві різні точки графіка

Тепер визначимо дані, цільові значення і дані навчання.

In[2]

```
Data={{-1.8,-1.5},{-1,-1.7},{-1.5,-1},{-1,-1},{-0.5,-1.2},{-1,-0.7},{1,1},
{1.7,1},{0.5,2},{0.1,0.3},{0.5,1},{0.6,1.3}};
Target={-1,-1,-1,-1,-1,-1,1,1,1,1,1,1};
TrainData=MapThread[#1→ {#2}&,{Standardize[Data],Target},1];
```

Тепер визначимо мережеву модель.

```
In[3]:= Model=NetChain[{LinearLayer[1,"Input"→ 2], ElementwiseLayer[Ra
mp[#]&]}];
```

3.1.2 Фаза навчання

Підготувавши дані і модель, приступаємо до навчання моделі. Як тільки навчання починається, з'являється інформаційна панель з чотирма основними результатами.

1. Summary: містить релевантну інформацію про партії, раундах і терміни.
2. Data: включає оброблену інформацію про дані.
3. Method: показує використовуваний метод, розмір партії і пристрій, що використовується для навчання.
4. Round: поточний стан розміру збитку.

```
In[4]:= Net=NetTrain[Model,TrainData,All,LearningRate→0.01,PerformanceGoal
→"TrainingSpeed",TrainingProgressReporting→"Panel",TargetDevice→"CPU",
RandomSeeding→88888,WorkingPrecision→"Real64"]
Out[4]=
```

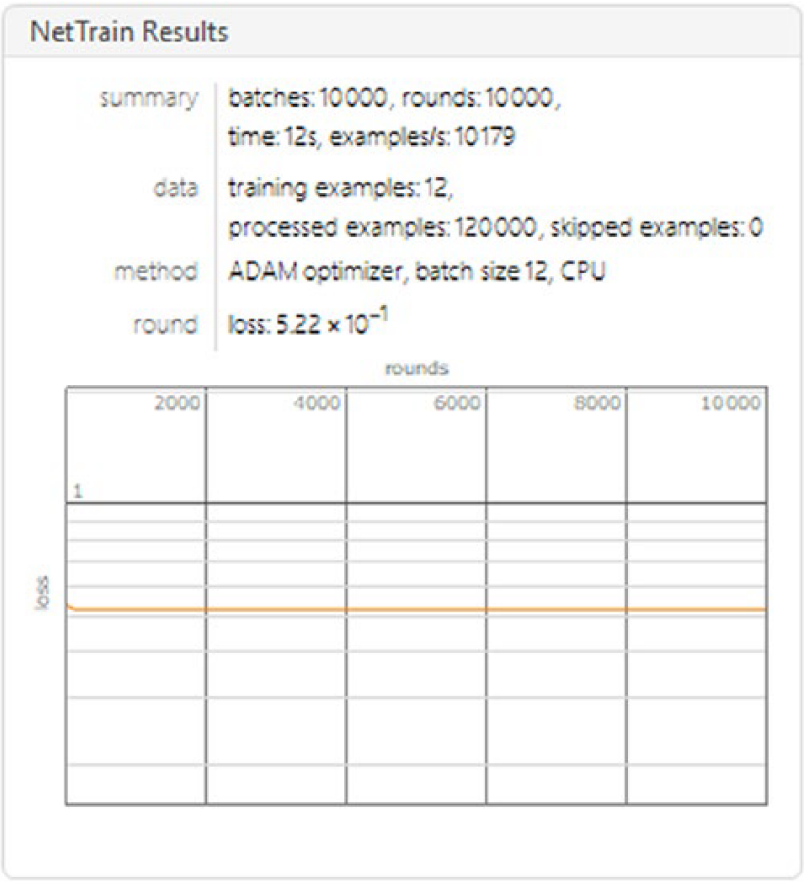


Рис. 3.2. NetTrainResultsObject

На Рис. 3.2 показаний графік втрат в залежності від навчальних раундів. Оптимізатор Adam - це варіант стохастичного градієнтного спуску, який буде застосовано далі. Створений об'єкт називається NetTrainResultsObject.

3.2. Імплементація моделі

Після того, як навчання пройдено, отримання навченої мережі та реалізації моделі виглядає наступним чином на Рис. 3.3.

In[5]:= TrainedNet1=Net["TrainedNet"]

Out[5]=

Рис. 3.3. Виведена навчена мережа

Тепер продемонструємо, як навчена мережа ідентифікує кожну з точок, викреслюючи кордон з графіком розподілу щільності.

In[6]:= Show[DensityPlot[TrainedNet1[{x,y}],{x,-2,2},{y,-3,3},PlotPoints→50,ColorFunction→(RGBColor[1-#,2*#,1]&)],Plt]

Out[6]=

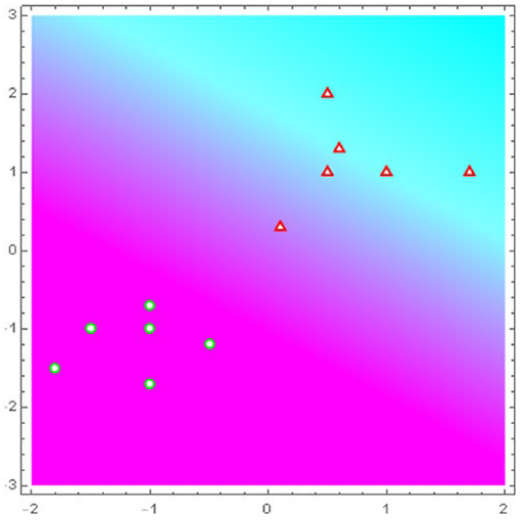


Рис. 3.4. Графік класифікації нейронної мережі

Дивлячись на графік, ми бачимо, що межі нечітко визначені і точки, близькі до нуля, можуть бути неправильно класифіковані. Це може бути пов'язано з тим, що функція ramp видає 0, якщо отримує будь-який негативний число, але для будь-якого позитивного значення вона повертає це значення. Ми бачимо, що цю модель все ж можна поліпшити, можливо за іншим призначенням активації на гіперболічний тангенс, щоб мати надійні межі.

3.2.1 Розміри пакетів і раунди

Якщо розмір пакета не вказано, він буде мати автоматичне значення, майже завжди значення воно рівне 64 або ступінь двійки. Слід розуміти, що розмір пакету вказує кількість прикладів, які модель використовує при навчанні перед оновленням внутрішніх параметрів моделі. Кількість пакетів - це поділ прикладів в навчальному наборі даних за розміром пакета[12,13]. Оброблені приклади - це кількість раундів (епох), помножене на кількість навчальних прикладів. Як правило, розмір пакета вибирається таким чином, щоб він рівномірно ділив розмір навчального набору.

Параметр MaxTrainingRounds визначає кількість проходів навчального набору даних на етапі навчання. Коли ви проходите весь навчальний набір тільки один раз, це називається епохою. Щоб краще зрозуміти це, в попередньому прикладі автоматично був обраний розмір пакета, що дорівнює 12, що дорівнює кількості прикладів в навчальному наборі. Це означає, що для епохи або раунду входить пакет з 12/12 -> 1. Тепер кількість епох було автоматично вибрано рівним 10000, це говорить нам, що буде 1 * 10000 пакетів. Крім того, кількість оброблених прикладів буде 12 * (10000), що дорівнює 120000. Якщо розмір пакета не ділить навчальний набір рівномірно, це означає, що в остаточному пакеті буде менше прикладів, ніж в інших пакетах[14]. Більш того, додавання шару функції втрат до контейнера або додавання втрат за допомогою опції LossFunction -> «Рівень втрат» має той же ефект. У цьому випадку ми будемо використовувати MeanSquaredLossLayer в якості опції функції втрат і змінимо функцію активації на Tanh[x]. І встановимо Batchsize -> 5 і MaxTrainingRounds -> 1000.

```
In[7]:= Net2=NetTrain[NetChain[{
LinearLayer[1,"Input"-> 2],ElementwiseLayer[Tanh[#]&]}],TrainData,All,Lear
```

ningRate->0.01,PerformanceGoal->"TrainingSpeed",TrainingProgressReporting-
>"Panel",TargetDevice->"CPU",RandomSeeding->88888,
WorkingPrecision->"Real64",LossFunction->MeanSquaredLossLayer[],BatchSize->5,
MaxTrainingRounds->1000]
Out[7]=

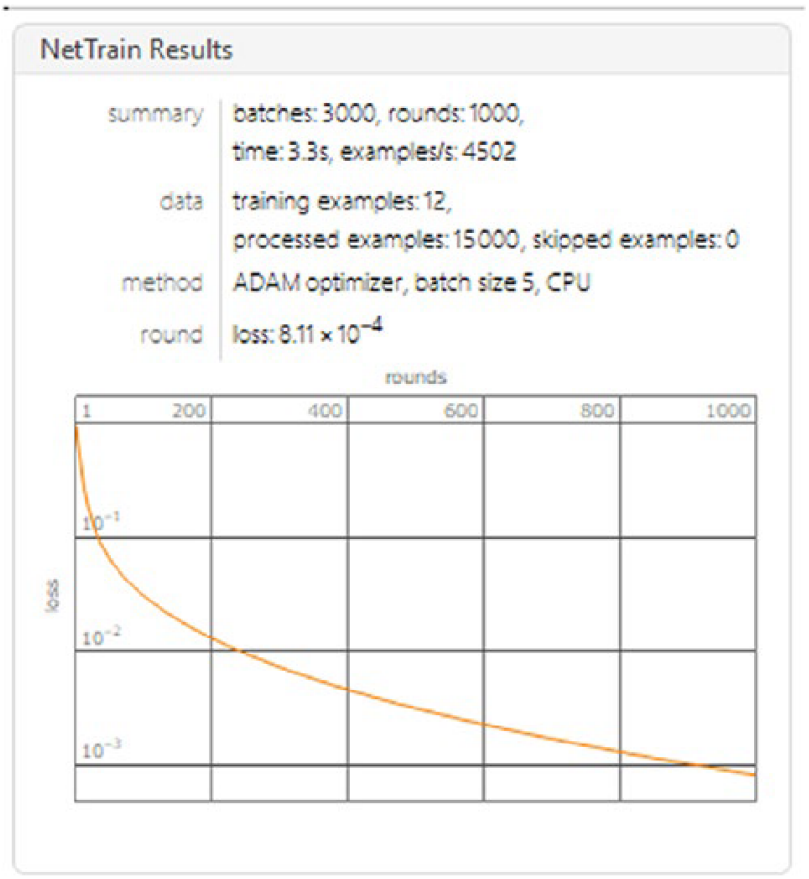


Рис. 3.5. Результати навчання Net2

Ми бачимо, що втрати значно знизились (Рис. 3.5). Подивимося, як влаштована класифікація.

In[8]:= TrainedNet2=Net2["TrainedNet"];
Show[DensityPlot[TrainedNet2[{x,y}],{x,-2,2},{y,-3,3},PlotPoints-
>50,ColorFunction->(RGBColor[1-#,2*#,1]&)],Plt]
Out[8]=

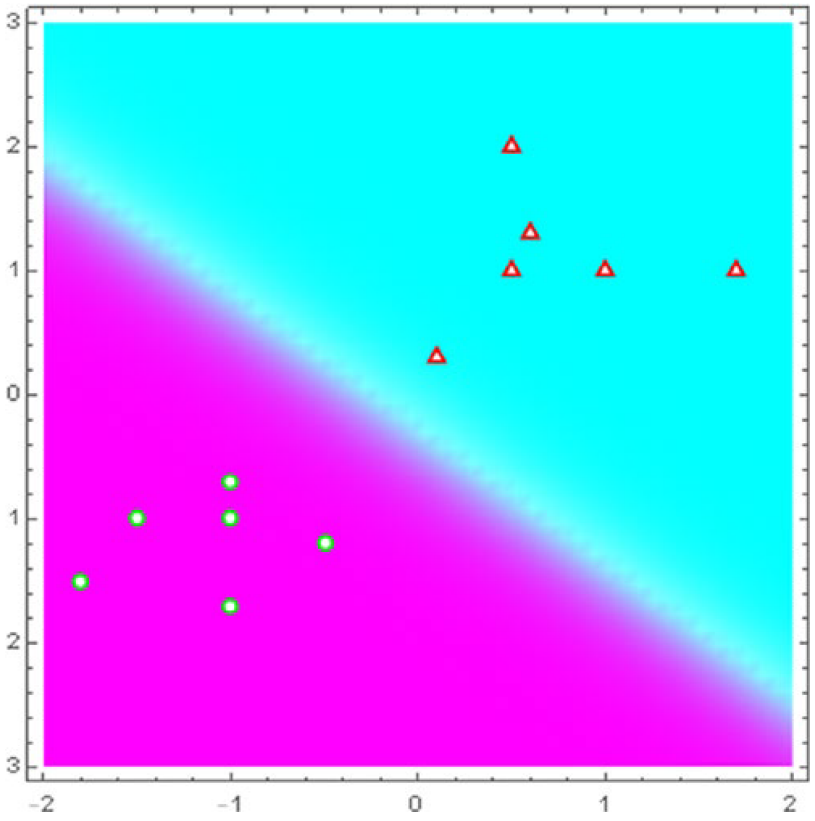


Рис. 3.6. Графік класифікації для мережі Net2

Ми можемо бачити, як краще тепер означені дві границі(Рис. 6.). Попередні моделі являють собою прогноз лінійного шару, в якому ця класифікація порівнюється з цільовими значеннями, так що помилка стає все меншою і меншою.

Щоб отримати графік, який показує значення помилки відповідно до кількості раундів, які виконуються в навчанні, ми робимо це через властивості навченої мережі[15]. Ми також можемо побачити, як виглядає мережева модель після додавання функції втрат.

```
In[9]:= Dataset[{Association["LossPlot"-> Net2["LossPlot"]],Association
["NetGraph"-> Net2["TrainingNet"]]}]
Out[9]=
```

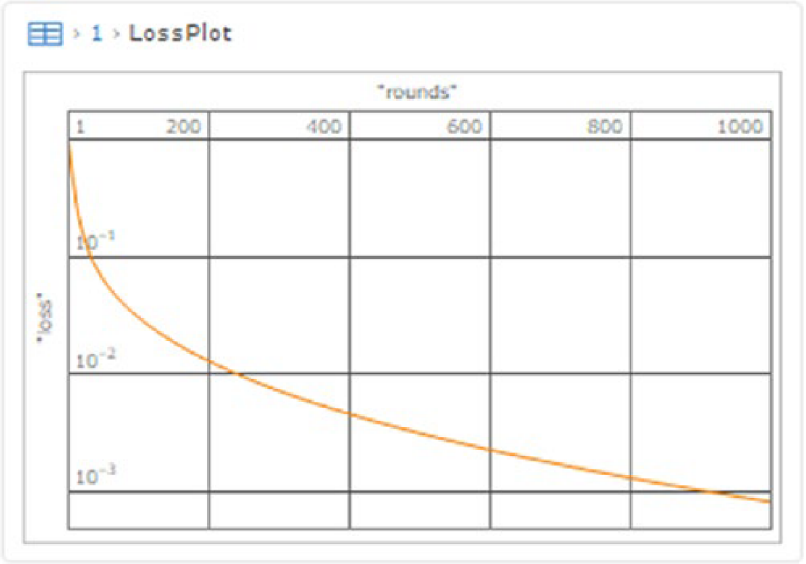


Рис. 3.7. LossPlot, що міститься в наборі даних

На Рис. 7. ми бачимо графік втрат, які швидко зменшуються в залежності від кількості раундів. Щоб побачити мережу, яка використовується для навчання, слід скомпілювати наведений далі код. Mathematica автоматично додає функцію втрат в нейронну мережу (Рис. 3.8) на основі шарів моделі.

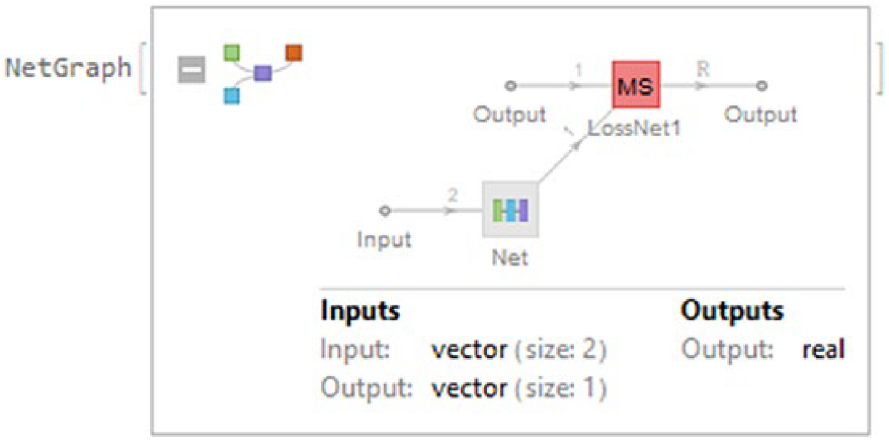


Рис. 3.8. Мережева модель до етапу навчання

Щоб побачити всі властивості моделі, ми додаємо рядок Properties в якості аргументу.

```
In[11]:= Net2["Properties"]
```

Out[11]= {ArraysLearningRateMultipliers,BatchesPerRound,BatchLossList,Batch
Measurements,BatchMeasurementsLists,BatchSize,BestValidationRound, Checkpoint
ingFiles,ExamplesProcessed,FinalLearningRate,FinalPlots,InitialLearningRate,
InternalVersionNumber,LossPlot,MeanBatchesPerSecond,MeanExamplesPerSecond,
NetTrainInputForm,OptimizationMethod,ReasonTrainingStopped,RoundLoss,Round
LossList,RoundMeasurements,RoundMeasurementsLists,RoundPositions,Skipped
TrainingData,TargetDevice,TotalBatches,TotalRounds,TotalTrainingTime,TrainedNet,
TrainingExamples,TrainingNet,TrainingUpdateSchedule,ValidationExamples,
ValidationLoss,ValidationLossList,ValidationMeasurements,ValidationMeasure
mentsLists, ValidationPositions}

3.3 Метод навчання

Розглянемо застосування методу навчання для попередньо розробленої нами нейронної мережі за допомогою OptimizationMethod. Існують варіанти алгоритму градієнтного спуску, які пов'язані з розміром пакета[16]. Перший - це стохастичний градієнтний спуск (SGD). SGD приймає один навчальний пакет за раз, перш ніж зробити наступний крок. Цей алгоритм переглядає навчальні приклади в стохастичною формі, тобто без послідовного шаблону і тільки по одному прикладу за раз.

Другий варіант - це пакетний градієнтний спуск, що означає, що розмір пакету встановлюється рівним розміру навчального набору[17]. У цьому методі використовуються всі навчальні приклади і виконується тільки одне оновлення внутрішніх параметрів.

І третій варіант - це міні-пакетний градієнтний спуск, який складається з поділу навчального набору на частини менші, ніж весь набір даних, щоб часто

оновлювати внутрішні параметри моделі для досягнення збіжності. Суть математичних розрахунків для SGD та SGD для міні-пакетів більш детально описана у праці: M. Li, T. Zhang, Y. Chen, and A.J. Smola. Efficient Mini-Batch Training for Stochastic Optimization, Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, 2014.

```
In[12]:= Net2["OptimizationMethod"]
Out[12]= {ADAM,Beta1->0.9,Beta2->0.999,Epsilon->1/100000,GradientClipping-
>None,L2Regularization->None,LearningRate->0.01,LearningRateSchedule-
>None,WeightClipping->None}
```

Автоматично вибирається метод оптимізатора ADAM, який використовує метод SGD із адаптованою швидкістю навчання. Інші доступні методи – це RMSProp, SGD та SignSGD. Серед доступних методів є варіанти для вказівки швидкості навчання, коли масштабувати, коли використовувати регуляризацію L2, градієнт і обмеження ваги.

3.4 Вимірювання продуктивності

Крім методів ми можемо встановити, які заходи слід враховувати на етапі навчання. Ці параметри залежать від типу функції втрат, яка внутрішньо пов'язана з типом завдання, наприклад, класифікація, регресія, кластеризація і т.д. абсолютне значення середнього залишку[18]. MeanSquare – це середній квадрат залишків, RSquared – коефіцієнт детермінації, а стандартне відхилення – це середньоквадратичне значення залишків. Після закінчення навчання міра з'явиться у чистих результатах.

```
In[13]:= Net3=NetTrain[NetChain[{LinearLayer[1,"Input"-> 2],Elementwise
Layer["SoftSign"]}],TrainData,All,LearningRate->0.01,PerformanceGoal->
```

"TrainingSpeed",TrainingProgressReporting->"Panel",TargetDevice->"CPU",RandomSeeding->88888,WorkingPrecision->"Real64",
Method->"ADAM",LossFunction->MeanSquaredLossLayer[],BatchSize->5,MaxTrainingRounds->1000,TrainingProgressMeasurements->{"MeanDeviation","MeanSquare","RSquared","StandardDeviation"}]
Out[13]=

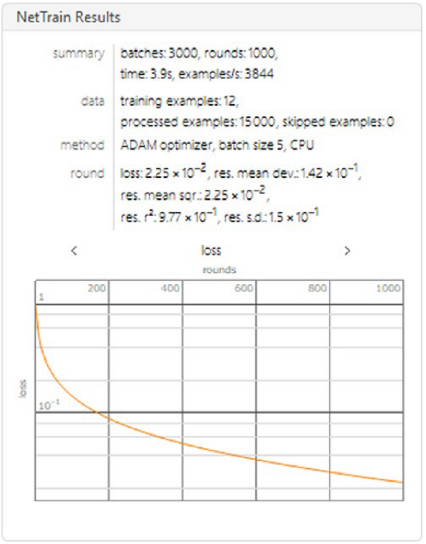


Рис. 3.9. Чисті результати з додаванням нових показників

3.4.1 Оцінка моделі

Щоб отримати доступ до значень вибраних показників, використовується опція NetResultsObject. У разі значень навчального набору вони перебувають у властивостях RoundLoss (дає середнє втрат), RoundLossList (повертає середні значення втрат під час навчання), RoundMeasurements (виміри навчання останнього round) і RoundMeasurementsLists (вказані вимірювання кожного раунду). Це зображено на Рис. 3.10.

In[14]:= Net3[#]&/@{"RoundMeasurements"}//Dataset[#]&

Out[14]=

Loss	0.0224962
MeanDeviation	0.141848
MeanSquare	0.0224962
RSquared	0.977403
StandardDeviation	0.149987

Рис. 3.10. Набір даних з новими показниками

Щоб отримати всі графіки, скористайтємося опцією FinalPlots.

In[15]:= Net3["FinalPlots"]//Dataset;

Щоб відтворити графіки вимірювань, отримаємо значення вимірювання кожного раунду за допомогою RoundMeasurementsLists.

In[16]:=Measures=Net3[#]&/@{"RoundMeasurementsLists"};

Keys[Measures]

Out[16]= {{Loss,MeanDeviation,MeanSquare,RSquared,StandardDeviation}}

Давайте побудуємо значення для кожного раунду, починаючи з Loss і закінчуючи StandardDeviation. Ми також можемо встановити підхід, як мережева модель робить межі класифікації

In[17]:=

TrainedNet3=Net3["TrainedNet"];

Grid[{{ListLinePlot[{Measures[[1,1]](*Loss*),Measures[[1,2]]

(*MeanDeviation*),Measures[[1,3]](*MeanSquare*),Measures[[1,4]]

(*RSquared*),Measures[[1,5]](*StandardDeviation*)},PlotStyle->Tab

le[ColorData[101,i],{i,1,5}],Frame->True,FrameLabel->{"Number of

Rounds",None},PlotLabel->"Measurements Plot",GridLines->All,

wPlotLegends->SwatchLegend[{Style["Loss",#],Style["MD",#],Style["MS",#],

```
Style["RS",#],Style["STD",#]},LegendLabel->Style["Measurements",#],
LegendFunction->(Framed[#,RoundingRadius->5,Background->LightGray]&)],
ImageSize->Medium]&[Black],Show[DensityPlot[TrainedNet3[{x,y}],{x,-2,2},{y,-3,3},
PlotPoints->50,ColorFunction->(RGBColor[1-#,2*#,1]&)],Plt,ImageSize-> 200]}}]
Out[17]=
```

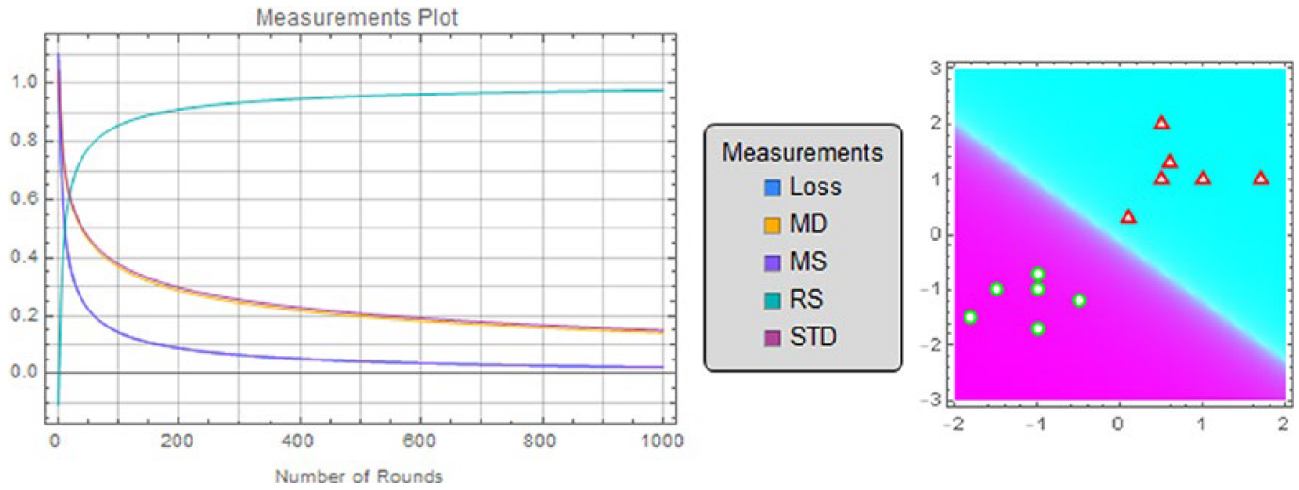


Рис. 3.11. Графік кількості раундів і графік щільності

Loss та MeanSquared мають однакові значення, тому два графіки перекриваються. У разі середнього відхилення та стандартного відхилення вони мають схожі значення, але не однакові. Слід зауважити, що ми будуємо три моделі, змінюючи активацію функції у кожному процесі. Дивлячись на графіки, ми можемо побачити, як кожна функція змінює спосіб навчання моделі нейронної мережі даних навчання.

Перед цим ми отримали графіки, що по-суті були графіком втрат для процесу навчання. У наступному розділі ми побачимо, як побудувати графік втрат і графік перевірки під час фази навчання, щоб перевірити, що мережева модель дійсно навчається під час навчання та наскільки добре модель може працювати з даними, яких ніколи раніше не було (набір для перевірки).

3.4.2 Експорт нейронної мережі

Після навчання моделі мережі ми будемо експортувати цю навчену мережу у хмарний ресурс WLNet, щоб у майбутньому мережу можна було використовувати без необхідності навчання. Метод експорту також працює для неініціалізованих мережевих архітектур.

```
In[18]:=Export["C:\\Users\\My-pc\\Desktop\\TrainedNet3.
wlnet",Net3["TrainedNet"]]
Out[18]= C:\Users\My-pc\Desktop\TrainedNEt.wlnet
```

Імпорт їх назад виконується так само, як і будь-який інший файл, але можна вказати імпортовані елементи. Net імпортує мережеву модель та всі ініціалізовані масиви; UninitializedNet і ArrayList імпортують об'єкти числового масиву лінійних шарів; ArrayAssociation імпортує числові масиви у формі асоціації, а WLVersion використовується для перегляду версії мови Wolfram Language, яка використовується для побудови мережі. Усі параметри відображаються у наступному наборі даних.

```
In[19]:= Dataset@ AssociationMap[Import["C:\\Users\\My-pc\\Desktop\\
TrainedNet3.wlnet",#]&,{"Net","UninitializedNet","ArrayList","Array
Association","WLVersion"}]
Out[19]=
```

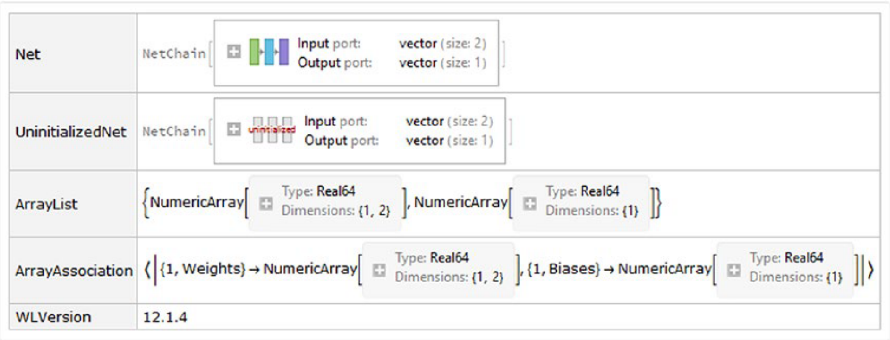


Рис. 3.12. Набір даних із доступними параметрами імпорту

3.4.3 Репозитарій нейронної мережі Wolfram

Репозитарій нейронних мереж Wolfram - це веб-сайт із безкоштовним доступом, який містить репертуар безлічі попередньо навчених моделей нейронних мереж. Моделі класифікуються за типом вхідних даних, які вони отримують, і типом даних, будь то аудіо, зображення, числовий масив, або текст. Крім того, вони також класифікуються за типом завдання, що виконується: від аудіоаналізу або регресії до класифікації. Головна сторінка сайту представлена на Рис. 3.13.

Рис. 3.13. Домашня сторінка репозиторію нейронної мережі Wolfram

Щоб отримати доступ до веб-сторінки, сслід ввести наступну URL-адресу в браузері https://resources.wolframcloud.com/NeuralNetRepository/ або запустимо SystemOpen з Mathematica, який відкриє веб-сторінку в браузері системи за замовчуванням.

```
In[20]:=SystemOpen["https://resources.wolframcloud.com/
NeuralNetRepository/"];
```

Як тільки сайт завантажений, мережні моделі можуть бути переглянуті за допомогою введення або за допомогою завдання. Моделі, знайдені у цьому репозиторії, побудовані мовою Wolfram Language, що дозволяє використовувати їх у системі Mathematica. Це призводить до знаходження моделей у формі, до якої можна отримати доступ з Mathematica або Wolfram Cloud для швидкого виконання. Якщо ми прокрутимо вниз, ми побачимо, що моделі структуровані на ім'я та дані, що використовуються для навчання, разом з коротким описом. Проілюструємо це для розробленої нами мережі Wolfram AudioIdentify V1, яка навчається за допомогою даних AudioSet і визначає звуки в аудіосигналах. Щоб переглянути категорії, ми можемо вибрати категорію з меню. На Рис. 14 показано, як виглядає сайт після вибору вхідної категорії - в даному випадку нейронні мережі, які отримують зображення як вхідні дані.

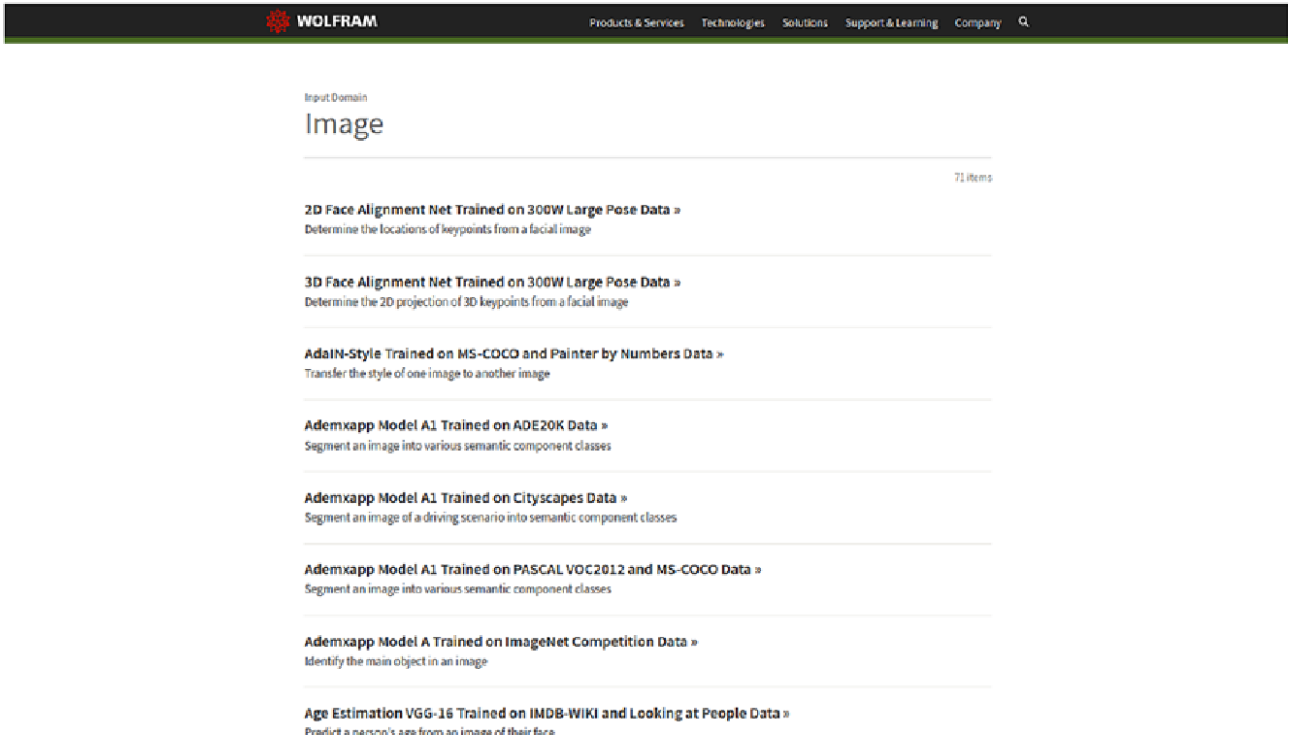


Рис. 3.14. Категорія на основі вхідного зображення

3.4.4 Вибір моделі нейронної мережі

Після вибору категорії будуть показані всі ланцюжки моделей, пов'язані з обраною вхідною категорією. Як і у випадку з репозиторієм даних Wolfram, після вибору моделі в ній відображатиметься відповідна інформація, як на Рис. 3.15, де обраною мережевою моделлю є нейронна мережа Wolfram ImageIdentify Net V1.

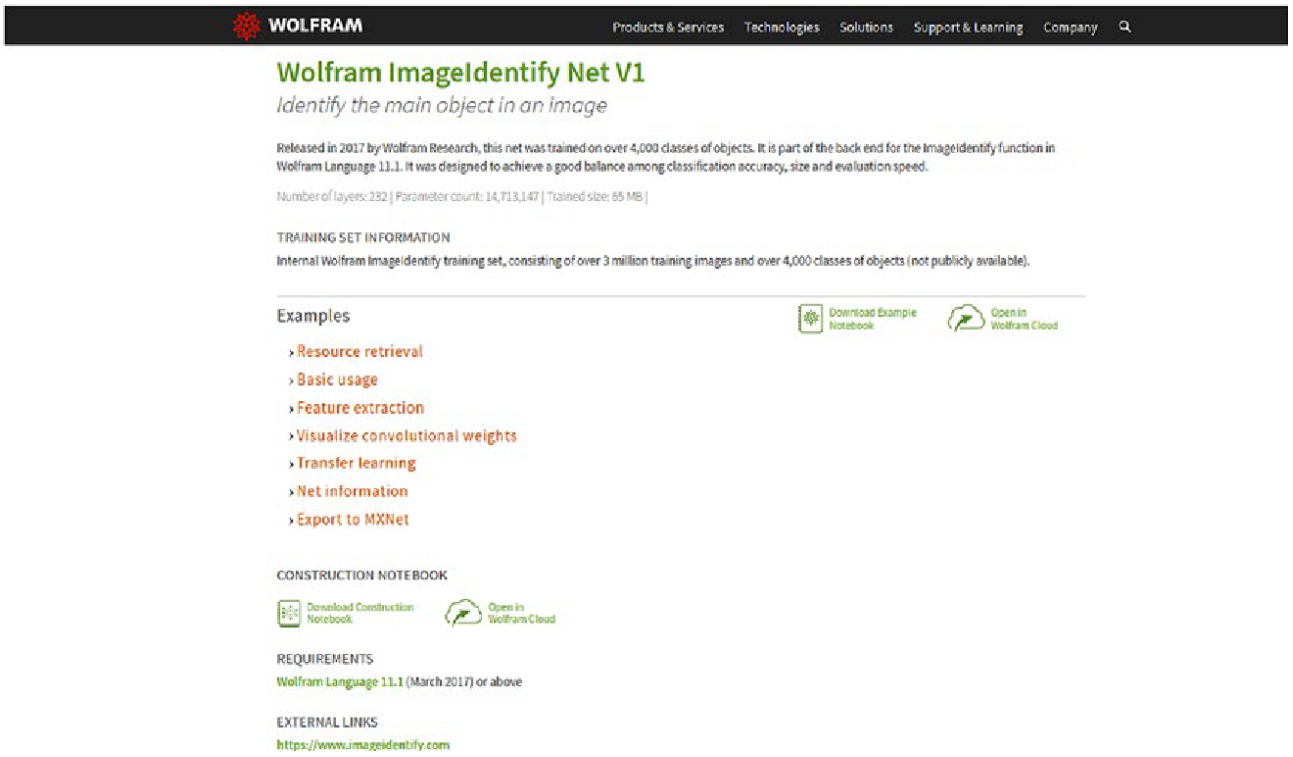


Рис. 3.15. Wolfram ImageIdentify Net V1

Існує можливість переходу з веб-сайту та завантаження записника з мережевою моделлю, але це також можливо з системи Mathematica. Іншими словами, шукайте мережеві моделі через ResourceSearch. У прикладі показаний пошук, якщо нам було цікаво дізнатися про моделі мереж, що містять слово image.

```
In[21]:= ResourceSearch[{"Name"-> "Image","ResourceType"->"NeuralNet"}]//
Dataset[#,MaxItems->{4,3}]&
```

Out[21]=

Name	ResourceType	ResourceObject
Colorful Image Colorization Trained on ImageNet Competition Data	NeuralNet	ResourceObject["Colorful Image Colorization Trained on ImageNet Competition Data"]
ColorNet Image Colorization Trained on ImageNet Competition Data	NeuralNet	ResourceObject["ColorNet Image Colorization Trained on ImageNet Competition Data"]
Wolfram ImageIdentify Net V1	NeuralNet	ResourceObject["Wolfram ImageIdentify Net V1"]
Ademxapp Model A Trained on ImageNet Competition Data	NeuralNet	ResourceObject["Ademxapp Model A Trained on ImageNet Competition Data"]

rows 1–4 of 24 columns 1–3 of 6

Рис. 3.16. Набір даних ресурсу

Набір даних, показаний на зображенні Рис. 3.16 має лише три стовпці для зручності відображення, але за допомогою повзунка ви можете переміщатися по всьому набору даних. На зображенні не показані стовпці: "Опис", "Розташування" та "Посилання на документацію". Останній стовпець містить посилання, яке веде до сторінки веб-моделі.

3.4.5 Доступ зсередини середовища Mathematica

Доступ до архітектури моделі додає аргумент об'єкта. Наприклад, для мережі Wolfram ImageIdentify Net V1 (Рис. 17) виконати такі дії.

```
In[22]:= ResourceSearch[{"Name"-> "Wolfram ImageIdentify","ResourceType"->
"NeuralNet"},"Object"]
Out[22]=
```

Рис. 3.17. Ресурс Wolfram ImageIdentify Net V1

Доступ до попередньо навченої моделі. Наступний код тут пригнічений, але видалення точки з комою повертає об'єкт NetChain попередньо навченої нейронної мережі.

```
In[23]:=ResourceSearch[{"Name"-> "Wolfram ImageIdentify","ResourceType"->
"NeuralNet"},"Object"][[1]]//ResourceData;
```

3.4.6 Отримання відповідної інформації

Варто зазначити, що доступ до інформації про модель здійснюється з ResourceObject. Нижче наведено відповідну інформацію з моделі ImageIdentify у вигляді набору даних (рис. 3.18). Щоб побачити всю інформацію у форматі набору даних, введемо:

```
ResourceObject ["Wolfram ImageIdentify Net V1"][All]//Dataset [#] &.
In[24]:=Dataset[AssociationMap[ResourceObject["Wolfram ImageIdentify Net
V1"][#]&,Map[ToString,{Name,RepositoryLocation,ResourceType,ContentElement,
Version,Description,TrainingSetData,TrainingSetInformation,InputDomains,
TaskType,Keywords,Attributes,LatestUpdate,DownloadedVersion,Format,
ContributorInformation,DOI,Originator,ReleaseDate,ShortName,Wolfram
LanguageVersionRequired},1]]]
Out[24]=
```

Name	Wolfram ImageIdentify Net V1
RepositoryLocation	https://www.wolframcloud.com/objects/resourcesyste...
ResourceType	NeuralNet
ContentElements	{ConstructionNotebook, EvaluationNet, UninitializedEvaluationNet, ConstructionNotebookExpression, EvaluationExample}
Version	1.10.0
Description	Identify the main object in an image
TrainingSetData	—
TrainingSetInformation	Internal Wolfram ImageIdentify training set, consisting of over 3 million training images and over 4,000 classes of objects (not publicly available).
InputDomains	Image
TaskType	Classification
Keywords	{ImageIdentify, object classification}
Attributes	{LocalCopyable, CloudCopyable, Multipart}
LatestUpdate	Fri 28 Feb 2020 01:00:00
DownloadedVersion	1.10.0
Format	<|EvaluationNet → WLNet, UninitializedEvaluationNet → WLNet, ConstructionNotebookExpression → NB, EvaluationExample → WXF|>
ContributorInformation	<|PublisherID → Wolfram, DisplayName → Wolfram Research|>
DOI	https://doi.org/10.24097/wolfram.34204.data
Originator	Wolfram Research
ReleaseDate	Mon 20 Feb 2017 17:00:00
ShortName	Wolfram-ImageIdentify-Net-V1

Рис. 3.18. Набір даних деяких властивостей Wolfram ImageIdentify Net V1

Тут, за кілька кроків, є спосіб доступу до навченої нейронної мережі на додаток до великої кількості важливої інформації, пов'язаної з нейронною мережею. Слід зазначити, що цей процес також використовується для пошуку інших ресурсів, що знаходяться у хмарі Wolfram Cloud або локальних ресурсів, а не тільки в нейронних мережах, оскільки в цілому ResourceSearch шукає об'єкт, який знаходиться у системі ресурсів Wolfram. Така справа з моделями нейронних мереж, які знаходяться в репозиторії нейронних мереж Wolfram.

3.5 Нейронна мережа LeNet

Тепер у наступному прикладі ми реалізуємо модель нейронної мережі з ім'ям LeNet. Незважаючи на можливість доступу до моделі ресурсу Wolfram, як ми бачили раніше, можна виконувати операції з мережами, знайденими в репозиторії нейронних мереж Wolfram, за допомогою команди NetModel. Щоб краще

зрозуміти, як використовується ця мережа, давайте спочатку подивимося на опис мережі, її назву, як вона використовується і де вона була запропонована вперше.
За допомогою NetModel ми можемо отримати інформацію про мережу LeNet, яка була попередньо навчена.

```
In[25]:= NetModel["LeNet Trained on MNIST Data",#]&/@{"Details","ShortName"
,"TaskType","SourceMetadata"}//Column
Out[25]=
```

Ця піонерська робота з класифікації зображень за допомогою згорткових нейронних мереж була випущена в 1998 році. Вона була розроблена Яном ЛеКуном та його співробітниками з AT&T Labs, коли вони експериментували з широким спектром рішень машинного навчання для класифікації набору даних MNIST. LeNet-Trained-on-MNIST-Data

```
{Classification}
<|Citation->Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, "Gradient-Based
Learning Applied to Document Recognition," Proceedings of the IEEE,
86(11), 2278-2324 (1998),Source->http://yann.lecun.com/exdb/lenet,Date-
>DateObject[{1998},Year,Gregorian,-5.]|>
In[26]:= NetModel["LeNet Trained on MNIST Data",#]&/@{"TrainingSetInformati
on","InputDomains","Performance"}//Column
Out[26]= MNIST Database of Handwritten Digits, consisting of 60,000
training and 10,000 test grayscale images of size 28x28.
{Image}
```

Ця модель сягає 98,5% точності набору даних MNIST[19].

3.6 MNIST Dataset.

Ця мережа використовується для оцінки, як і TaskType. Цифри знаходяться у базі даних, відомої як база даних MNIST. База даних MNIST - це велика база

даних рукописних цифр (Рис. 19), що містить 60 000 зображень для навчання та 10 000 зображень для тестування, останнє використовується для отримання остаточної оцінки того, наскільки добре працює модель нейронної мережі[20]. Щоб переглянути повний набір даних, завантажуємо його з репозиторію даних Wolfram за допомогою ResourceData і за допомогою ImageDimensions перевіряємо, що розміри зображень становлять 28 x 28 пікселів.

```
In[27]:= (*This is for seven elements randomly sampled, but you can check
the whole data set.*)
TableForm[
SeedRandom[900];
RandomSample[ResourceData["MNIST","TrainingData"],7],TableDirections->Row]
Map[ImageDimensions,%[[1;;7,1]]](*Test set : ResourceData["MNIST","TestData"]*)
Out[27]//TableForm=
Out[28]=
{{28,28},{28,28},{28,28},{28,28},{28,28},{28,28},{28,28}}
```

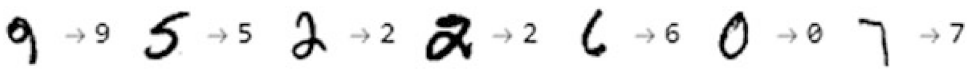


Рис. 3.19. Випадкова вибірка навчального набору MNIST

На Рис. 19 показано зображення чисел і класу, до якого вони належать, а також розміри кожного зображення. Далі ми витягуємо набори, навчальні набори та набори тестів, які будемо використовувати пізніше.

```
In[29]:={TrainData,TestData}={ResourceData["MNIST","TrainingData"], Resource
Data["MNIST","TestData"] };
```

3.7 Розробка архітектури фреймворку нейронної мережі.

Ми розпочнемо із завантаження нейронної мережі з команди NetModel, яка витягує модель з репозиторію нейронних мереж Wolfram. У наступній вправі ми завантажимо мережу, яка не була навчена, оскільки ми виконуватимемо процес навчання та перевірки. Слід зазначити, що модель LeNet мовою Wolfram Language є різновидом вихідної архітектури (Рис. 20).

```
In[30]:= UninitLeNet=NetModel["LeNet Trained on MNIST Data","Uni
nitializedEvaluationNet"](*To work locally with the untrained model:
NetModel["LeNet"]*)
Out[30]=
```

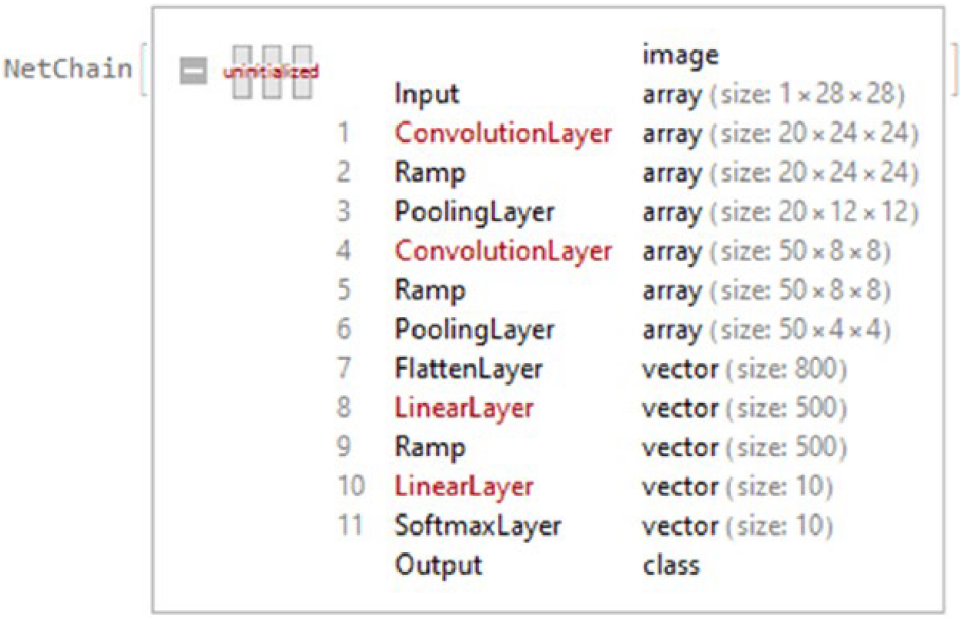


Рис. 3.20. Архітектура нейронної мережі LeNet

Ми бачимо, що мережа LeNet у репозиторії нейронних мереж Wolfram побудована з 11 рівнів. Шари, виділені червоним кольором, є шарами з навчальними параметрами: два згорткових шари і два лінійних шари.

3.7.1 MXNet Framework

Використовуючи фреймворк MXNet, спочатку візуалізуємо процес цієї мережі через графік роботи MXNet.

In[31]:=Information[UninitLeNet,"MXNetNodeGraphPlot"]

Ouy[31]=

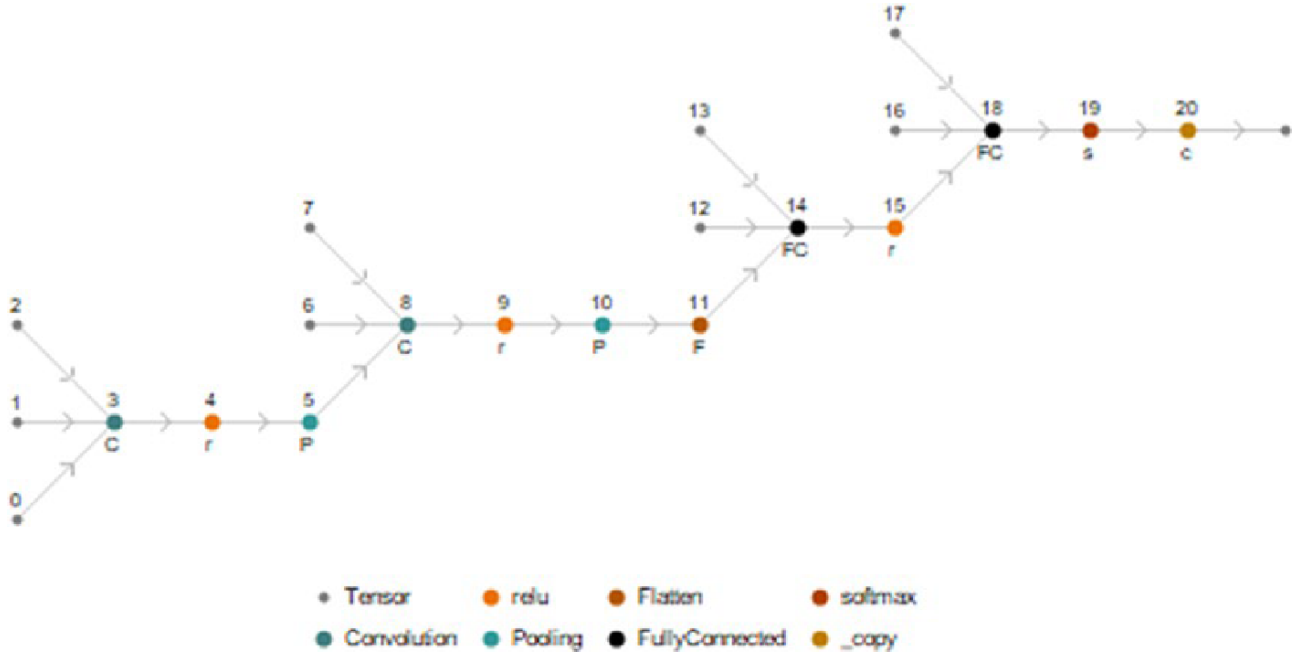


Рис. 3.21. Граф MXNet

Архітектура LeNet починається на вході з кодувальника, який перетворює зображення в числовий масив, за яким слідує перша операція, яка є згорткою, яка повертає карту з 20 ознак, з випрямленою функцією активації лінійного блоку у вузлах 3 і 4. Потім перша операція максимального об'єднання (шари підвибірки), яка вибирає максимальне значення у вузлі об'єднання 5. Потім йде друга згорткова операція, яка повертає карту з 50 об'єктами також з випрямленою функцією активації лінійного блоку у вузлах 8 і 9. За останньою операцією згортки слідує

ще одна операція максимального об'єднання (вузол 10), за якою слідує операція вирівнювання (вузол 11), яка об'єднує вихідні дані операції об'єднання в один вектор[21,22]. Остання операція об'єднання дає масив розміром 50*4*4, а операція згладжування повертає 800-вектор, який є входом наступної операції. Потім бачимо перший повністю пов'язаний шар (лінійний шар; вузол 14), перший повністю пов'язаний шар має випрямлену лінійну одиничну функцію (вузол 15), а другий повністю пов'язаний шар має функцію softmax (шар softmax; вузол 19). Останній повністю пов'язаний шар можна інтерпретувати як багатошаровий перцептрон (MLP), який використовує softmax для нормалізації вихідних даних у розподіл ймовірностей, щоб визначити ймовірність кожного класу. Нарешті, тензор перетворюється на клас із декодером. Вузли 4, 9, 15 та 19 є рівнями для нелінійних операцій.

3.8. Підготовка LeNet

Оскільки LeNet працює як нейронна мережа для класифікації зображень, необхідно використовувати кодувальник та декодер. NetEncoder вставлений у вхідний NetPort, а NetDecoder - у вихідний NetPort. Вивчення NetGraph (Рис. 22) може бути корисним для розуміння процесу всередині мови Wolfram Language. При натисканні на вхід та вихід відображається відповідна інформація.

```
In[32]:= NetGraph[UninitLeNet]
Out[32]=
```

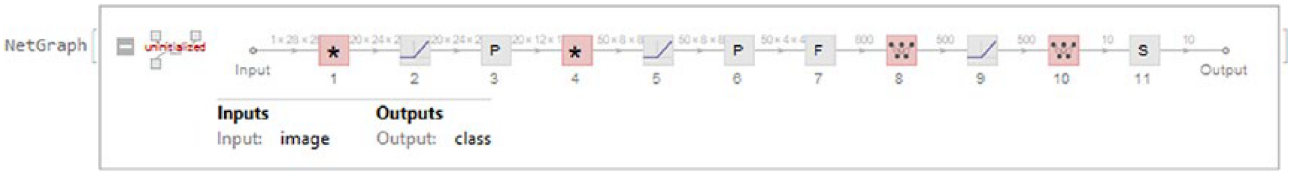


Рис. 3.22. NetGraph моделі LeNet

Ми можемо отримати кодер та декодер, щоб перевірити їх інфраструктуру. Кодер отримує зображення розміром 28 на 28 будь-якого колірного простору і кодує зображення в колірний простір, встановлений у відтінки сірого, потім повертаючи масив розміром 1 на 28 на 28. З іншого боку, декодер є декодер класу, який отримує вектор розміром 10, який повідомляє ймовірність для міток класів, які дорівнюють 0, 1, 2, 3, 4, 5, 6, 7, 8 та 9.

```
In[33]:={Enc=NetExtract[UninitLeNet,"Input"],Dec=NetExtract[UninitLeNet,
"Output"]}//Row;
```

Давайте спочатку подивимося, як мережева модель працює з NetInitialize. Як приклад ми використовуємо зображення числа 0 у навчальній вибірці.

```
In[34]:= TestNet=NetInitialize[UninitLeNet, RandomSeeding->8888];
TestNet@TrainData[[1,1]](*TrainData[[1,1]] belongs to a zero*)
Out[34]= 9
```

Мережа повертає інформацію, що зображення належить до класу 9, що означає, що зображення має номер 9; явно, це неправильно. Спробуймо NetInitialize ще раз, але з іншими доступними методами. Запис all як другий аргумент NetInitialize перезаписує будь-які попередньо існуючі параметри навчання в мережі.

```
In[35]:= {net1,net2,net3,net4}=Table[NetInitialize[UninitLeNet, All, Method-
>i,RandomSeeding->8888],{i,{"Kaiming","Xavier","Orthogonal","Identity"}}];
{net1[TrainData[[1,1]]],net2[TrainData[[1,1]]],net3[TrainData[[1,1]]],net4[
TrainData[[1,1]]]}
Out[35]= {9,9,5,3}
```

Кожна нейронна мережа не може віднести зображення до правильного класу. Це пов'язано з тим, що нейронна мережа не була навчена, на відміну від NetInitialize, яка лише випадковим чином ініціалізує параметри, що навчаються, але без виконання належного навчання. Ось чому NetInitialize не може правильно класифікувати це зображення. Але спочатку ми збираємося створити мережевий граф, щоб краще проілюструвати ідею (Рис. 23).

```
In[36]:= LeNet=NetInitialize[NetGraph[<|"LeNet NN"->UninitLeNet,"LeNet
Loss"->CrossEntropyLossLayer@"Index"|>,{NetPort@"Input"->"LeNet NN","LeNet
NN"->NetPort@{"LeNet Loss","Input"},NetPort@"Target"->NetPort@{"LeNet Loss"
,"Target"}}],RandomSeeding->8888]
Out[36]=
```

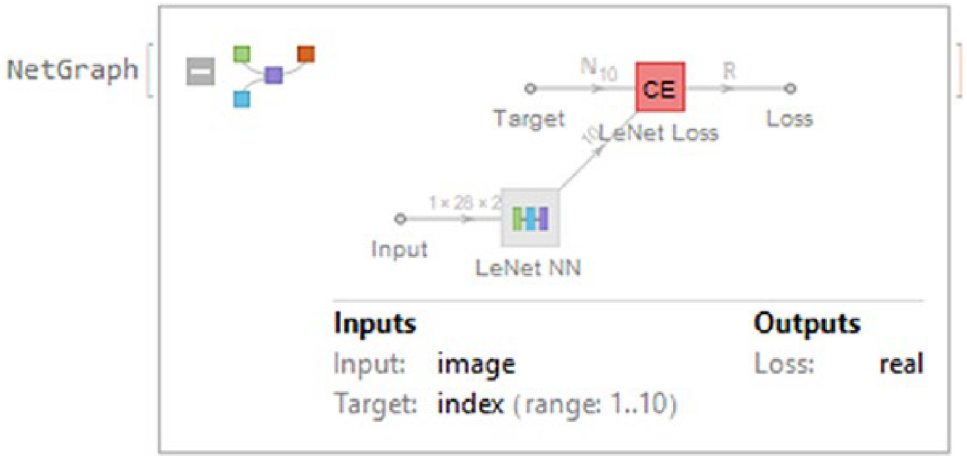


Рис. 3.23. LeNet готова до навчання

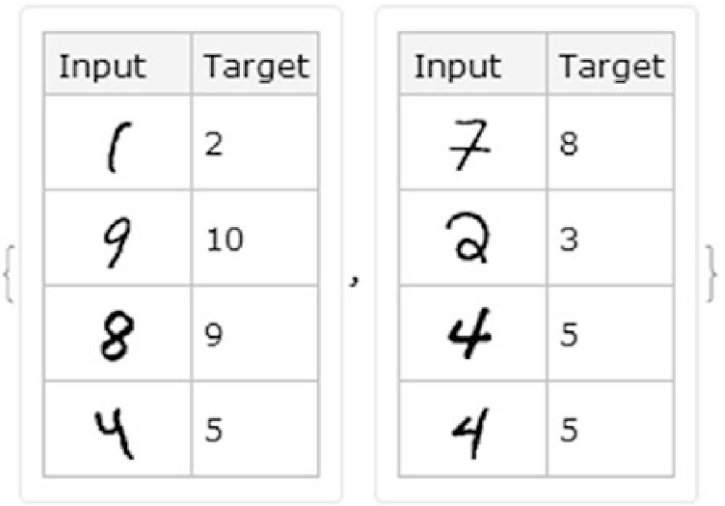
Перш ніж ми приступимо до навчання мережі, нам потрібно зробити набір перевірки відповідним для CrossEntropyLossLayer у цільовому введенні, тому що класи починаються з 0 і закінчуються на 9, а мета індексу починається з 1 і продовжується. Отже, цільове введення має бути від 1 до 10.

```
In[37]:=
]TrainDts=Dataset@Join[AssociationThread["Input"->#]& /@Keys[TrainData],
AssociationThread["Target"-> #]&/@Values[TrainData]+1,2];
TestDts=Dataset@Join[AssociationThread["Input"->#]& /@Keys[TestData],
```

```
AssociationThread["Target"-> #]&/@Values[TestData]+1,2];
```

Навчальний набір та набір перевірки мають форму набору даних. На рис. 24 показані лише чотири випадкові вибірки.

```
In[38]:=BlockRandom[SeedRandom[999];
{RandomSample[TrainDts[[All]],4],RandomSample[TestDts[[All]],4]}]
Out[38]=
```



Input	Target
	2
	10
	9
	5

Input	Target
	8
	3
	5
	5

Рис. 3.24. Набір даних навчального та тестового набору

3.9 Навчання мережі LeNet

Тепер, коли ми зрозуміли процес цієї моделі нейронної мережі, ми можемо розпочати навчання моделі нейронної мережі. За допомогою NetTrain ми поступово модифікуємо навчальні параметри нейронної мережі, щоб зменшити втрати[23,24]. Наступний навчальний код визначається параметрами, показаними в попередньому розділі, але тут ми додаємо нові параметри, які також доступні для навчання. Перший - TrainingProgressMeasurements. З допомогою TrainingProgressMeasurements ми можемо вказати, що такі заходи, як точність, прецизійність тощо. буд., вимірюються на етапі навчання або циклічно, або

пакетно. ClassAveraging використовується для вказівки отримання макро-середнього чи мікро-середнього значення зазначеного виміру <| "Measurement" -> "measurement" (Accuracy, RSquared, Recall, MeanSquared і т. д.), "ClassAveraging" -> "Macro" |>.

Другий варіант - це TrainingStoppingCriterion, який використовується для додавання ранньої зупинки, щоб уникнути перенавчання під час фази навчання, на основі різних критеріїв, таких як зупинка навчання, коли втрата перевірки не покращується, вимір абсолютної або відносної зміни вимірювання (точність, точність, втрата та т. д.), або зупинка навчання, коли втрата чи інші критерії не покращуються після певної кількості раундів <|Criterion->“measurement” (Accuracy, Loss, Recall, etc.),“Patience”-> # of rounds|>.

```
In[39]:= NetResults=NetTrain[LeNet,TrainDts,All,ValidationS
et->TestDts,MaxTrainingRounds->15,BatchSize->2096,LearningRate-
>Automatic,Method->"ADAM",TargetDevice->"CPU",PerformanceGoal-
>"TrainingMemory",WorkingPrecision->"Real32",RandomSeeding->99999,Train
ingProgressMeasurements->{<|"Measurement"->"Accuracy","ClassAveraging"-
>"Macro"|>, <|"Measurement"->"Precision","ClassAveraging"-
>"Macro"|>,<|"Measurement"->"F1Score","ClassAveraging"-
>"Macro"|>,<|"Measurement"->"Recall","ClassAveraging"-
>"Macro"|>,<|"Measurement"->"ROCCurvePlot","ClassAveraging"-
>"Macro"|>,<|"Measurement"->"ConfusionMatrixPlot","ClassAveragi
ng"->"Macro"|> },TrainingStoppingCriterion-> <|"Criterion"->"Loss","Absolut
eChange"->0.001|>]
Out[39]=
```

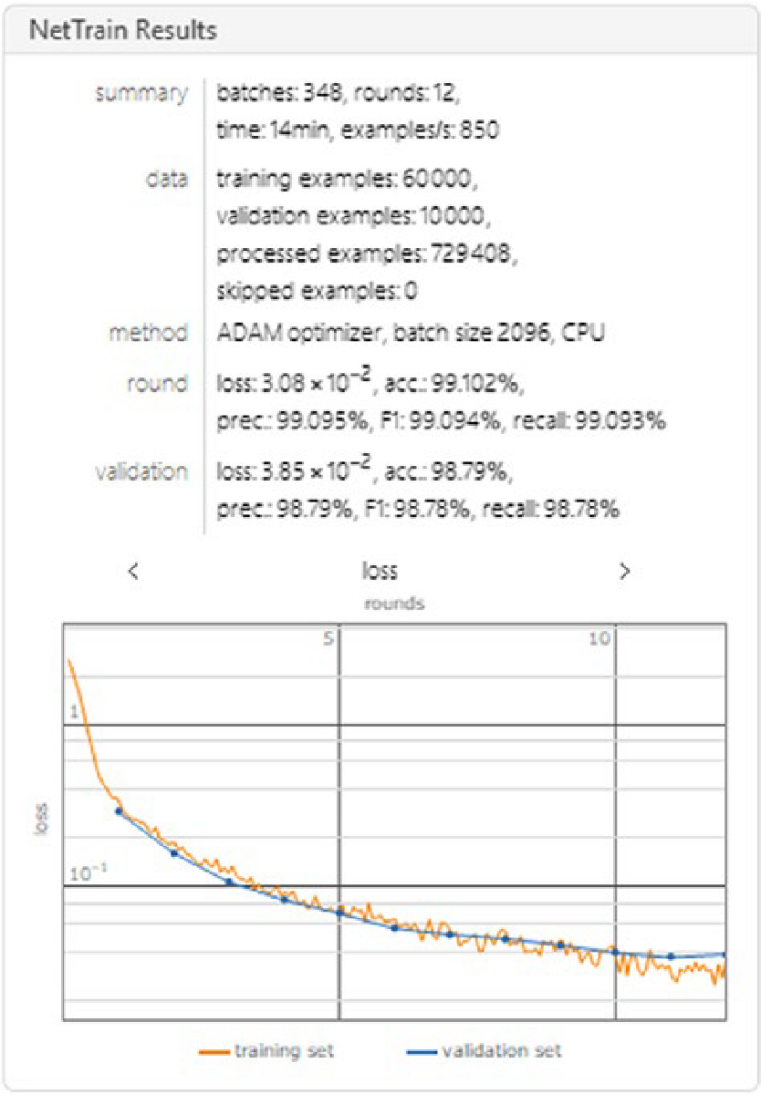


Рис. 3.25. Чисті результати навчання мережі LeNet

Остаточні результати етапу навчання показано на Рис. 25. Виймання навченої моделі та додавання мережного кодера та декодера виконується, тому що навчена мережа не має кодувальника та декодера на вхідних та вихідних портах.

```
In[40]:=NetExtract[NetResults["TrainedNet"],"LeNet NN"];
TrainedLeNet=NetReplacePart[%,{"Input"->Enc,"Output"->Dec}];
```

3.10 Оцінка моделі LeNet

Наступна сітка (Рис. 3.26) показує відстежувані вимірювання та графіки навчальної вибірки. Вимірювання навчального набору перебувають у властивості RoundMeasurements. Щоб отримати список значень у кожному раунді, використовуйте RoundMeasurementsLists. Продуктивність навчального набору оцінюється за допомогою круглих вимірів, а тестовий набір оцінюється за допомогою перевірочних вимірів[25]. Крім того, в обох випадках показані криві ROC та графік матриці неточностей.

```
In[41]:=NetResults["RoundMeasurements"][[1;;5]];
Normal[NetResults["RoundMeasurements"][[6;;7]]];
Grid[{{Style["RoundMeasurements",#1,#2],Style[%[[1,1]],#1,#2],Style[%[[2,1]],
#1,#2]},{Dataset[%%],%[[1,2]],%[[2,2]]}},Dividers->Center]&[Bold,FontFamily
->"Alegreya SC"]
Out[41]=
```

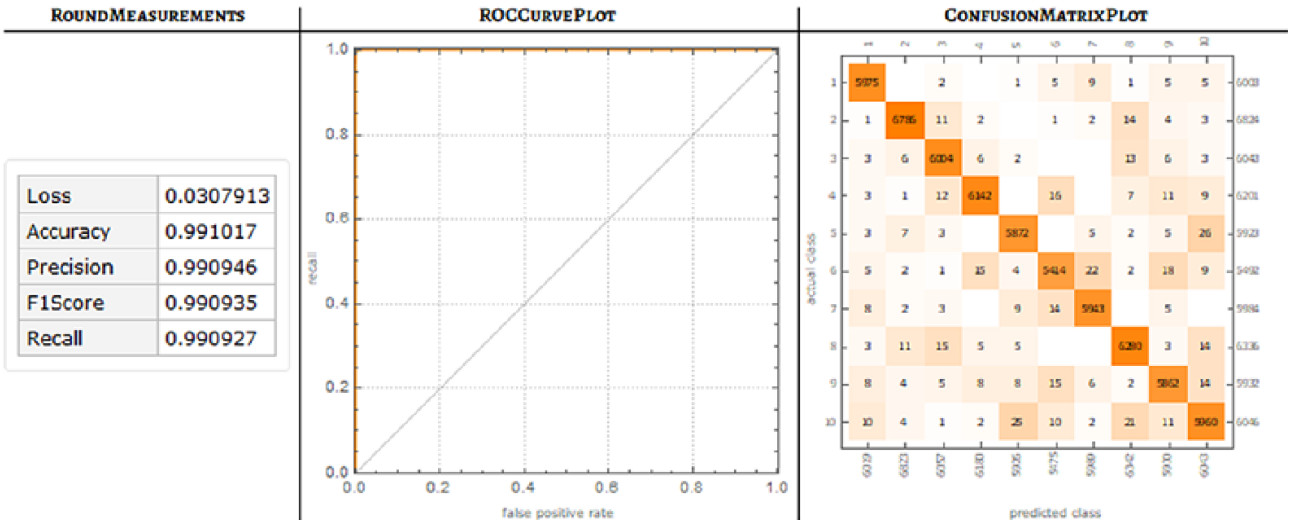


Рис. 3.26. Вимірювання тренувального набору

Щоб побачити, як модель працює на перевірочному наборі (Рис. 27), див. ValidationMeasurements. Щоб отримати список значень у кожному раунді, використовуємо ValidationMeasurementsLists.

In[42]:=NetResults["ValidationMeasurements"][[1;;5]];
Normal[NetResults["ValidationMeasurements"][[6;;7]]];
Grid[{{Style["ValidationMeasurements",#1,#2],Style[%[[1,1]],#1,#2],Style[%[
[2,1]],#1,#2]},{Dataset[%%],%[[1,2]],%[[2,2]]}}, Dividers->Center]&[Bold,FontFamily->"
Alegreya SC"]
Out[42]=

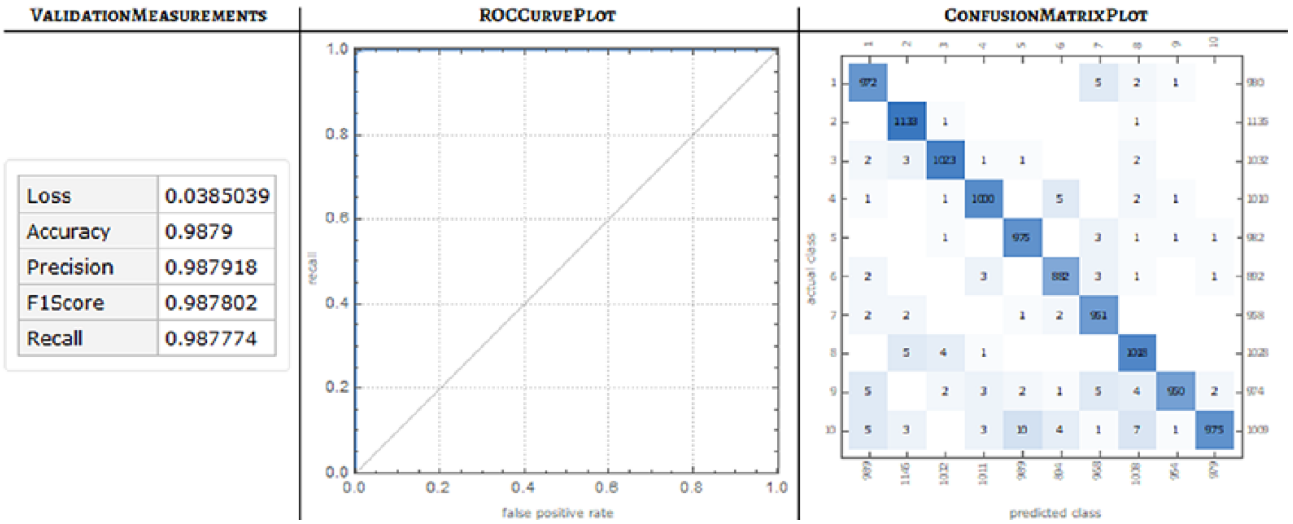


Рис. 3.27. Вимірювання тестового набору

3.11 Тестування нейронної мережі LeNet

Завершивши навчання та розглянувши сумарні засоби та засоби перевірки, ми тепер можемо протестувати навчену нейронну мережу LeNet із деякими складними зображеннями, щоб побачити, як вона працює.

In[43]:=

```
Expls=Keys[{TestData[[2150]],TestData[[3910]],TestData[[6115]],TestData[[60
11]], TestData[[7834]]}]
Out[43]=
```

Рис. 3.28. Складні приклади набору тестів MNIST

Вибрані зображення відносяться до номерів 2, 3, 6, 5 та 7.

```
In[44]:= TrainedLeNet[Expls,"TopProbabilities"]
Out[44]= {{2->0.998171},{3->0.999922},{6->0.587542,0->0.404588},
{6->0.971103},{7->0.99937}}
```

Тепер подамо усі результати з максимальною ймовірністю за допомогою TableForm.

```
In[45]:= TableForm[Transpose@{TrainedLeNet[
Expls,{"TopDecisions",2}],TrainedLeNet[
Expls,{"TopProbabilities",2}]},TableHeadings-
>{Map[ToString,{2,3,6,5,7},1],{"Top Decisions","Top Probabilities"}},
TableAlignments->Center]
Out[45]//TableForm=
```

	Top Descisions	Top Probabilities
2	3	3 -> 0.00165948
	2	2 -> 0.998171
3	9	9 -> 0.0000534744
	3	3 -> 0.999922
6	0	0 -> 0.404588
	6	6 -> 0.587542
5	0	0 -> 0.0140468
	6	6 -> 0.971103
7	3	3 -> 0.000526736
	7	7 -> 0.99937

Табл. 2. Результати тестування

Ми можемо бачити, що навчена мережа неправильно класифікувала зображення числа 5, тому що головними рішеннями є 0 або 6, і очевидно, що це неправильно. Також ми можемо бачити ймовірність прийняття найкращих рішень. Інша форма для оцінки навченої мережі в тестовому наборі - використання NetMeasurements для встановлення мережевої моделі, тестового набору та заходи, що цікавлять. У цьому прикладі цікавою мірою є ConfusionMatrixPlot.

In[26]:=NetMeasurements[TrainedLeNet,TestData,"ConfusionMatrixPlot"]

Out[26]=

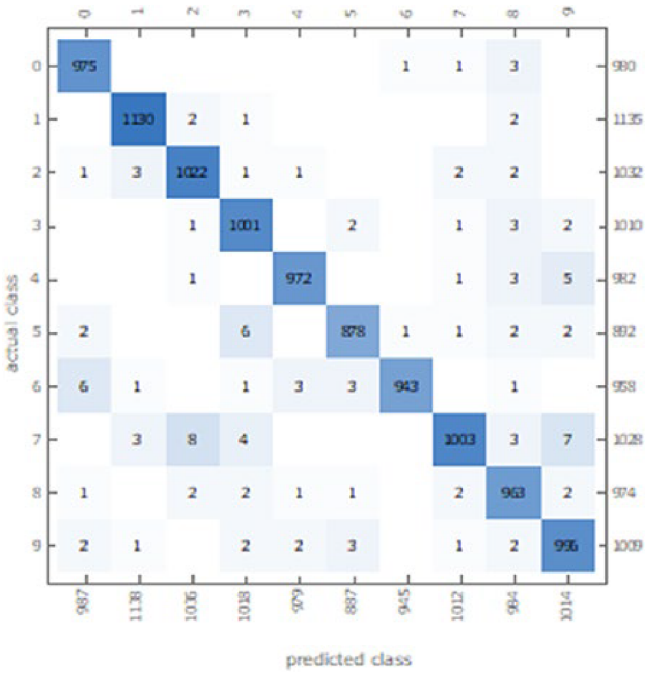


Рис. 3.29. ConfusionMatrixPlot від набору NetMeasurements

Підсумовуючи, можна дійти висновку, що у загальних рисах підходи до побудови, тестування й реалізації машинного навчання чи реалізації нейронної мережі у межах системи Wolfram Language є також як показано нижче.

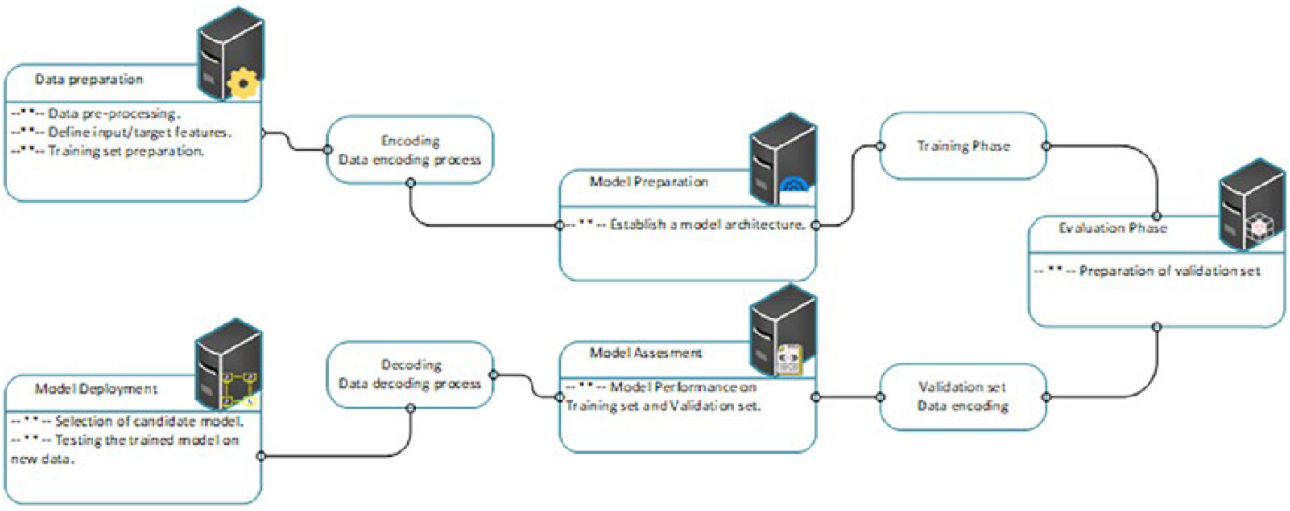


Рис. 3.30. Вимірювання тренувального набору

4. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Охорона праці

При написанні кваліфікаційної роботи та розробці методів дослідження нейронних мереж використовувався персональний комп’ютер, тому слід зазначити, що при роботі за комп’ютерною технікою необхідно дотримуватися вимог охорони праці з ціллю збереження здоров'я.

При роботі з комп’ютером значним чином відбувається вплив на нервово-емоційний стан операторів, така робота характеризується великим навантаженням на м’язи рук при роботі з клавіатурою комп'ютера, високою інтенсивністю зорової роботи та значним розумовим перенапруженням.

Отже, раціональне планування робочого місця повинно забезпечити: зменшення втоми працівників та підвищення продуктивності праці, уникнення загального дискомфорту, якнайкраще розташування інструментів та предметів праці.

Для того, щоб виявити та проаналізувати шкідливі і небезпечні виробничі фактори необхідно почати з аналізу дотримання вимог, встановлених санітарними правилами і нормами [ДСанПіН 3.3.2.007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин»] для виробничих приміщень та робочих місць.

На виробництві для дійсної оцінки умов праці відбувається сертифікація робочих місць відповідно до умов праці та використовується «Гігієнічна класифікація праці» за показниками небезпечності та шкідливості факторів виробничого середовища, напруженості та тяжкості трудового процесу. На основі

принципів гігієнічної класифікації умови праці під час роботи з системою досліджень нейронних мереж належать до першого класу. Згідно якого створюються оптимальні умови праці, для підтримки високого рівня працездатності.

На підставі сертифікації робочого місця необхідно охарактеризувати інтенсивність робіт за такими напрямами:

- відповідність обладнання нормативно-технічним вимогам;
- документації, а також характер і обсяг виконуваних робіт;
- відповідність площі та обсягу займаного робочого місця чинним нормам;
- спеціалізоване устаткування робочого місця (засоби захисту пристроїв та їх технічний стан);
- відповідність технологічного процесу, інструментів, устаткування, засобів контролю вимогам стандартів безпеки і нормам охорони праці.

Обладнання та організація робочого місця ВДТ ЕОМ повинні задовольняти відповідність конструкції всіх елементів робочого місця та їх відносного розташування ергономічним вимогам з урахуванням особливостей та характеру трудової діяльності [ДСТУ 7299:2013 «Дизайн і ергономіка. Робоче місце оператора. Взаємне розташування елементів робочого місця. Загальні вимоги ергономіки»].

Раціональне планування робочого місця повинно забезпечити: найкраще розміщення інструментів та предметів праці, уникати загального дискомфорту, зменшити втому працівників та підвищити продуктивність праці.

Заходи щодо усунення ризику ураження електричним струмом зводяться до правильного розміщення обладнання та електричних кабелів. Також інші заходи для забезпечення електробезпеки збігаються із загальними заходами пожежної та електробезпеки.

В якості заходів профілактики для того, щоб забезпечити пожежну безпеку необхідно у приміщеннях використовувати приховану електромережу, ввімкнення та вимкнення живлення виконувати обладнанням за допомогою стандартних вимикачів, надійні розетки з важкозаймистих матеріалів. Необхідно регулярно чистити внутрішні частини комп'ютерів та інше обладнання від пилу, комп'ютери розміщувати на окремих вогнестійких столах. Для запобігання іскроутворення необхідно рідше вставляти і виймати вилки з розеток. Щодо розташування робочих місць, то вони мають бути розміщені на відстані, що не є меншою 1,5 м від стіни з вікнами, та від інших стін на відстані близько 1 м, між бічними поверхнями ВДТ - 1,2 м; від задньої площини одного ВДТ до іншого екрану - 2,5 м.

Конструкція робочої поверхні користувача ВДТ повинна забезпечувати підтримку оптимального робочого положення. Сприятливе робоче положення в процесі роботи за комп’ютером забезпечується налаштуванням висоти робочого столу, підставки для ніг і стільця.

Важливою є форма спинки стільця, яка повинна повторювати форму спини працівника. Висота стільця має бути такою, щоб користувач не відчував тиску на стегна чи куприк. Бажано обладнати крісло підлокітниками. Потрібно встановити їх так, щоб не довелося тягнутися до клавіатури.

Порівняно з вікнами робоче місце доцільно розмістити таким чином, щоб на нього природне світло потрапляло з бокової частини, переважно зліва. Робочі зони потрібно розташовувати так, щоб пряме світло не потрапляло в очі. Рекомендовано розміщувати джерела світла по обидва боки екрану, паралельно напрямку погляду. Для того, щоб уникнути відблисків на екрані, клавіатурі в напрямку очей користувача, від загальних освітлювальних приладів або сонячного світла, потрібно використовувати спеціалізовані фільтри для екранів, захисні навіси, антипроблискові сітки та жалюзі на вікнах.

Екран ВДТ повинен бути розташований на оптимальній відстані від очей користувача, яка становить від 600 до 700 мм, але не ближче 600 мм, враховуючи розмір буквено-цифрових символів та знаків.

При обладнанні робочого місця ВДТ лазерним принтером параметри лазерного випромінювання повинні відповідати вимогам [ДСанПІН 3.3.2.007-98].

Дотримання всіх необхідних заходів з охорони праці забезпечує комфортні умови праці та відсутність шкоди для здоров’я, що сприяє підвищенню продуктивності праці, а також меншому виснаженню при роботі за персональним комп’ютером.

4.2 Фактори що впливають на функціональний стан користувачів комп'ютерів

Трудова діяльність користувачів комп'ютерів (ВДТ) відбувається у певному виробничому середовищі, яке впливає на їх функціональний стан. Психофізіологічні та емоційні перенапруження, втома можуть призвести у комп'ютеризованих системах керування до помилок і як наслідок – до значних економічних втрат. Визначення та вивчення факторів, що впливають на функціональний стан користувачів комп'ютерів дозволить виділити основні причини виникнення станів напруженості, стомлення, стресу і здійснити відповідні профілактичні заходи.

До основних факторів, що впливають на функціональний стан користувачів комп’ютера належать:

1) виробниче середовище – характеризується такими шкідливими факторами:

1.1 фізичні: електромагнітні хвилі різних частотних діапазонів, електростатичні поля, шум, параметри мікроклімату та ряд світлотехнічних показників;

1.2 хімічні: пил, шкідливі хімічні речовини, які виділяються при роботі принтера і копіювальної техніки;

1.3 біологічні: підвищений вміст в повітрі патогенних мікроорганізмів, особливо у приміщенні з великою кількістю працюючих, при недостатній вентиляції, особливо у період епідемій;

1.4 психофізіологічні: напруження зору та уваги, інтелектуальні та емоційні навантаження, тривалі статичні навантаження і монотонність праці.

2) трудовий процес - характеризується значними статичними фізичними навантаженнями; недостатньою руховою активністю; напруженнями сенсорного апарату, вищих нервових центрів, які забезпечують функції уваги, мислення, регуляції рухів. Окрім того, трудовий процес користувачів комп'ютерів відзначається значними інформаційними навантаженнями;

3) внутрішні засоби діяльності – це професійні риси та виробничий досвід, які обумовлюють надійну та безпомилкову діяльність користувачів комп'ютерів, дозволяють знаходити безпечні методи розв'язання виробничих завдань навіть у нестандартних ситуаціях;

4) зовнішні засоби діяльності - визначаються ергономічними показниками щодо організації робочого місця, форми та параметрів його елементів, просторового розташування основного і допоміжного устаткування, які можуть суттєво

знизити фізичні та психофізіологічні навантаження, що діють на користувачів комп'ютерів;

5) соціально-психологічні фактори трудових взаємовідносин.

На функціональний стан людини та на її здоров’я під час роботи з ПК впливає комплекс чинників, зокрема, зміст і обсяг інформації, інтенсивність і тривалість роботи за ПК, якість і досконалість використовуваних програмних продуктів, їхні ергономічні, педагогічні, психогігієнічні властивості. Окрім того, об’єктивними також вважають чинники які випливають з внутрішнього середовища приміщення, які виникають під час роботи комп’ютерів: показники мікроклімату, освітленість, яскравість, контрастність і колір зображення на екрані дисплея, іонізуюче та неіонізуюче опромінення, шум тощо.

У повітрі зовнішнього природного середовища, як і в повітрі приміщень, наявна певна кількість заряджених частинок, що називаються іонами. У приміщеннях, де працюють з комп'ютерами, концентрація легких негативних іонів зменшується (за 5 хвилин у вісім разів, а за 3 години – до нуля).

Така зміна складу повітря призводить до несприятливого впливу на здоров'я користувачів ПЕОМ, на їх розумову та фізичну діяльність.

Гранично допустимі норми рівнів іонізації повітря приміщень при роботі з ЕОМ повинні відповідати ДНАОП 0.03-3.06-80.

Засоби забезпечення необхідної концентрації іонізації повітря такі:

- кондиціювання повітря;

- генератори негативних іонів (іонізатори);

- збільшення вологості повітря у приміщеннях.

Накопичення електричного заряду на поверхні обладнання може досягати кількох тисяч вольт (переважно на електронно-променевій трубці відеотерміналу, зокрема, на екрані). При дотику до такого обладнання може статись електричний удар.

Для захисту від статичної електрики необхідно:

– кілька разів протягом робочого дня мити руки і обличчя водою, а після закінчення роботи вимити руки й лице з милом;

– щоденно протирати екран монітора, клавіатуру, пристрій “миша”, а якщо є приекранний фільтр то і його антистатичною серветкою;

– щоденно у приміщенні з ПК проводити вологе прибирання;

– установити нейтралізатори статичної електрики;

– підтримувати у приміщенні відносну вологість повітря не нижче 45-50 %, разом з тим недопустима вологість повітря більше 75%;

– виконати у відповідності з ДНАОП 0.00-1.21-98 “Правила безпечної експлуатації електроустановок споживачів” заземлення ВДТ;

– користувачу ПК бажано носити одяг з природних (льняних) волокон.

Напруга електростатичного поля на робочих місцях, у тому числі й відеоматеріалах, не повинна перевищувати 20 кВ/м відповідно до ГОСТ 12.1.045-84 "ССБТ. Электростатические поля. Допустимые уровни на рабочих местах и требования к проведенню контроля". Відповідно до цього нормативного документа робочі місця з ВДТ повинні:

- розміщуватися в окремих приміщеннях;

- площа одного робочого місця - не менше 6,0 м^2;

- об'єм приміщення - не менше 20,0 $м^3$;

- обладнуватись аптечками першої медичної допомоги;

- розміщуватись на відстані не менше 1 м від стін з вікнами;

- прохід між рядами робочих місць - не менше 1 м;

- бути обладнані системою автоматичної пожежної сигналізації з димовими оповіщувачами та вуглекислотними вогнегасниками з розрахунку два вогнегасника на кожні 20 $м^2$ площі приміщення та з урахуванням гранично допустимих концентрацій вогнегасної речовини;

- не рідше, як раз на квартал очищувати від пилу агрегати та вузли, кабельні канали та підлогу.

Під час роботи з дослідження нейронних мереж головними факторами що будуть впливати на роботу користувача будуть виробниче середовище та зовнішні засоби діяльності. Відтак правильна організація робочого місця водночас може позбавити одразу від двох цих факторів, а значить допоможе уникнути будь яких шкідливих наслідків для користувача.

ВИСНОВКИ

Розвинені схеми побудови нейронної мережі, яким можна слідувати безпосередньо; Незважаючи на це, всередині схеми можуть бути проміжні точки між кожним процесом, оскільки схема може змінюватись в залежності від того, який тип завдання або проблеми вирішується. Проте схема нейронної мережі фокусується на виявленні важливих та загальних моментів для побудови моделі з використанням мови Wolfram Language. На етапі підготовки даних є попередні процеси, такі як інтеграція даних, тип даних, що збираються (структуровані або неструктуровані), перетворення даних, очищення модулів даних і т. д. Тому, перш ніж перейти до наступного етапу, необхідно провести попередню обробку даних, щоб дані були готові до завантаження в модель.

Підготовка моделі охоплює такі аспекти, як вибір алгоритму або методів, що використовуються, залежно від типу навчання; встановлення чи виявлення структури моделі; та визначення характеристик, вхідних параметрів та типу даних, які будуть використовуватися, чи то текст, звук, числові дані та інструменти, які будуть використовуватися. Все це пов'язано з процесом, який називається функціональною інженерією, основною метою якого є вилучення цінних атрибутів з даних. Це необхідно, щоб перейти до наступного етапу – етапу навчання.

Етап оцінки та оцінка моделі складаються з визначення показників оцінки, які різняться в залежності від типу розв'язуваної задачі або проблеми, на додаток до підготовки валідації, яка буде використовуватись пізніше. На цьому етапі необхідно підкреслити, що підготовка моделі, навчання, оцінка та оцінка можуть бути ітеративним процесом, який може включати налаштування гіперпараметрів,

коригування методів алгоритмів та конфігурацій моделі, таких як внутрішні функції моделі. Ціль полягає в тому, щоб створити найкращу можливу модель, яка здатна забезпечити адекватні результати і, нарешті, досягти етапу розгортання моделі, який визначає модель, яка буде обрана і протестована на нових даних.

ПЕРЕЛІК ПОСИЛАНЬ

1. Zielesny, A. (2016). From Curve Fitting to Machine Learning. Intelligent Systems Reference Library. doi:10.1007/978-3-319-32545-3.
2. J. Barrat, Our Final Invention: Artificial Intelligence and the End of the Human Era, New York 2013, Thomas Dunne Books.
3. Freeman J. A. (1994): Simulating Neural Networks with Mathematica, Addison-Wesley, Massachusetts.
4. Haykin S. (2009): Neural Networks and Learning Machines,Third Edition,Prentice Hall,New York, London,Sydney,
5. J. A. Freeman, Simulating Neural Networks with Mathematica, Boston 1993, Addison-Wesley Longman Publishing Co.
6. A. Forsgren, P. E. Gill, M. H. Wright, Interior Methods for Nonlinear Optimization, SIAM Rev. 44 (4), 525-597, 2002.
7. Awange, J., Paláncz, B., & Völgyesi, L. (2020). Hybrid Imaging and Visualization. doi:10.1007/978-3-030-26153-5
8. S. S. Keerthi, E. G. Gilbert, Convergence of a Generalized SMO Algorithm for SVM Classifier Design, Machine Learning 46, 351-360, 2002.
9. Srinivasan J, Han YK and Ong SH (1993): Image reconstruction by a Hopfield neural network, Image and Vision Computing 11(5), pp. 279-282.
10. C. M. Bishop, Pattern Recognition and Machine Learning, New York 2006, Springer.
11. Rozhdestvensky, K., Ryzhov, V., Fedorova, T., Safronov, K., Tryaskin, N., Sulaiman, S. A., Hassan, S. (2020). Computer Modeling and Simulation of Dynamic Systems Using Wolfram SystemModeler. doi:10.1007/978-981-15-2803-3

12. R. J. Barlow, Statistics: A Guide to the Use of Statistical Methods in the Physical Sciences, Chichester 1989, Wiley VCH.
13. T. Glasmachers, C. Igel, Maximum-Gain Working Set Selection for SVMs, Journal of Machine Learning Research 7, 1437-1466, 2006.
14. Computational Intelligence Packages (CIP), Version 2.0. Open source library for Mathematica 10 or higher designed by Achim Zielesny.
15. P. Bevington, D. K. Robinson, Data Reduction and Error Analysis for the Physical Sciences, New York 2002, McGraw-Hill, 3rd Edition.
16. Banerjee T (2018): Deep Learning and Computer Vision: Converting Models for the Wolfram Neural Net Repository, Wolfram Blog.
17. Usama M et al (2017): Unsupervised Machine Learning for Networking:Techniques, Applications and Research Challenges. https://arxiv.org/pdf/1709.06599.pdf
18. Scitovski, R., Sabo, K., Martínez-Álvarez, F., & Ungar, Š. (2021). Cluster Analysis and Applications. doi:10.1007/978-3-030-74552-3.
19. S. Brandt, Data Analysis: Statistical and Computational Methods for Scientists and Engineers, New York 1998, Springer, 3rd Edition.
20. V. Cherkassy, D. Gehring, F. Mulier, Comparison of adaptive methods for function estimation from samples, IEEE Trans. Neural Networks 7 (4), 969-984, 1996.
21. N. Cristianini, J. Shawe-Taylor, An Introduction to Support Vector Machines and other kernel-based learning methods, Cambridge 2000, Cambridge University Press.
22. Haykin S. (2009): Neural Networks and Learning Machines,Third Edition,Prentice Hall,New York, London,Sydney,
23. G. A. Carpenter, S. Grossberg, D. B. Rosen, ART 2-A: An Adaptive Resonance Algorithm for Rapid Category Learning and Recognition, Neural Networks 4, 493-504, 1991.

24. S. Chatterjee, A. Hadi, B. Price, Regression Analysis by Example, New York 2000, John Wiley & Sons, 3rd Edition. Chapter 3: Multiple Linear Regression, pages 51-84.
25. K. P. Murphy, Machine Learning. A Probabilistic Perspective, Cambridge (Massachusetts) 2012, MIT Press.

ДОДАТКИ

ДОДАТОК А

ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

ІМЕНІ ІВАНА ПУЛЮЯ

КАФЕДРА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

Затверджую

Завідувач кафедри програмної інженерії

Петрик Михайло Романович

“___”_____ 2021 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на кваліфікаційну роботу

«Розробка методів дослідження нейронних мереж з використанням середовища Wolfram Mathematica та мови програмування C++»

Розробники:

виконавець ст. гр. СПм-61

Лавренів Андрій Дмитрович

_____

(підпис)

керівник роботи

К.ф.-м. н., доцент, Бойко І.В

_____

(підпис)

Тернопіль 2021

ЗМІСТ

1 ПІДСТАВИ ДО РОЗРОБКИ

Розробка проводиться у відповідності до графіку навчального плану підготовки магістрів за спеціальністю 121 «Інженерія програмного забезпечення».

Тема роботи: «Розробка методів дослідження нейронних мереж з використанням середовища Wolfram Mathematica та мови програмування C++».

Термін виконання: до “___”_____ 2021 р.

2 ПРИЗНАЧЕННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

Програмний продукт призначений для дослідження нейронних мереж та робити з ними. Для реалізації використано мову програмування C++.

Мета, яка поставлена в даній магістерській роботі досягається шляхом виконання таких завдань для середовища Wolfram Mathematica:

- дослідження основних видів нейронних мереж та способів їх подання;
- навчання нейронної мережі;
- формування репозиторію нейронної мережі Wolfram;
- способи отримання необхідної інформації у нейронній мережі;
- розробка самостійної LeNet архітектури та MXNet фреймворку для нейронних мереж.

3 ВИМОГИ ДО ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Функціональні вимоги

Система повинна виконувати наступні вимоги:

- Здатність роботи на хмарних ресурсах із мінімальними ресурсами;
- Можливість інтеграції системи з різними операційними системами;
- Можливість використовувати навчену нейронну мережу як portable програму, попередньо налаштовану до певного кола задач;
- Швидкий доступ до оновлюваних ресурсів для навчання і покращення роботи вже навченої мережі.

3.2 Технічні вимоги

Вимоги до клієнтської частини: підтримка в ОС мови програмування C++, та середовища Wolfram Mathematica.

Додаткові вимоги: не менше 8ГБ ОЗП, не менше 4ГБ місця на жорсткому диску.

3.3 Програмні вимоги

Використання мови програмування та середовища:

- C++;
- Wolfram Mathematica;
- Microsoft Visual Studio;

Додаткові вимоги: 500 Мб на хмарному ресурсі Wolfram (забезпечується у режимі вільного доступу)

4 ЕТАПИ РОЗРОБКИ

Розробка інформаційної системи проводилась в наступному порядку:

- аналіз особливостей та принципів роботи з нейронними мережами в середовищі Wolfram Mathematica;
- дослідження властивостей нейронних мереж;
- розробка фреймворку нейронної мережі;
- написання пояснювальної записки до кваліфікаційної роботи;
- створення презентації до кваліфікаційної роботи.

5 СУПРОВІДНА ДОКУМЕНТАЦІЯ

Для інформаційної системи повинні бути розроблені наступні документи:

- технічне завдання;
- пояснювальна записка до роботи;
- презентація роботи;
- рецензія на роботу.

6 ПОРЯДОК ЗДАЧІ РОБОТИ

Розроблена інформаційна системи повинна відповідати вимогами, що складаються з перерахованих у п.3.1 цього документу характеристик.

Для здачі роботи необхідно підготувати весь перелік документів зазначений у п.5 цього документу.

Приймання роботи проводиться спеціально створеною екзаменаційною комісією №43 в термін до:

“___”_____ 2021 р.

7 ВІДМІТКИ ПРО ВИКОНАННЯ ЕТАПІВ ТА ЗМІНИ В РОБОТІ

Назва етапу	Відмітка*
Аналіз особливостей та принципів роботи з нейронними мережами в середовищі Wolfram Mathematica	Виконано
Дослідження властивостей нейронних мереж	Виконано
Розробка фреймворку нейронної мережі	Виконано
Охорона праці та безпека в надзвичайних ситуаціях	Виконано
Підготовка звіту	Виконано
Підготовк апрезентації та доповіді	Виконано
Попередній захист	Виконано
Нормконтроль, рецензування	Виконано
Захист кваліфікаційної роботи	

* відмітки про виконання етапу ставляться керівником роботи

ДОДАТОК Б

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Тернопільський національний технічний університет імені Івана Пулюя (Україна)
Університет імені П'єра і Марії Кюрі (Франція)
Маріборський університет (Словенія)
Технічний університет у Кошице (Словаччина)
Вільнюський технічний університет ім. Гедимінаса (Литва)
Білоруський національний технічний університет (Республіка Білорусь)
Міжнародний університет цивільної авіації (Марокко)
Наукове товариство ім. Т.Шевченка

АКТУАЛЬНІ ЗАДАЧІ СУЧАСНИХ ТЕХНОЛОГІЙ

Збірник
тез доповідей
Том І

X Міжнародної науково-практичної конференції молодих учених та студентів
24-25 листопада 2021 року

УКРАЇНА
ТЕРНОПІЛЬ – 2021

УДК 004.434; 004.852

А.Д. Лавренів, І.В. Бойко канд. фіз.-мат. наук, доцент
Тернопільський національний технічний університет імені Івана Пулюя, Україна

РОЗРОБКА МЕТОДІВ ДОСЛІДЖЕННЯ НЕЙРОННИХ МЕРЕЖ З ВИКОРИСТАННЯМ СЕРЕДОВИЩА WOLFRAM MATHEMATICA ТА МОВИ ПРОГРАМУВАННЯ C++

A.D. Lavreniv , I.V. Boyko Ph.D, Assoc. Prof.

DEVELOPMENT OF METHODS FOR RESEARCH OF NEURAL NETWORKS USING WOLFRAM MATHEMATICA AND C ++ PROGRAMMING LANGUAGE

В мові Wolfram Language нейронні мережі побудовані з шарів. Шар - це термін, який можна застосовувати до набору вузлів, які працюють разом на певному рівні в нейронній мережі. Дані, що обробляються шарами, відносяться до числового типу, а не будь-якого іншого. Вхідні змінні можуть бути: вектором, одновимірним списком; матрицями, двовимірними списками і масиви, списками списків або будь-яким іншим числовим тензором.

Лінійний шар є найбільш поширеним і широко використовуваним шаром в нейронної мережі.

Рис. 1. Об’єкт, що відповідає лінійному шару LinearLayer

Кожен створений нами шар має порт введення і порт виводу. Кожен порт має пов'язаний з ним розмір того, що входить в шар, а що виходить. Загальний вигляд лінійного шару задається наступним виразом скалярного добутку $\boldsymbol{w \cdot x} + \boldsymbol{b}$, де $\boldsymbol{x}$ - вектор даних, $\boldsymbol{w}$ - матриця ваг, а $\boldsymbol{b}$ - вектор зміщення.

Підготувавши дані і модель, приступаємо до навчання моделі. Як тільки навчання починається, отримується інформаційна панель з чотирма основними результатами.

1. Summary: містить релевантну інформацію про партії, раундах і терміни.
2. Data: включає оброблену інформацію про дані.
3. Method: показує використовуваний метод, розмір партії і пристрій, що використовується для навчання.
4. Round: поточний стан розміру збитку.

Після того, як навчання пройдено, отримання навченої мережі та реалізації моделі виглядає наступним чином на Рис.2

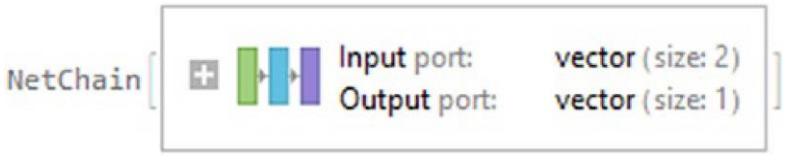


Рис. 2. Виведена навчена мережа

Тепер продемонструємо, як навчена мережа ідентифікує кожну з точок, викреслюючи кордон з графіком розподілу щільності.

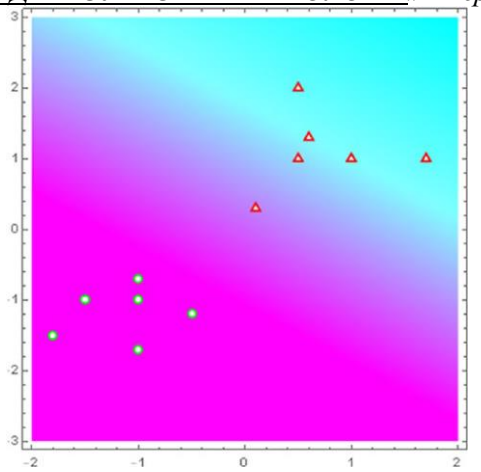


Рис. 3. Графік класифікації нейронної мережі

Дивлячись на графік, ми бачимо, що кордони нечітко визначені і точки, близькі до нуля, можуть бути неправильно класифіковані. Проте незважаючи на це, цю модель можливо поліпшити. Якщо розмір пакета не вказано, він буде мати автоматичне значення, майже завжди значення рівне 64 або ступеню двійки. Слід розуміти, що розмір пакету вказує кількість прикладів, які модель використовує при навчанні перед оновленням внутрішніх параметрів моделі. Кількість пакетів - це поділ прикладів в навчальному наборі даних за розміром пакета. Оброблені приклади - це кількість раундів (епох), помножене на кількість навчальних прикладів. Як правило, розмір пакета вибирається таким чином, щоб він рівномірно ділив розмір навчального набору.

Коли пройти весь навчальний цикл тільки один раз, це називається епохою. Щоб краще зрозуміти це, в попередньому прикладі автоматично був обраний розмір пакета, що дорівнює 12, що дорівнює кількості прикладів в навчальному наборі. Тепер кількість епох було автоматично вибрано рівним 10000. При такій кількості епох втрати значно знизяться що ми побачимо на Рис. 4.

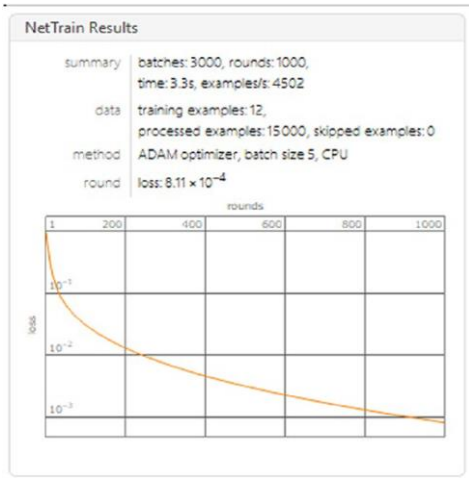


Рис. 4. Результати навчання

Література:

[1] S. Wolfram. An Elementary Introduction to the Wolfram Language Second Edition. Wolfram media, 2017. 339 p.

[2] J.A. Freeman. Simulating Neural Networks with Mathematica. Addison-Wesley Professional, 1993. 352 p.

[3] R.J. Gaylord, S.N. Karmin, P.R. Wellin. An Introduction to Programming With Mathematica Second Edition : Springfield, 1995. 475p.

Лістинг коду LeNet-Trained-on-MNIST-Data.nb

Resource retrieval

Retrieve the pre-trained net:

In[154]:= NetModel["LeNet Trained on MNIST Data"]

Out[154]= NetChain[Input image

array (size: 1*28*28)

1	ConvolutionLayer	array (size: 20*24*24)
2	Ramp	array (size: 20*24*24)
3	PoolingLayer	array (size: 20*12*12)
4	ConvolutionLayer	array (size: 50*8*8)
5	Ramp	array (size: 50*8*8)
6	PoolingLayer	array (size: 50*4*4)
7	FlattenLayer	vector (size: 800)
8	LinearLayer	vector (size: 500)
9	Ramp	vector (size: 500)
10	LinearLayer	vector (size: 10)
11	SoftmaxLayer	vector (size: 10)
	Output	class

]

Basic usage

Apply the trained net to a set of inputs:

In[155]:= NetModel["LeNet Trained on MNIST Data"][{, , }]

Out[155]= {6,8,0}

Give class probabilities for a single input:

In[156]:= NetModel["LeNet Trained on MNIST Data"][, "Probabilities"]

Out[156]=
<|0->7.28386*10^-13,1->1.8159*10^-16,2->2.15518*10^-12,3->3.2635*10^-12,4->9.74528*10^-15,
5->6.61678*10^-10,6->3.45442*10^-13,7->3.57894*10^-20,8->1.,9->2.42621*10^-12|>

Feature extraction

Create a subset of the MNIST dataset:

In[157]:= sample=Keys[RandomSample[ResourceData["MNIST"],150]]

Out[157]=
{,,}

Remove the last linear layer of the net, which will be used as a feature extractor:

In[158]:= extractor=Take[NetModel["LeNet Trained on MNIST Data"],{1,-3}]

Out[158]= NetChain[Input port: image

Output port: vector (size: 500)

Number of layers: 9

]

Visualize the features of a subset of the MNIST dataset:

In[159]:= FeatureSpacePlot[sample,FeatureExtractor->extractor]

Out[159]=

Visualization of net operation

Extract the convolutional features from the first layer:

In[160]:= convFeatures=NetModel["LeNet Trained on MNIST Data"][,NetPort[1,"Output"]]

Out[160]= {{ \[CenterEllipsis]1\[CenterEllipsis] }, \[CenterEllipsis]18\[CenterEllipsis] ,{{-1.80282,-1.80282,-1.80282,-1.80282,-1.80282,-1.80282, \[CenterEllipsis]12\[CenterEllipsis] ,-1.80282,-1.80282,-1.80282,-1.80282,-1.80282,-1.80282}, \[CenterEllipsis]22\[CenterEllipsis] ,{ \[CenterEllipsis]1\[CenterEllipsis] }}}

large output show less show more show all set size limit...

Visualize the features:

In[161]:= ImageCollage[Map[ImageAdjust@*Image,convFeatures],ImageSize->Medium]

Out[161]=

Training the uninitialized architecture

Retrieve the uninitialized architecture:

In[162]:= net=NetModel["LeNet Trained on MNIST Data","UninitializedEvaluationNet"]

Out[162]= NetChain[Input port: image

Output port: class

Number of layers: 11

]

Retrieve the MNIST dataset:

In[163]:= mnist=ResourceObject["MNIST"]

Out[163]= ResourceObject[Name: MNIST »

Type: DataResource \[SpanFromLeft]

Description: Dynamic \[SpanFromLeft]

]

Use the training dataset provided:

In[164]:= trainSet=ResourceData[mnist,"TrainingData"]

Out[164]=
{->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,-
>0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,
\[CenterEllipsis]59900\[CenterEllipsis]
,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,-
>9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9}

large output show less show more show all set size limit...

Use the test dataset provided:

In[165]:= valSet=ResourceData[mnist,"TestData"]

Out[165]=
{->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,-
>0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,
\[CenterEllipsis]9900\[CenterEllipsis]
,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,-
>9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9}

large output show less show more show all set size limit...

Train the net:

In[166]:= trained=NetTrain[net,trainSet,ValidationSet->valSet]

Out[166]= NetChain[Input port: image

Output port: class

Number of layers: 11

]

Generate a ClassifierMeasurementsObject of the net with the test set:

In[167]:= cm=ClassifierMeasurements[trained,valSet]

Out[167]= ClassifierMeasurementsObject[Classifier: Net

Number of test examples: 10000

Dynamic

]

Evaluate the accuracy on the validation set:

In[168]:= cm["Accuracy"]

Out[168]= 0.9912

Visualize the confusion matrix:

In[169]:= cm["ConfusionMatrixPlot"]

Out[169]=

Net information

Inspect the number of parameters of all arrays in the net:

In[170]:= NetInformation[NetModel["LeNet Trained on MNIST Data"],"ArraysElementCounts"]

Out[170]=
<|{1,Biases}->20,{1,Weights}->500,{10,Biases}->10,{10,Weights}->5000,{4,Biases}->50,{4,Weights}->25000,{8,Biases}->500,{8,Weights}->400000|>

Obtain the total number of parameters:

In[171]:= NetInformation[NetModel["LeNet Trained on MNIST Data"],"ArraysTotalElementCount"]

Out[171]= 431080

Obtain the layer type counts:

In[172]:= NetInformation[NetModel["LeNet Trained on MNIST Data"],"LayerTypeCounts"]

Out[172]=
<|ConvolutionLayer->2,ElementwiseLayer->3,PoolingLayer->2,FlattenLayer->1,LinearLayer->2,Softmax Layer->1|>

Display the summary graphic:

In[173]:= NetInformation[NetModel["LeNet Trained on MNIST Data"],"SummaryGraphic"]

Out[173]=

Export to MXNet

Export the net into a format that can be opened in MXNet:

In[174]:= jsonPath=Export[FileNameJoin[{$TemporaryDirectory,"net.json"}],NetModel["LeNet Trained on MNIST Data"],"MXNet"]

Out[174]= /private/var/folders/pz/94mxs33x2l512z6wtjbthvy0000_ck/T/net.json

Export also creates a net.params file containing parameters:

In[175]:= paramPath=FileNameJoin[{DirectoryName[jsonPath],"net.params"}]

Out[175]= /private/var/folders/pz/94mxs33x2l512z6wtjbthvy0000_ck/T/net.params

Get the size of the parameter file:

In[176]:= FileByteCount[paramPath]

Out[176]= 1724806

The size is similar to the byte count of the resource object:

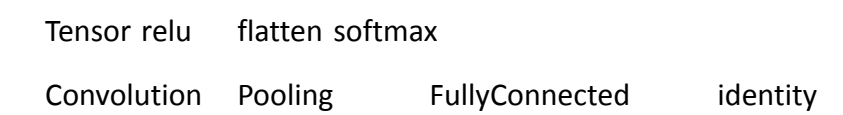
In[177]:= ResourceObject["LeNet Trained on MNIST Data"]["ByteCount"]

Out[177]= 1750489

Represent the MXNet net as a graph:

In[178]:= Import[jsonPath,{"MXNet","NodeGraphPlot"}]

Out[178]=



Лістинг коду CapsNet-Trained-on-MNIST.nb

Resource retrieval

Retrieve the pre-trained net:

In[21]:= NetModel["CapsNet Trained on MNIST Data"]

Out[21]= NetGraph[

Inputs	Outputs
Input: image	Classification: class
	Reconstruction: image

]

Basic usage

Apply the trained net to a set of inputs:

In[22]:= NetModel["CapsNet Trained on MNIST Data"][{, , }]

Out[22]= <|Classification->{6,8,0},Reconstruction->{,,}|>

Give class probabilities for a single input:

In[23]:= NetModel["CapsNet Trained on MNIST Data"][,"Classification"-> "Probabilities"]

Out[23]=
<|0->0.0609864,1->0.0358106,2->0.0273417,3->0.0746369,4->0.0634226,5->0.0679507,6->0.04934,7->0.026264,8->0.941028,9->0.0415873|>

Feature extraction

Create a subset of the MNIST dataset:

In[24]:= sample=Keys[RandomSample[ResourceData["MNIST"],100]]

Out[24]= {,,}

Remove the last linear layer of the net, which will be used as a feature extractor:

In[25]:= extractor=NetTake[NetModel["CapsNet Trained on MNIST Data"],{"ReLUConv1","Pick"}]

Out[25]= NetGraph[Input port: image

Output port: vector (size: 16)

Number of layers: 7

]

Visualize the features of a subset of the MNIST dataset:

In[26]:= FeatureSpacePlot[sample,FeatureExtractor->extractor]

Out[26]=

Image generation

Extract the image reconstruction part:

In[27]:= reconstructor=NetReplacePart[NetExtract[NetModel["CapsNet Trained on MNIST Data"],"Reconstruct"],"Output"->NetDecoder["Image"]]

Out[27]= NetChain[Input port: vector (size: 16)

Output port: image

Number of layers: 7

]

Extract the DigitCaps feature vector for a given digit image:

In[28]:= featureVect=NetModel["CapsNet Trained on MNIST Data"][,NetPort["Pick","Output"]]

Out[28]=
{0.315018,0.0878248,-0.367435,-0.143667,-0.385807,-0.177227,0.279463,0.292916,0.161793,-0.187058,-0.128582,0.156842,-0.173882,-0.059431,-0.34979,-0.0273752}

Reconstruct the image from the feature vector:

In[29]:= reconstructor[featureVect]

Out[29]=

Experiment with changing the feature vector. Add a shift along a single coordinate at a time:

In[30]:= reconstructor@Table[featureVect+c*UnitVector[16,11],{c,-1,1,0.1}]

Out[30]= {,,,,,,,,,,,,,,,,,,,,}

In[31]:= reconstructor@Table[featureVect+c*UnitVector[16,14],{c,-1,1,0.1}]

Out[31]= {,,,,,,,,,,,,,,,,,,,,}

In[32]:= reconstructor@Table[featureVect+c*UnitVector[16,16],{c,-1,1,0.1}]

Out[32]= {,,,,,,,,,,,,,,,,,,,,}

Training the uninitialized architecture

Retrieve the uninitialized training architecture:

In[33]:= trainingNet=NetModel["CapsNet Trained on MNIST Data","TrainingNet"]

Out[33]= NetGraph[Number of inputs: 2

Number of outputs: 2

Number of layers: 5

]

Retrieve the MNIST dataset:

In[34]:= mnist=ResourceObject["MNIST"]

Out[34]= ResourceObject[Name: MNIST »

Type: DataResource \[SpanFromLeft]

Description: Dynamic \[SpanFromLeft]

]

Use the training dataset provided:

In[35]:= trainSet=ResourceData[mnist,"TrainingData"]

Out[35]=
{->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,
\[CenterEllipsis]59900\[CenterEllipsis]
,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9}

large output show less show more show all set size limit...

Use the test dataset provided:

In[36]:= valSet=ResourceData[mnist,"TestData"]

Out[36]=
{->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,->0,
\[CenterEllipsis]9900\[CenterEllipsis]
,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9,->9}

large output show less show more show all set size limit...

Initialize the "W" matrices properly:

```
In[37]:= trainingCapsNetInitialized=NetReplacePart[
trainingNet,
{"CapsNet","PrimaryPredVects",2,"Array"}->RandomVariate[UniformDistribution[{-1,1}*0.005],{1152,
10,16,8}]
]

Out[37]= NetGraph[Number of inputs:          2
Number of outputs:    2
Number of layers:       5

]
```

Train the net (if a GPU is available, setting TargetDevice -> "GPU" is recommended):

```
In[48]:= trained=NetTrain[
trainingCapsNetInitialized,trainSet,
ValidationSet->valSet,
LossFunction->{"ClassLoss"->Scaled[1],"RecoLoss"->Scaled[0.392]},
MaxTrainingRounds->50,
TargetDevice->"CPU"
]
Out[48]= NetGraph[Number of inputs:          2
Number of outputs:    2
Number of layers:       5

]
```

Net information

Inspect the number of parameters of all arrays in the net:

```
In[39]:= NetInformation[NetModel["CapsNet Trained on MNIST Data"],"ArraysElementCounts"]
```

Out[39]=
<|{PrimaryCaps,1,Biases}->256,{PrimaryCaps,1,Weights}->5308416,{PrimaryPredVects,2,Array}->1474560,{Reconstruct,1,Biases}->512,{Reconstruct,1,Weights}->8192,{Reconstruct,3,Biases}->1024,{Reconstruct,3,Weights}->524288,{Reconstruct,5,Biases}->784,{Reconstruct,5,Weights}->802816,{ReLUConv1,1,Biases}->256,{ReLUConv1,1,Weights}->20736|>

Obtain the total number of parameters:

In[40]:= NetInformation[NetModel["CapsNet Trained on MNIST Data"],"ArraysTotalElementCount"]

Out[40]= 8141840

Obtain the layer type counts:

In[41]:= NetInformation[NetModel["CapsNet Trained on MNIST Data"],"LayerTypeCounts"]

Out[41]=
<|ConvolutionLayer->2,ElementwiseLayer->12,TransposeLayer->1,ReshapeLayer->4,AggregationLayer->8,ReplicateLayer->8,ThreadingLayer->8,ConstantArrayLayer->1,SoftmaxLayer->2,FlattenLayer->1,SequenceLastLayer->1,OrderingLayer->1,ExtractLayer->1,LinearLayer->3|>

Display the summary graphic:

In[42]:= NetInformation[NetModel["CapsNet Trained on MNIST Data"],"SummaryGraphic"]

Out[42]=

Export to MXNet

Export the net into a format that can be opened in MXNet:

In[43]:= jsonPath=Export[FileNameJoin[{$TemporaryDirectory,"net.json"}],NetModel["CapsNet Trained on MNIST Data"],"MXNet"]

Out[43]= /private/var/folders/pz/94mxs33x2l512z6wtjbthvy0000_ck/T/net.json

Export also creates a net.params file containing parameters:

In[44]:= paramPath=FileNameJoin[{DirectoryName[jsonPath],"net.params"}]

Out[44]= /private/var/folders/pz/94mxs33x2l512z6wtjbthvy0000_ck/T/net.params

Get the size of the parameter file:

In[45]:= FileByteCount[paramPath]

Out[45]= 32568233

The size is similar to the byte count of the resource object:

In[46]:= ResourceObject["CapsNet Trained on MNIST Data"]["ByteCount"]

Out[46]= 32834400

Represent the MXNet net as a graph:

In[47]:= Import[jsonPath,{"MXNet","NodeGraphPlot"}]

Out[47]=

Tensor sqrt SwapAxis BlockGrad FullyConnected

Convolution _PlusScalar split argmax sigmoid

relu _PowerScalar softmax sign identity

transpose _Mul flatten _Minus

reshape broadcast_axis _Plus broadcast_mod

square expand_dims concat _arange

sum_axis broadcast_like SequenceLast gather_nd

ДОДАТОК Г

ДИСК