УДК 004
DOI: https://doi.org/10.26642/ten-2020-1(85)-135-146

**О.В. Коротун, к.пед.н., доц.**
**Г.В. Марчук, ст. викладач**
**Д.К. Марчук, аспірант**
**О.В. Талавер, студент**
*Державний університет «Житомирська політехніка»*

# Система розпізнавання рукописних цифр з оцінкою якості

*У статті розглянуто одну з областей штучного інтелекту – нейронні мережі, що застосовуються у різних галузях суспільства. Функціонування будь-якої системи стає набагато ефективніше за допомогою розв'язання задач на основі нейронних мереж. Класи задач, які можна розв'язувати за допомогою нейронних мереж, визначаються тим, як вони працюють і навчаються. Метою роботи є дослідження потенційних можливостей різних алгоритмів розпізнавання рукописних символів, зокрема цифр. Написання бібліотеки для роботи з нейронними мережами та розробка програмного додатка, який дасть можливість створювати, зберігати, тренувати та тестувати різні моделі нейронних мереж.*

*Результатом дослідження визначено перевірку спроєктованої моделі нейронної мережі з мінімальною кількістю шарів та аналіз якості розпізнавання.*

*У роботі проаналізовані різні моделі нейронних мереж, а саме мережі прямого поширення (feed forward neural networks) або багатошаровий персептрон (perceptrons). Досліджено найвідоміший алгоритм навчання – так званий алгоритм зворотного поширення (back propagation). Проведено аналіз порогової та сигмоїдальної функції активації. Для знаходження мінімуму функції використовувався градієнт.*

*Застосовано набір даних MNIST (Modified National Institute of Standards and Technology database) – це велика база рукописних цифр, яка зазвичай використовується для навчання різних систем обробки зображень. Побудована бібліотека для роботи з нейронною мережею, яка разом з набором тренувальних даних MNIST дала акуратність близько 98 %.*

*Методи, продемонстровані у роботі, є основою для побудови більш складних систем. Розроблений програмний додаток дає можливість створювати, зберігати, тренувати та тестувати нейронні мережі.*

***Ключові слова**: нейрон; нейронна мережа; перцептрон; помилка; алгоритм; навчання; тренування.*

**Актуальність теми.** Широко досліджуваною проблемою, яка розпочалася ще у 70-х роках XX століття, є розпізнавання рукописного тексту. Зараз точність розпізнавання нижча, ніж для рукописного «друкарського» тексту. І досі виникає багато проблем, що пов'язані з великим різноманіттям написання окремих рукописних символів. Розробки у цьому напрямі проводяться університетами різних країн: США, Швейцарії, Франції тощо.

Нещодавній вибух технологій у сфері нейронних мереж сприяв поліпшенню сучасних прийомів, що використовуються в обробці тексту, виявленні об'єктів тощо.

На сьогодні можна зустрітися з цією технологією у багатьох програмах, враховуючи Google Maps, Google Lens тощо.

**Аналіз останніх досліджень та публікації, на які спирається автор.** На думку П.А. Хаустова [8], істотною перевагою розробки алгоритмів розпізнавання рукописних символів на основі побудови структурних моделей є можливість роботи в умовах малої кількості еталонних зображень. Запропонований підхід до скелетизації бінарного представлення зображення символу, заснований на спільному застосуванні алгоритмів Зонга-Суня і Ву-Цая, не дав бажаних результатів на наборі рукописних цифр MNIST.

У [9] проаналізовані найпоширеніші методи розпізнавання символів на прикладі поетапного процесу розпізнавання автомобільних номерів. Наведено переваги і недоліки кожного з методів, запропоновані варіанти застосування методів залежно від поставленого завдання. Якщо розглядати методи і способи розпізнавання символів, то тут перевагу матиме нейронна мережа, оскільки вона може навчатися у процесі роботи.

**Метою статті** є дослідження потенційних можливостей різних алгоритмів розпізнавання рукописних символів, зокрема цифр, з використанням нейронної мережі; написання бібліотеки для роботи з декількома видами нейронних мереж; розробка програмного додатка, який дасть можливість створювати, зберігати, тренувати та тестувати різні моделі нейронних мереж.

Результатом дослідження буде перевірка спроєктованої моделі нейронної мережі з мінімальною кількістю шарів та аналіз якості розпізнавання.

**Викладення основного матеріалу.** Роботу систем розпізнавання символів можна поділити на три основні етапи [3]:

- *попередня обробка зображення* – зображення вирівнюється, прибирається шум, зображення конвертується в бінарне (чорно-біле), видаляються усі лінії, які не є частиною зображення, аналізується позиціювання блоків тексту та завершується виділенням окремих параграфів, рядків, символів;
- *розпізнавання символів* – на цьому етапі застосовується декілька підходів, вибір яких залежить від типу тексту (рукописний, друкований). Для розпізнавання друкованого тексту використовують алгоритм k-nearest neighbors, для рукописного – нейронні мережі;
- *подальша обробка* – для підвищення точності, на останньому етапі порівнюють отримані слова із словами зі словника.

У роботі буде відтворено частину системи розпізнавання цифр за допомогою публічних баз даних, таких як MNIST, та власної бібліотеки.

**Аналіз роботи та розробка власного методу побудови нейронної мережі.** Штучна нейронна мережа – це система, архітектура якої була натхненна будовою мозку тварин. Нейронні мережі нагадують справжні нейрони. Найпростішим прикладом штучного нейрона є перцептрон (від англ. perceptron) (рис. 1), який був розроблений ще у 1950–1960 роках Френком Розенблатом [10].

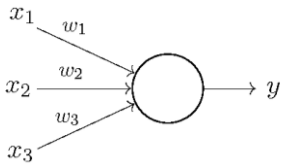



*Рис. 1. Перцептрон*

Перцептрон приймає декілька вхідних сигналів (рис. 1, *x*), кожен з них має власну вагу (рис. 1, *w*). Вихідний сигнал (рис. 1, *y*) визначається шляхом сумування усіх добутків отриманих сигналів з відповідними вагами. Після цього отриманий результат порівнюється з пороговим значенням.

$$output = \begin{cases} 0 & if \sum_{i=1}^{n} w_i x_i \le threshold \\ 1 & if \sum_{i=1}^{n} w_i x_i > threshold \end{cases}. \quad (1)$$

Знаючи лише це, можна використати один нейрон для розрахування певного рішення, опираючись на декілька вхідних параметрів, наприклад, якщо потрібно дізнатися, чи підемо ми сьогодні в університет, де вхідні параметри будуть: сьогодні вихідний день (1 – так, 0 – ні), сьогодні є заняття (1 – так, 0 – ні), самопочуття (0 – погане, 1 – нормальне), підібравши коефіцієнти ваг та поріг, що дорівнює п’яти, можна побудувати найпростішу модель, яка буде працювати доволі справно. Звісно, один перцептрон не є повною моделлю прийняття рішень, але це описує основну ідею того, як працює нейронна мережа.

Модель нейронної мережі, зображена на рисунку 2, називається багатошаровий перцептрон. Існує три види шарів: вхідний, вихідний та прихований (рис. 2). Нейронну мережу з більш ніж одним прихованим шаром називають глибокою нейронною мережею (Deep Neural Network) [11].

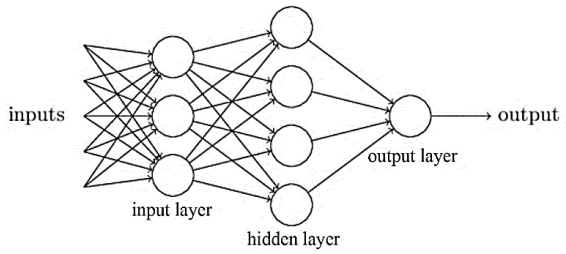



*Рис. 2. Приклад найпростішої нейронної мережі*

Для того, щоб спростити рівняння (1), введемо нову змінну, яка з англійської звучить як схильність (bias), *bias = -threshold*

$$output = \begin{cases} 0 & if \sum_{i=1}^{n} w_i x_i + bias \le 0 \\ 1 & if \sum_{i=1}^{n} w_i x_i + bias > 0 \end{cases}. \quad (2)$$

Уявимо, що є модель нейронної мережі для вирішення певного завдання. Ідея полягає у тому, щоб підібрати такі ваги, які дадуть бажаний результат. Оскільки під час створення усі параметри нейронів

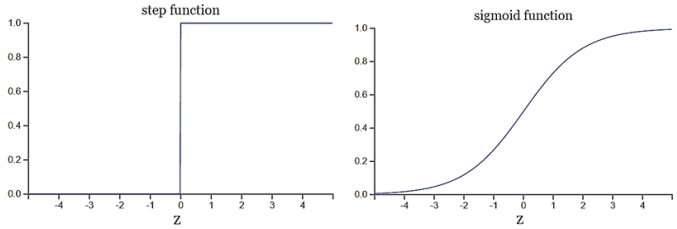
підбираються довільним чином, то спочатку мережа нейронів ледве буде розуміти, як підійти до проблеми і буде видавати неправильні результати. Тому, щоб вплинути на результат, потрібно взяти довільний параметр та частково змінити його, після цього можна буде побачити, що вихідні значення також змінилися.

У такому випадку натренувати нейронну мережу не було б такою великою проблемою, але перцептрон не дасть можливість це зробити, адже при невеликій зміні ваг у першому випадку вихідне значення не зміниться. У другому випадку зміниться на протилежне, що може спричинити великі зміни у вихідних даних.

Для вирішення цієї проблеми можна використати сигмоїдні нейрони. Вони дуже схожі на перцептрон, єдина відмінність – це функція активації, яку треба застосовувати до суми добутків ваг на вхідні значення, тому вихідне значення може бути будь-яким значенням між 0 та 1.

$$\sigma(x) = \frac{1}{1+\exp(-\sum_j w_j x_j - bias)}. \tag{3}$$

Перцептрон має порогову функцію активації (рис. 3, *a*), і якщо порівняти її з сигмоїдною (рис. 3, *b*), то можна побачити: друга має плавний перехід, що дасть можливість спостерігати плавну зміну вихідного значення нейрона.




*a)* *b)*

*Рис. 3. Графік функцій активації: a) порогова, b) сигмоїдна*

Отже, формула зміни вихідного значення нейрона буде приблизно дорівнювати сумі добутків зміни ваг на залежність вихідного значення від ваг.

$$\Delta output \approx \sum_j \frac{\partial output}{\partial w_j} \Delta w_j + \frac{\partial output}{\partial bias} \Delta bias\ . \tag{4}$$

Feed-Forward алгоритм [12] дозволяє отримати опрацьовані нейронною мережею дані. Суть алгоритму полягає у тому, щоб передати вхідні дані на перший (вхідний) шар після чого зібрати вихідні значення та передати їх на вхід наступного шару, і так, поки не отримаємо остаточні результати з останнього (вихідного шару).

Одним з методів тренування нейронної мережі є алгоритм зворотного поширення помилки – Back Propagation [1]. Для кращого розуміння роботи алгоритму визначимо, яким чином можна змінити вихідні дані нейронної мережі. Відповідно до алгоритму прямого поширення похибки, потрібно знайти добуток вхідних даних у нейрон та ваг, після чого додати змінну схильності (bias) та застосовувати функцію активації. Тому, якщо спробувати змінити значення ваги або змінної схильності, можна змінити вихідне значення нейронної мережі. Знаходження відповідних ваг та змінних схильності і є тренуванням.

З метою визначення того, наскільки добре нейронна мережа робить свою роботу, введемо функцію вартості, або функцію помилки (Cost Function), що дорівнює середній сумі різниці бажаних результатів та фактичних результатів у квадраті.

$$C(w, bias) = \frac{1}{2n}\sum_x \left\| y(x) - a \right\|^2. \tag{5}$$

Насправді це лише одна з можливих функцій, якої вистачить для початку. Тепер кожного разу, коли необхідно отримувати вихідні дані мережі, можна порівняти їх з очікуваними, використовуючи квадратичну функцію помилки. Чим більше отримане значення буде відмінне від нуля, тим більше ваги мають бути скоректовані. Задача полягає у тому, щоб мінімізувати функцію помилки, використовуючи алгоритм відомий як градієнтний спуск (Gradient Descent).

Виникає питання доцільності введення додаткової функції, якщо можна знайти кількість правильно визначених результатів. Насправді проблема у тому, що, використовуючи функцію кількості правильно класифікованих результатів, не потрібно точно оцінювати ситуацію, оскільки при зміні ваг кількість коректних відповідей може і не змінитися. У той же час функція помилки дасть миттєвий фідбек.

Для знаходження мінімуму функції скористаємося градієнтом. Обрахувати градієнт можна, знайшовши похідну від усіх змінних (залежність функції від конкретної змінної) функції та записати їх у вектор:

$$\nabla C = \left( \frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2} \right). \tag{6}$$

Розрахувати зміну градієнта можливо, якщо помножити вектор градієнта на вектор змін змінних:

$$\Delta C \approx \nabla C \cdot \Delta v . \tag{7}$$

З цього рівняння чітко видно, як потрібно змінити змінні, щоб зменшити помилку:

$$\Delta v \approx -\eta \nabla C , \tag{8}$$

де η – коефіцієнт тренування (Learning Rate), який визначає з якою швидкістю нейронна мережа буде тренуватися.

Зазвичай вибирають невелике значення, щоб не пропустити найнижчу точку, цей, як і більшість інших параметрів, визначаються експериментально. Тепер, коли є значення, на яке потрібно зменшити змінну, можна порахувати значення самої змінної:

$$v = v - \eta \nabla C . \tag{9}$$

Формула (9) використовується для знаходження значення ваг.

Отже, шар, керований значенням ваг, спускається до мінімуму функції. Використовуючи рівняння (9), можна застосувати алгоритм градієнтного спуску, замінивши $\Delta C$ на $\frac{\partial C}{\partial w_k}$ та $\frac{\partial C}{\partial bias}$, що є залежністю певних значень від функції *C*, для того, щоб змінити ваги та змінні схильності і таким чином зменшити функцію вартості нейронної мережі.

$$\begin{aligned} w_k &= w_k - \eta \frac{\partial C}{\partial w_k} \\ bias_l &= bias_l - \eta \frac{\partial C}{\partial bias_l} \end{aligned} . \tag{10}$$

Отже, було розглянуто, яким чином можливо вплинути на вихідні значення функції через зменшення вартості за допомогою зміни ваг та змінних схильності.

Для продовження, потрібно роз'яснити деякі позначення. $w_{jk}^{l}$ використовується для визначення ваги з'єднання *k*-го нейрона на *(l – 1)*-му шарі з *j*-им нейроном на *l*-му шарі (рис. 4).

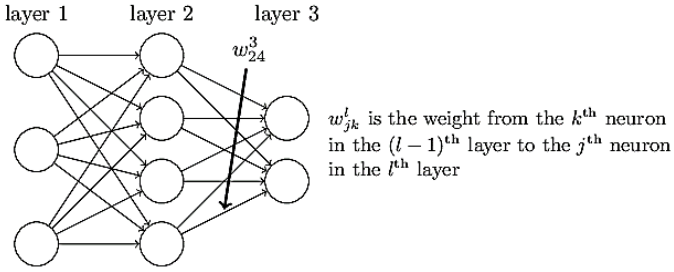



*Рис. 4. Пояснення позначень ваг*

Аналогічно позначимо змінні схильності та функції активації нейронної мережі (рис. 5). Використовуємо $b_j^l$ для схильності *j*-го нейрона на *l*-му шарі, $a_j^l$ – для позначення функції активації.

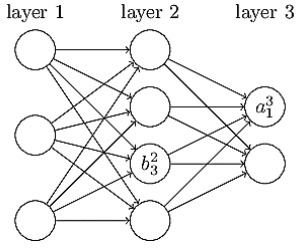



*Рис. 5. Пояснення позначень змінних схильності та функцій активації*

За такою формою запису з'являється достатня гнучкість для визначення складних формул, наприклад:

$$a_j^l = \sigma\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right). \quad (11)$$

Як показано в рівнянні, потрібно просумувати добутки відповідних ваг з вихідним значеннями із попереднього шару, додати змінну схильності, після чого нормалізувати. Для того щоб прибрати знак суми, використаємо векторну форму деяких значень, тому рівняння може бути переписане таким чином:

$$a^L = \sigma\left(w^L a^{L-1} + b^L\right). \quad (12)$$

Останньою зміною буде вираз зваженого входу у дужках, оскільки він буде з'являтися досить часто.

$$\begin{aligned} z^L &= w^L a^{L-1} + b^L \\ a^L &= \sigma\left(z^L\right) \end{aligned}. \quad (13)$$

Як вже було зазначено раніше (10), щоб змінити значення ваг та змінних схильності на нові, потрібно обчислити їх залежність від функції вартості, тобто $\frac{\partial C}{\partial_{w_k}}$ та $\frac{\partial C}{\partial bias_k}$. Оскільки кожного разу, коли вхідні дані потрапляють у нейрон, потрібно визначити зважений вхід, після чого нормалізувати його, лише потім вихідні дані можуть потрапити до наступного нейрона та за алгоритмом прямого поширення помилки дійти до кінця, що у результаті перетвориться у довгий ланцюг операцій.

Для початку потрібно спростити завдання, узявши нейронну мережу з двома шарами та двома нейронами, по одному нейрону на кожному шарі.

У цьому випадку на вхід подається значення, оскільки перший шар – вхідний, він просто передає його далі, де на другому нейроні визначається зважений вхід, він нормалізується, після чого визначається значення функції вартості (рис. 6).

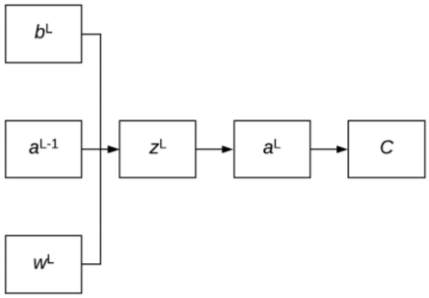



*Рис. 6. Ланцюг залежності функції вартості для останнього шару*

Отже, використовуючи правило ланцюга $\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial z_j^l}{\partial w_{jk}^l}\frac{\partial a_j^l}{\partial z_j^l}\frac{\partial C}{\partial a_j^l}$, крок за кроком визначаємо залежність функції вартості, спочатку від функції активації, після чого від зваженого входу і ваг нейрона. Далі визначимо конкретні значення похідних:

– $\frac{\partial C}{\partial a_j^l}$ – це похідна функції вартості (квадрата різниці) від функції активації, яка дорівнює $\frac{\partial C}{\partial a_j^l} = a_j^l - y$, де $y$ – бажаний результат;

– $\frac{\partial a_j^l}{\partial z_j^l}$ – похідна функції активації (сигмоїди) від зваженого входу та дорівнює $\frac{\partial a_j^l}{\partial z_j^l} = \sigma'(z_j^l) = \sigma(z_j^l)(1-\sigma(z_j^l))$, останньою є залежність ваг від зваженого входу $\frac{\partial z_j^l}{\partial w_{jk}^l}$ з

рівняння (13), за яким розраховується зважений вхід досить просто буде розрахувати похідну, адже вага – єдина змінна, тому $\frac{\partial z_j^l}{\partial w_{jk}^l} = a_k^{l-1}$.

Схожим чином визначаємо залежність функції вартості від змінної схильності $\frac{\partial C}{\partial bias_j^l} = \frac{\partial z_j^l}{\partial bias_j^l}\frac{\partial a_j^l}{\partial z_j^l}\frac{\partial C}{\partial a_j^l}$, оскільки рівняння дуже схожі, визначимо частину, яка відрізняється, а саме $\frac{\partial z_j^l}{\partial bias_j^l}$, знову з рівняння зваженого входу (13) визначаємо похідну, яка цього разу буде дорівнювати $\frac{\partial z_j^l}{\partial b_j^l} = 1$. Підставляючи ці рівняння у формулу оновлених параметрів (10), отримаємо:

$$\begin{aligned} w_{jk}^l &= w_{jk}^l - \eta\left(a_k^{l-1}\right)\left(\sigma'\left(z_j^l\right)\right)\left(a_j^l - y\right) \\ b_j^l &= b_j^l - \eta\left(\sigma'\left(z_j^l\right)\right)\left(a_j^l - y\right) \end{aligned}. \qquad 14$$

Дізнавшись, яким чином можливо змінити параметри останнього шару на прикладі двох нейронів, можемо плавно підійти до основного принципу роботи алгоритму зворотного розповсюдження.

До спрощеної нейронної мережі додамо ще один шар та спробуємо визначити залежність функції вартості від параметрів прихованого шару.

Аналогічно зазначеного вище побудуємо ланцюг рівнянь з додаванням прихованого (другого) шару.

З рисунка 7 ланки ланцюга почали повторюватися. Виведемо рівняння залежності функції вартості від ваг на передостанньому шарі $\frac{\partial C}{\partial w_{mk}^{l-1}} = \frac{\partial z_k^{l-1}}{\partial w_{mk}^{l-1}}\frac{\partial a_k^{l-1}}{\partial z_k^{l-1}}\frac{\partial z_j^{l-1}}{\partial a_k^{l-1}}\frac{\partial a_j^l}{\partial z_j^l}\frac{\partial C}{\partial a_j^l}$, щоб позбутися зайвого, вводимо нову змінну під назвою помилка (error):

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} = \frac{\partial a_j^l}{\partial z_j^l}\frac{\partial C}{\partial a_j^l} = \sigma'\left(z_j^l\right)\left(a_j^l - y\right). \qquad (15)$$

Тепер спростимо рівняння для оновлення ваг та змінних схильності:

$$\begin{aligned} w_{jk}^l &= w_{jk}^l - \eta\, a_k^{l-1}\delta_j^l \\ bias_j^l &= bias_j^l - \eta\,\delta_j^l \end{aligned}. \qquad (16)$$

Підставивши помилку у рівняння $\frac{\partial C}{\partial w_{mk}^{l-1}} = \frac{\partial z_k^{l-1}}{\partial w_{mk}^{l-1}}\frac{\partial a_k^{l-1}}{\partial z_k^{l-1}}\frac{\partial z_j^{l-1}}{\partial a_k^{l-1}}\frac{\partial a_j^l}{\partial z_j^l}\frac{\partial C}{\partial a_j^l}$, отримуємо $\frac{\partial C}{\partial w_{mk}^{l-1}} = \frac{\partial z_j^{l-1}}{\partial w_{mk}^{l-1}}\frac{\partial a_j^{l-1}}{\partial z_j^{l-1}}\frac{\partial z_j^l}{\partial a_j^{l-1}}\delta_j^l$ та здобуваємо метод поширення помилки на попередні шари:

$$\delta_j^l = \sum_j \left(w_{jk}^{l+1}\delta_j^{l+1}\right)\frac{\partial a_j^l}{\partial z_j^l}. \qquad (17)$$

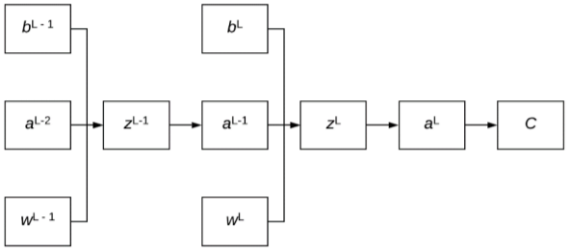



*Рис. 7. Ланцюг залежності функції вартості для передостаннього разу*

Поширюємо помилки на шари, після чого оновлюємо параметри нейронів. Тепер можемо записати повний алгоритм:

1. Feed-Forward: пропускаємо вхідні дані через нейронну мережу та отримуємо результати;
2. Визначаємо помилки вхідного шару: використовуємо рівняння (15) для того, щоб визначити помилки вхідного шару;
3. Поширюємо помилку на попередній шар: за допомогою рівняння (17) отримуємо помилки для попереднього шару;
4. Оновлюємо параметри нейронів: підставляємо параметри у рівняння (16) та отримуємо нові.

Зазначений вище алгоритм – алгоритм зворотного поширення помилки, але зазвичай його можна зустріти разом з алгоритмом тренування під назвою стохастичний градієнтний спуск (Stochastic Gradient Descent), суть якого полягає у тому, щоб тренувати партіями по декілька екземплярів за раз після чого вже оновлювати параметри нейронів, використовуючи середнє значення похідних.

**Методи покращення тренування нейронної мережі.** Потрібно наголосити, досі було обговорено тільки одну функцію вартості (5), але є функції, які можуть значно покращити швидкість тренування нейронної мережі. Якщо уважно придивитися до рівняння (17), то можна знайти одну проблему. Просідання дає похідна функції сигмоїди, яка дорівнює схилу функції в певній точці, тому, якщо вхідне значення або дуже мале, або дуже велике, то функція перенасичується, тобто похідна зменшується до нуля, це може означати, що на початку тренування нейронної мережі можливе уповільнення.

Щоб вирішити цю проблему, використаємо функцію під назвою Cross entropy:

$$C = \frac{1}{n}\sum_{x}\left[y\ln a + (1-y)\ln(1-a)\right]. \tag{18}$$

На перший погляд дуже складно зрозуміти, як вона допоможе, але визначивши похідну та підставивши у рівняння залежності ваг від функції вартості, отримаємо пояснення.

$$\begin{aligned}\frac{\partial C}{\partial_{w_j}} &= \frac{1}{n}\sum_{x}\frac{\sigma'(z)x_j}{\sigma(z)(1-\sigma(z))}(\sigma(z)y)\\ \frac{\partial C}{\partial_{w_j}} &= \frac{1}{n}\sum_{x}x_j(\sigma(z)-y)\end{aligned} \tag{19}$$

Можна побачити, що множник похідної функції сигмоїди скорочується, що вирішує одну з проблем.

Якщо говорити про функції активації та вартості, то їх достатньо багато, але застосовуються вони в різних випадках і не дають великого приросту в ефективності. Їх потрібно підбирати, зважаючи на завдання, яке стоїть перед нейронною мережею.

Перенасичення – досить велика проблема. Головною задачею тренування нейронних мереж є узагальнення тренувальних даних. З цієї причини варто використовувати багато тренувальних даних для того, щоб нейронна мережа змогла краще узагальнити шаблони.

Одним зі способів боротьби з перенасиченням є використання параметра регуляції. Її ідея полягає у додаванні змінної, яка буде відповідати за утримання значення ваг на певному рівні. Регулювати їх можна, зменшивши перед тим, як оновити:

$$w = \left(1-\frac{\eta\lambda}{n}\right)w - \frac{\eta}{m}\sum_{x}\frac{\partial C}{\partial w}, \tag{20}$$

де λ – коефіцієнт регуляції. Чим він більше, тим менші будуть ваги.

Одним із шляхів вдосконалення нейронної мережі є додавання шарів. Так, наприклад, шар софтмакс (Softmax Layer) [13], який є вихідним, може виводити результати, які за своєю суттю є вірогідністю.

**Розробка архітектури нейронної мережі.** З метою вирішення проблеми розпізнавання чисел, розділимо її на декілька частин. По перше, не будемо розпізнавати числа цілком, а будемо ділити їх на розряди (рис. 8).

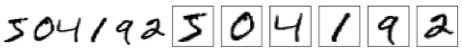


*Рис. 8. Приклад розділення числа на розряди*

Люди вправно вирішують проблему сегментації, але це складно для програм. Щоб розпізнати окремі цифри, необхідно використовувати нейронну мережу з трьома шарами.

Вхідний шар буде містити 784 нейрони, що відповідає зображенню 28 на 28 пікселів, де значення кожного відповідає за яскравість. Наступний шар, прихований, буде мати від 30 до 100 нейронів, будемо експериментувати, щоб отримати кращий результат. Можна міркувати про середній шар як про окремі групки нейронів, які намагаються розпізнати окремі частини цифр (рис. 9).

*Рис. 9. Окремі частини цифр*

Вихідний шар буде мати десять нейронів, які відповідають цифрам від 0 до 9. Один з альтернативних варіантів може мати 4 вихідних нейрони, які будуть кодувати число у двійковій системі числення.

Навчальний набір MNIST складається з 30 000 моделей з NIST`s Special Database 3 та 30 000 моделей з NIST’s Special Database1. Тестовий набір складався з 5000 зображень з SD-3 та 5000 зразків SD-1. Набір навчальних та тренувальних даних у розмірі 60 000 екземплярів цифр містить приклади приблизно 250 письменників.

**Розробка програмного забезпечення.**

При вході у програму та завантаженні нейронної мережі, стає доступним полотно для малювання. Завдання полягає у тому, щоб після того, як користувач натисне кнопку, яка буде опрацьовувати зображення, програма має знайти усі цифри, виділити їх прямокутниками, після чого скопіювати в окремі зображення. Оскільки цифри будуть довільного розміру, по-перше, потрібно буде додати внутрішній відступ, щоб цифри не прилипали до стінок, потім зжати їх до певного розміру і відправити на вхід до нейронної мережі. На рисунку 10 знаходиться блок-схема алгоритму для пошуку та виділення цифр.

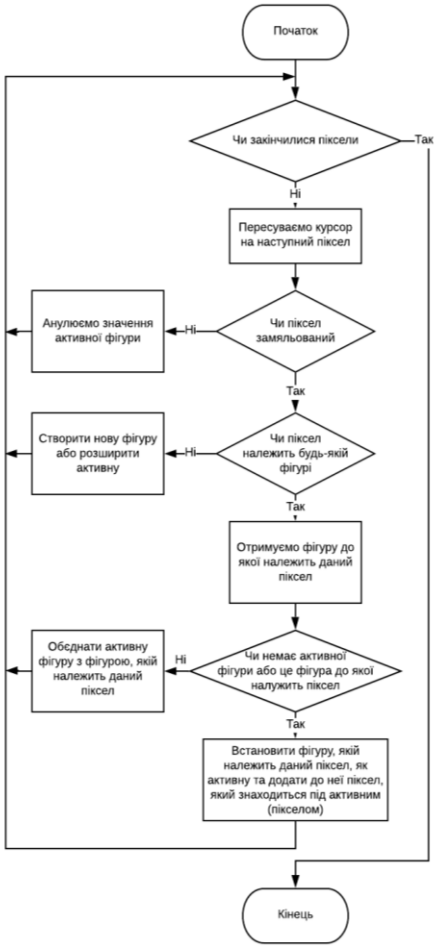



*Рис. 10. Блок-схема алгоритму знаходження фігур*

Три головні завдання, які має виконувати готова програма: створення своїх тренувальних даних, тренування нейронної мережі за допомогою вибраних файлів та заданих параметрів, тестування шляхом малювання цифр та оцінки акуратності.

**Створення нестандартних наборів даних.** Створимо окреме поле, в яке необхідно записати очікувані результати, після чого збережемо. Проблема полягає у тому, що зазвичай у тренувальних наборах тисячі або навіть десятки тисяч позначених даних, і за короткий час неможливо відтворити достатню кількість даних, щоб ефективно тренувати нейронну мережу. Щоб вирішити цю проблему, будемо штучним шляхом розширювати кількість даних. У випадку з картинками це досить просто зробити. Спочатку можна взяти оригінальне зображення цифри та повернути його на деякий кут вліво та вправо, тепер у нас є три екземпляри, які різняться, далі можна стискати, розтягувати, скошувати та додавати іншого роду спотворення. Зрештою можна отримати приблизно 50 копій при тому, що намалювали лише одну цифру. Це дасть вагомий приріст в акуратності після тренування.

**Тренування нейронної мережі.** Для тренування необхідно розробити окрему вкладку, де можна тренувати мережу, а також проглядати миттєвий фідбек того, як добре нейронна мережа справляється зі своєю роботою у вигляді графіків. Має бути можливість одночасного тестування на додаткових наборах даних.

**Розробка бібліотеки для роботи з нейронними мережами.** На рисунку 11 зображено діаграму класів.

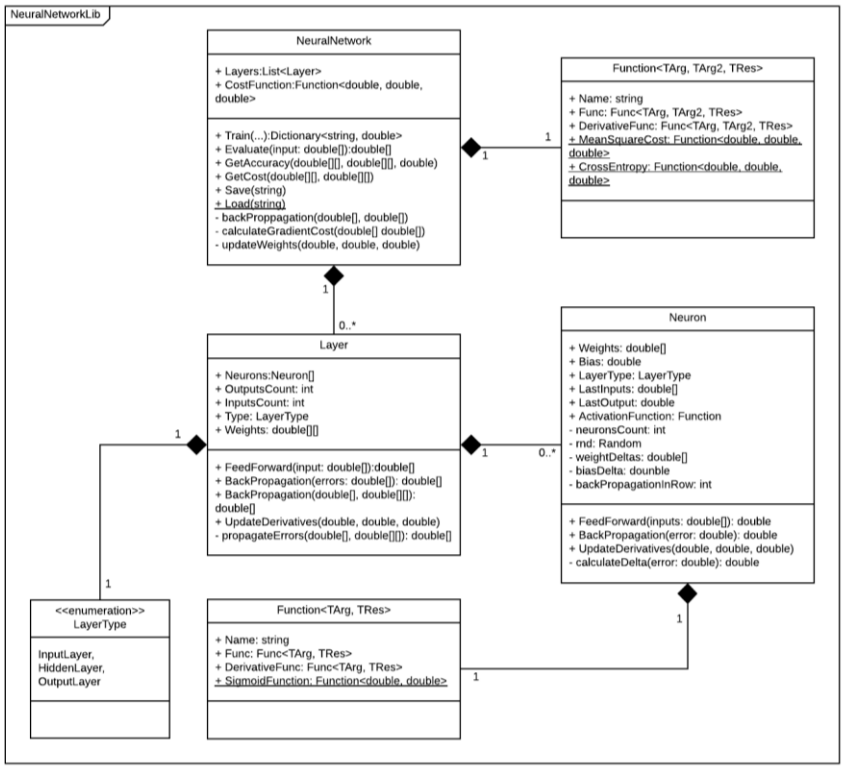



*Рис. 11. UML-діаграма класів*

Клас Neuron є найнижчою функціональною одиницею нейронної мережі. Він зберігає інформацію про ваги, функцію активації та вміщує в собі методи для поширення вихідного результату вперед (FeedForward) або помилки назад (BackPropagation), ці два методи разом з UpdateDerivatices становлять головний функціонал нейронної мережі.

Клас Layer слугує обгорткою для нейронів та проміжною ланкою, яка інкапсулює методи для керування групою нейронів. Має типи: вхідний, прихований та вихідний шар. Також може слугувати базовим класом для інших, вузьконаправлених типів шарів.

Метод FeedForward отримує вектор вхідних значень та, прогнавши його через кожен з нейронів, повертає результуючий вектор шару. До того ж, якщо тип шару вхідний, дані повертаються без опрацювання нейронами.

```
public double[] FeedForward(double[] inputs) {
   if (InputsCount != inputs.Length)
       throw new ArgumentException("Neurons of the layer don't accept the number of given inputs");
         if (Type == LayerType.InputLayer)  return inputs;
         double[] outputs = new double[OutputsCount];
```

```
            for (int i = 0; i < Neurons.Length; i++)
                outputs[i] = Neurons[i].FeedForward(inputs);
            return outputs;
        }
```

Клас бібліотеки NeuralNetwork зберігає шари з нейронами, надає користувачу методи, які інкапсулюють усі попередні класи.

**Опис роботи з програмним додатком**

Головне вікно програмного додатка поділено на дві частини. Ліва відповідає за завантаження, зберігання, створення та відображення головної інформації нейронної мережі. Права призначена для створення або завантаження нейронної мережі. Щоб створити нейронну мережу, спочатку потрібно в полі «Topology» через пробіл записати кількість нейронів на кожному з шарів, яких має бути не менше 3, після чого вибрати функції помилки й активації та натиснути «Create».

Перша з двох доступних вкладок дає можливість протестувати завантажену нейронну мережу. Намалювавши цифру або декілька цифр і натиснувши «Process», з'явиться результат (рис. 12). Результат може бути трьох кольорів: зелений, жовтий, червоний. Від кольору залежить впевненість нейронної мережі у результаті, але це не означає, що він є правильним. У цьому вікні можна знайти додаткові можливості, наприклад, обведення знайдених цифр прямокутником (слугує для налагодження) або класифікації результатів. Можна ввести правильні відповіді (без пробілів) у відповідне поле, що з'явиться після того, як поставити відповідний прапорець. Далі, якщо натиснути «опрацювати знову», то побачимо, що показник класифікованих даних збільшився, а також активується кнопка для збереження цих даних. Можна використати класифіковані дані потім, коли потрібно тренувати нейронну мережу.

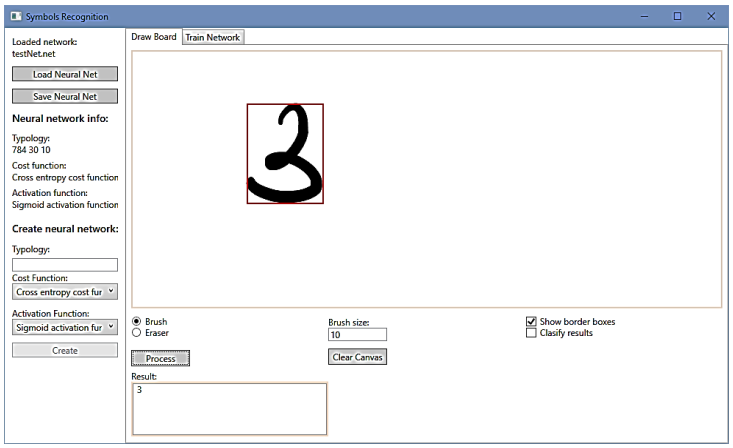



*Рис. 12. Виведення меж цифр*

На вкладці тренування нейронної мережі (рис. 13) можна знайти багато параметрів, які впливають на результати. Найменше, що потрібно зробити для того, щоб почати тренування, це завантажити тренувальні файли з вхідними даними та ярликами до них, після цього активується кнопка «Train». Є можливість вибрати кількість даних, які потрібно завантажити, або зазначити «all», якщо потрібні усі дані, кількість поколінь.

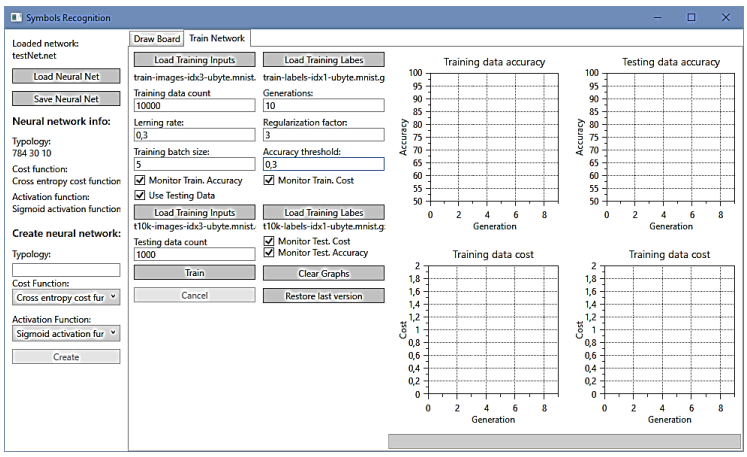
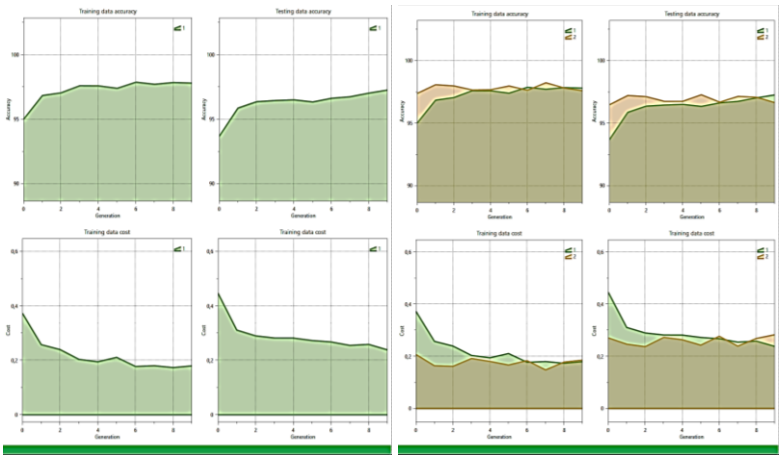



*Рис. 13. Вкладка тренування нейронної мережі*

Далі коефіцієнт швидкості тренування, коефіцієнт регулювання, кількість екземплярів даних у порції, поріг акуратності, прапорці активації графіків. За бажанням можна завантажити дані для тестів та визначити їх кількість.

Після початку тренування інформація буде з'являтися на графіках (рис. 14, *a*), якщо поставити відповідні прапорці, зображення буде автоматично масштабуватися відповідно до найнижчих та найвищих значень. Запустивши тренування декілька разів, можна побачити, що графіки не стираються, а лягають один на одного (рис. 14, *b*), що дуже зручно для порівняння. Для того, щоб очистити графіки, потрібно натиснути «Clear Graphs». Під час тренування стає активною кнопка «Cancel», яка відміняє тренування та повертає попередню версію нейронної мережі. Також є можливість повернутися навіть після того, як тренування завершилося – кнопка «Restore last version» (така функція повертає тільки на один крок назад).



*a* *b*

*Рис. 14: a) виведення результатів тренування; b) порівняння результатів тестування*

**Висновки та перспективи подальших досліджень.** Отже, штучні нейронні мережі досягли широкого розповсюдження. Вони можуть обрахувати безперервну функцію будь-якої складності з достатнім наближенням, що робить їх надзвичайно гнучкими для застосування у всіх сферах життя. За останні декілька десятиліть були розроблені підходи до тренування та застосування нейронних мереж, але багато з них досліджені емпіричним методом, іноді підібрати правильні параметри можна тільки експериментально. Це свідчить про те, що ця область науки недостатньо вивчена.

У цій статті були досліджені різні види нейронних мереж та розроблена бібліотека для роботи з ними, яка разом з набором тренувальних даних MNIST дала акуратність близько 98 %. У будь-якому випадку методи, які розглядалися у цій роботі, є основою для побудови більш складних систем.

Додатком до бібліотеки створено програмний продукт на фреймворці WPF, який надає можливість зберігати, тренувати та тестувати нейронну мережу.

**Список використаної літератури:**


1. *Neural M.* Networks and Deep Learning / *M.Nielsen* // Determination Press. – 2015. – P. 224.
2. *Neural M.* Neural Networks and Deep Learning / *M.Neural* [Електронний ресурс]. – Режим доступу : http://neuralnetworksanddeeplearning.com.
3. Optical character recognition [Електронний ресурс]. – Режим доступу : https://en.wikipedia.org/wiki/Optical_character_recognition.
4. Artificial neural network [Електронний ресурс]. – Режим доступу : https://en.wikipedia.org/wiki/Artificial_neural_network.
5. Machine learning [Електронний ресурс]. – Режим доступу : https://en.wikipedia.org/wiki/Machine_learning.
6. *LeCun Y.* The mnist database / *Y.LeCun* [Електронний ресурс]. – Режим доступу : http://yann.lecun.com/exdb/mnist/.
7. EMNIST: an extension of MNIST to handwritten letters [Електронний ресурс]. – Режим доступу : https://arxiv.org/pdf/1702.05373.pdf.
8. *Khaustov P.A.* Algorithms for handwritten character recognition based on constructing structural models / *P.A. Khaustov* // Computer Optics. – 2017. – № 41 (1). – P. 67–78.
9. *Моругов А.М.* Методы распознавания символов / *А.М. Моругов, С.В. Волков* // Труды Международного симпозиума «Надежность и качество». – Пенза : ПГУ, 2017. – Т. 1. [Електронний ресурс]. – Режим доступу : https://cyberleninka.ru/article/n/metody-raspoznavaniya-simvolov/viewer.
10. Perceptron [Електронний ресурс]. – Режим доступу : https://en.wikipedia.org/wiki/Perceptron.

11. Deep learning [Електронний ресурс]. – Режим доступу : https://en.wikipedia.org/wiki/Deep_learning.
12. *Upadhyay Y.* Introduction to Feed Forward Neural Networks / *Y.Upadhyay*. – 2019 [Електронний ресурс]. – Режим доступу : https://en.wikipedia.org/wiki/Perceptron.
13. *Lan H.* The Softmax Function, Neural Net Outputs as Probabilities, and Ensemble Classifiers / *H.Lan*. – 2017 [Електронний ресурс]. – Режим доступу : https://towardsdatascience.com/the-softmax-function-neural-net-outputs-as-probabilities-and-ensemble-classifiers-9bd94d75932.

**References:**

1. Neural, Michael (2015), «Networks and Deep Learning», *Determination Press APK*, 224 p.
2. Neural, M. «*Neural Networks and Deep Learning*», [Online], available at: http://neuralnetworksanddeeplearning.com
3. Optical character recognition, [Online], available at: https://en.wikipedia.org/wiki/Optical_character_recognition
4. Artificial neural network, [Online], available at: https://en.wikipedia.org/wiki/Artificial_neural_network
5. Machine learning, [Online], available at: https://en.wikipedia.org/wiki/Machine_learning
6. LeCun, Y., *The mnist database*, [Online], available at: http://yann.lecun.com/exdb/mnist/
7. EMNIST: an extension of MNIST to handwritten letters, [Online], available at: https://arxiv.org/pdf/1702.05373.pdf
8. Khaustov, P.A. (2017), «Algorithms for handwritten character recognition based on constructing structural models», *Computer Optics*, No. 41 (1), pp. 67–78.
9. Morugov, A.M. and Volkov, S.V. (2017), «Character Recognition Methods», *Proceedings of the International Symposium «Reliability and quality»*, PSU, Penza, Vol. 1, [Online], available at: https://cyberleninka.ru/article/n/metody-raspoznavaniya-simvolov/viewer
10. Perceptron, [Online], available at: https://en.wikipedia.org/wiki/Perceptron
11. Deep learning, [Online], available at: https://en.wikipedia.org/wiki/Deep_learning
12. Upadhyay, Y. (2019), *Introduction to FeedForward Neural Networks*, [Online], available at: https://en.wikipedia.org/wiki/Perceptron
13. Haihan Lan (2017), *The Softmax Function, Neural Net Outputs as Probabilities, and Ensemble Classifiers*, [Online], available at: https://towardsdatascience.com/the-softmax-function-neural-net-outputs-as-probabilities-and-ensemble-classifiers-9bd94d75932

**Коротун** Ольга Володимирівна – доцент кафедри комп'ютерних наук, кандидат педагогічних наук, заступник декана факультету інформаційно-комп'ютерних технологій Державного університету «Житомирська політехніка».

Наукові інтереси:
- бази даних;
- теорія складності алгоритмів та обчислень;
- чисельні методи.

https://orcid.org/0000-0003-2240-7891.

**Марчук** Галина Вікторівна – старший викладач кафедри комп'ютерних наук Державного університету «Житомирська політехніка».

Наукові інтереси:
– інтелектуальний аналіз даних;
– комп'ютерна дискретна математика.

https://orcid.org/0000-0003-2954-1057.

**Марчук Дмитро Костянтинович** – асистент кафедри інженерії програмного забезпечення Державного університету «Житомирська політехніка»

Наукові інтереси:
– інтелектуальний аналіз даних
– розробка ігор

**Талавер Олег Володимирович –** студент факультету інформаційно комп'ютерних технологій Державного університету «Житомирська політехніка»

Наукові інтереси:
– інженерія програмного забезпечення

Стаття надійшла до редакції 24.03.2020.