МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
імені В. Н. КАРАЗІНА

ШТУЧНІ НЕЙРОННІ МЕРЕЖІ

Методичні вказівки
до виконання практичних завдань та
самостійного вивчення теми «Штучні нейронні мережі» з дисципліни
«Алгоритми аналізу даних та штучні нейронні мережі»
для здобувачів вищої освіти другого (магістерського) рівня
за спеціальністю F1 «Прикладна математика»

Електронний ресурс

Харків – 2026

УДК 004.8
Ш 94

Рецензенти:

Є. В. Петров – кандидат фізико-математичних наук, доцент кафедри фундаментальної математики факультету математики і інформатики Харківського національного університету імені В. Н. Каразіна;

Ю. В. Ромашов – доктор технічних наук, професор кафедри комп'ютерно-інтегрованих технологій, автоматизації та робототехніки Харківського національного університету радіоелектроники.

Затверджено до розміщення в мережі Інтернет рішенням Науково-методичної ради Харківського національного університету імені В. Н. Каразіна (протокол № 8 від 21 травня 2026 року)

Ш 94 **Штучні** нейронні мережі : методичні вказівки до виконання практичних завдань та самостійного вивчення теми «Штучні нейронні мережі» з дисципліни «Алгоритми аналізу даних та штучні нейронні мережі» для здобувачів вищої освіти другого (магістерського) рівня за спеціальністю F1 «Прикладна математика» [Електронний ресурс] / уклад. В. В. Карєва. – Харків : ХНУ імені В. Н. Каразіна, 2026. – (PDF 41 с.)

Методичні вказівки до вивчення дисципліни розроблено для здобувачів 1 курсу другого (магістерського) рівня вищої освіти факультету математики і інформатики за спеціальністю F1 «Прикладна математика».

Методичні рекомендації містять поради з підготовки до практичних занять і організації самостійної роботи з теми «Штучні нейронні мережі» з дисципліни «Алгоритми аналізу даних та штучні нейронні мережі». Наведені матеріали ознайомлюють здобувачів з методами та алгоритмами нейронних мереж на початковому рівні.

УДК 004.8

ЗМІСТ

ВСТУП

Машинне навчання та штучний інтелект упродовж останніх років стали одними з найбільш актуальних напрямів розвитку сучасних інформаційних технологій. У різних формах вони вже інтегровані в значну кількість програмних продуктів і сервісів, а їх застосування продовжує активно розширюватися. Приклади використання машинного навчання включають автоматичне визначення важливості електронних листів, генерацію відповідей, створення музики, а також складні інтелектуальні системи, такі як AlphaGo.

Глибоке машинне навчання є підрозділом штучного інтелекту, що базується на застосуванні статистичних методів і багатошарових нейронних мереж для побудови моделей, здатних навчатися на даних і покращувати свою продуктивність під час виконання конкретних завдань. Предметом вивчення навчальної дисципліни «Алгоритми аналізу даних та штучні нейронні мережі» є методи класифікації, кластеризації, прогнозування, виявлення аномалій на основі сучасних моделей нейронних мереж, а також розробка програмного забезпечення з використанням цих методів.

У межах дисципліни студенти опановують підходи до побудови аналітичних моделей і алгоритмів, що застосовуються для розв’язання прикладних задач. Такі моделі дозволяють отримувати надійні та відтворювані результати, а також виявляти приховані закономірності у даних на основі аналізу історичних залежностей і тенденцій.

Метою викладання дисципліни є підготовка фахівців, здатних застосовувати сучасні методи глибокого машинного навчання для розв’язання прикладних задач у різних галузях, зокрема в робототехніці, медицині, інформаційному пошуку та інших.

Здобувачам вищої освіти другого (магістерського) рівня за спеціальністю F1 «Прикладна математика» у межах вивчення дисципліни «Алгоритми аналізу даних та штучні нейронні мережі» пропонуються практично орієнтовані завдання, спрямовані на формування сучасних компетентностей у галузі штучного інтелекту.

Особливу увагу приділено застосуванню методів глибокого навчання для розв’язання прикладних задач, зокрема обробки зображень, аналізу текстових даних, прогнозування та побудови інтелектуальних систем. Розглядаються сучасні підходи до побудови нейронних мереж, їх навчання та оптимізації із використанням сучасних програмних засобів.

Важливим складником освітнього процесу є самостійна робота здобувачів освіти, яка включає опрацювання теоретичного матеріалу, виконання практичних завдань, реалізацію програмних проєктів та аналіз отриманих результатів. Така діяльність сприяє формуванню як загальних, так і фахових компетентностей, необхідних для подальшої професійної діяльності.

Для ефективної організації навчання здобувачам рекомендується постійно взаємодіяти з викладачем, своєчасно виконувати завдання,

використовувати навчальні матеріали та сучасні онлайн-платформи. Значну роль відіграють практичні та проєктні форми роботи, які сприяють розвитку аналітичного мислення та навичок командної роботи.

Здобувачі освіти повинні дотримуватися принципів академічної доброчесності, зокрема уникати плагіату, забезпечувати коректне цитування джерел та самостійність виконання завдань. Використання допоміжних інструментів, у тому числі систем штучного інтелекту, має здійснюватися з дотриманням встановлених вимог та етичних норм.

Основними завданнями вивчення дисципліни є:

– засвоєння теоретичних основ глибокого машинного навчання;

– вивчення сучасних алгоритмів навчання нейронних мереж;

– формування навичок побудови та аналізу моделей для розв’язання прикладних задач;

– набуття практичних умінь реалізації алгоритмів машинного навчання.

У результаті вивчення дисципліни студент повинен:

знати:

– основні поняття, підходи, методи та проблеми глибокого машинного навчання;

– задачі класифікації, регресії та кластеризації;

– основні моделі нейронних мереж;

– принципи навчання з підкріпленням;

вміти:

– реалізовувати алгоритми глибокого навчання та оцінювати їх ефективність;

– будувати та навчати нейронні мережі різних типів (зокрема згорткові та рекурентні);

– застосовувати методи машинного навчання для розв’язання практичних задач.

Методичні вказівки розроблені відповідно до вимог підготовки фахівців з вищою освітою та містять рекомендації щодо організації навчального процесу, підготовки до практичних занять і самостійної роботи студентів. Подані матеріали сприяють формуванню системного розуміння сучасних підходів у галузі штучного інтелекту.

Курс побудований на основі класичної методики викладання базових дисциплін, яка передбачає поєднання послідовного викладу теоретичної частини на лекціях з подальшим закріпленням і поглибленням знань, поряд з набуттям практичних навичок і умінь на практичних заняттях.

Курс передбачає необхідність систематичного опанування студентами досить складного теоретичного матеріалу і набуття навичок вирішення окремих завдань і проблем. Це зумовлює значну питому вагу самостійної роботи студентів, включаючи вивчення необхідної літератури, вміння осмислити існуючі підходи до досліджуваної проблеми, сформувати власну точку зору і застосувати отримані знання для аналізу конкретних практичних ситуацій.

1. ОСОБЛИВОСТІ ОРГАНІЗАЦІЇ ПРАКТИЧНИХ ЗАНЯТЬ. КРИТЕРІЇ ОЦІНЮВАННЯ.

У вивченні дисципліни «Алгоритми аналізу даних та штучні нейронні мережі» важливу роль відіграють практичні заняття, які сприяють систематизації, поглибленню та узагальненню теоретичних знань, формуванню навичок самостійного аналізу задач машинного навчання, побудови моделей нейронних мереж та інтерпретації отриманих результатів.

Практичні заняття проводяться після опрацювання відповідного теоретичного матеріалу та передбачають виконання задач прикладного характеру із використанням сучасних програмних засобів (зокрема Python, TensorFlow, Keras тощо). У процесі підготовки до занять здобувачі освіти повинні ознайомитися з лекційним матеріалом, опрацювати рекомендовані джерела та підготуватися до виконання практичних завдань.

Практичне заняття, як правило, включає:

– розгляд ключових теоретичних положень;

– виконання практичних завдань;

– аналіз отриманих результатів;

– обговорення проблемних питань.

Оцінювання навчальної діяльності здобувачів здійснюється на основі їх роботи під час практичних занять, виконання індивідуальних завдань та підсумкового контролю. Під час занять оцінюється:

– правильність виконання практичних завдань;

– розуміння теоретичних основ;

– вміння застосовувати алгоритми машинного навчання;

– активність під час обговорення;

– якість програмної реалізації.

Практичні заняття є обов’язковими до відвідування. У разі пропуску заняття або отримання незадовільної оцінки здобувач повинен відпрацювати матеріал у встановлені терміни під час консультацій або із використанням дистанційних освітніх платформ.

Підсумкова оцінка формується на основі накопичувальної системи та враховує результати роботи на практичних заняттях, виконання індивідуальних завдань і результати підсумкового контролю.

Критерії оцінювання практичних занять:

Оцінювання результатів навчальної діяльності здобувачів під час практичних занять здійснюється за бінарною системою: «зараховано» (1 бал) або «не зараховано» (0 балів).

Оцінка «зараховано» виставляється у разі, якщо здобувач освіти виконав практичне завдання у повному обсязі, продемонстрував розуміння теоретичного матеріалу, коректно реалізував алгоритм та отримав обґрунтовані результати.

Оцінка «не зараховано» виставляється у випадку невиконання завдання, наявності суттєвих помилок у реалізації або відсутності розуміння основних принципів розв’язання задачі.

Підсумкова оцінка за практичні заняття визначається як середнє значення отриманих балів за всі виконані практичні роботи та враховується при формуванні загального рейтингового балу з дисципліни. Отримане значення масштабується відповідно до максимальної кількості балів, передбаченої робочою програмою дисципліни.

2. ПРАКТИЧНІ ЗАНЯТТЯ З ДИСЦИПЛІНИ.

Під час вивчення курсу використовуються різні форми навчальних занять: лекції та практичні заняття. Основна мета лекцій – сформувати у студентів уявлення про ключові поняття, закономірності та сучасні підходи у галузі штучного інтелекту та машинного навчання.

Лекції задають напрям подальшої самостійної роботи студентів, включаючи опрацювання навчальної літератури та додаткових джерел.

Підготовка до практичних занять передбачає ознайомлення з лекційним матеріалом, опрацювання теоретичних відомостей, засвоєння основних понять і алгоритмів. Практичні заняття спрямовані на закріплення теоретичних знань та формування навичок розв’язання прикладних задач.

Далі перейдемо до розгляду базових теоретичних моделей штучних нейронних мереж, які лежать в основі сучасних методів машинного навчання.

2.1. Персептрон Розенблатта.

Штучний інтелект (Artificial Intelligence, AI) – галузь науки, що досліджує методи створення інтелектуальних систем, здатних виконувати завдання, які традиційно потребують людського мислення.

Машинне навчання (Machine Learning, ML) – підрозділ штучного інтелекту, що вивчає методи побудови алгоритмів, здатних навчатися на основі даних.

Штучні нейронні мережі (Artificial Neural Networks, ANN) – спрощені математичні моделі біологічних нейронних мереж, призначені для розв’язання задач класифікації, регресії, прогнозування та розпізнавання образів.

Модель МакКаллока-Піттса

Розглянемо задачу навчання з учителем.

Нехай $X -$ простір об'єктів;

$Y -$ множина допустимих відповідей;

$y^*: X \to Y -$ цільова залежність, відома лише на об'єктах навчальної вибірки

$X^l = (x_i, y_i)_{i=1}^l,\ y_i = y^*(x_i).$

Задача навчання полягає у побудові алгоритму $a: X \to Y$, що апроксимує цільову залежність y^* на всій множині X.

Припускатимемо, що об'єкти описуються n числовими ознаками $f_j: X \to \mathbb{R}, j = 1,..,n$. Вектор $(f_1(x), \ldots, f_n(x)) \in \mathbb{R}^n$ називається *ознаковим описом* об'єкта x.

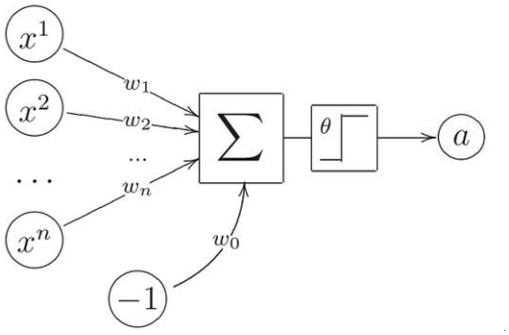

Рис. 1. Модель МакКаллока-Піттса

Модель нейрона МакКаллока–Піттса має вигляд:

$$a(x) = \varphi\left(\sum_{j=1}^{n} \omega_j x^j - \omega_0\right),$$

де $\varphi(z) = [z \geq 0]$ – ступінчаста функція Хевісайда. У теорії нейронних мереж функцію φ, що перетворює значення сумарного імпульсу у вихідне значення нейрона, прийнято називати *функцією активації*.

Введемо додаткову константну ознаку $x^0 \equiv -1$ і векторні позначення $\omega = (\omega_0, \omega_1, \ldots, \omega_n)^T$, $x = (x^0, x^1, \ldots, x^n)^T$. Тоді лінійну модель нейрона можна записати компактніше через скалярний добуток:

$$a(x) = \varphi\left(\sum_{j=0}^{n} \omega_j x^j\right) = \varphi(< \omega, x >).$$

Пізніше модель МакКаллока-Піттса була узагальнена на випадок довільних дійсних входів і виходів, і довільних функцій активації.

Персептрон Розенблатта

Персептрон Розенблатта є однією з перших моделей штучного нейрона, навчання якої здійснюється шляхом коригування вагових коефіцієнтів на основі похибки класифікації.

Алгоритм навчання персептрона Розенблатта.

Вхід:

X^l – навчальна вибірка;

η – темп навчання;

Вихід:

синаптичні ваги $\omega_0, \omega_1, \ldots, \omega_n$;

Кроки алгоритму:

1. Ініціалізувати ваги ω_j;

2. *Повторювати*
   Для всіх $i = 1, \dots, l$
   $\omega := \omega - \eta(a(x_i) - y_i)x_i$;
3. Продовжувати навчання, *поки* ваги ω змінюються.

Завдання. Реалізувати алгоритм персептрона Розенблатта програмними засобами та дослідити його роботу на простих наборах даних.

2.2. Метод стохастичного градієнта SG.

Більшість задач машинного навчання зводяться до задачі оптимізації, яка полягає у знаходженні такого вектора параметрів, що мінімізує функцію втрат:

$$\omega = \underset{\omega}{\operatorname{argmin}} \mathcal{L}(\omega, X, y)$$

Існує два основних підходи до розв'язання задач оптимізації:
– аналітичний розв'язок;
– ітераційні методи, зокрема градієнтний спуск.

Аналітичне рішення має найбільший недолік у тому, що не для будь-якого $\mathcal{L}$ воно існує, або може бути виписано аналітично, тому на практиці широко застосовуються чисельні методи оптимізації. Одним із найбільш ефективних підходів є стохастичний градієнтний спуск (Stochastic Gradient Descent, SGD). Однак у стохастичного градієнтного спуску є ряд обмежень, які потрібно задовольнити, щоб була збіжність.

З принципу мінімізації емпіричного ризику випливає, що задача налаштування синаптичних ваг може бути зведена до пошуку вектора ω, на якому досягається мінімум функціоналу якості:

$$Q(\omega) = \sum_{i=1}^{l} \mathcal{L}(a(x_i), y_i) \to \min_{\omega}.$$

Оновлення вагових коефіцієнтів у методі градієнтного спуску виконується за правилом:

$$\omega := \omega - \eta \frac{\partial Q}{\partial \omega}$$

У стохастичному варіанті градієнт оцінюється за одним (або невеликою підмножиною) об'єктом:

$$\omega := \omega - \eta \sum_{i=1}^{l} \mathcal{L}'_a(a(x_i), y_i)\varphi'(\langle \omega, x_i \rangle)\, x_i\,.$$

Алгоритм. Метод стохастичного градієнта.
Вхід:
X^l – навчальна вибірка;
η – темп навчання;
Вихід:

синаптичні ваги $\omega_0, \omega_1, \dots, \omega_n$;

Кроки алгоритму:

1. Ініціалізувати ваги ω_j: $\omega_j = random\left(-\frac{1}{2n}, \frac{1}{2n}\right)$;
2. Ініціалізувати оцінку функціоналу :$Q(\omega) = \sum_{i=1}^{l} \mathcal{L}(a(x_i), y_i)$;
3. *Повторювати*

   Випадково обрати $i = 1, \dots, l$, отримати об'єкт x_i;
   Обчислити $a(x_i)$ та помилку: $\varepsilon_i = \mathcal{L}(a(x_i), y_i)$;
   Зробити крок градієнтного спуску
   $\omega := \omega - \eta \mathcal{L}'_a(a(x_i), y_i)\varphi'(\langle \omega, x_i \rangle)x_i$;
   Оцінити значення функціоналу : $Q := \frac{l-1}{l} Q + \frac{1}{l} {\varepsilon_i}^2$;
4. Продовжувати, *поки* значення Q не стабілізується.

Приклад (Адаптивний лінійний елемент).

Нехай всі ознаки дійсні: $X = \mathbb{R}^n$, функція втрат квадратична: $\mathcal{L}(a, y) = (a - y)^2$, функція активації лінійна: $\varphi(z) = z$. Тоді правило оновлення ваг у методі стохастичного градієнта збігається з правилом Розенблатта:

$$\omega := \omega - \eta(a(x_i) - y_i)x_i.$$

Це правило навчання було запропоновано Відроу та Хоффом у 1960 році та називається дельта-правилом (delta-rule), а сам лінійний нейрон – адаптивним лінійним елементом або ADALINE.

Завдання.

1. Реалізувати алгоритм стохастичного градієнтного спуску.
2. Ознайомитися з реалізаціями SGD у сучасних бібліотеках:

– tf.keras.optimizers;
– https://keras.io/api/optimizers/sgd/;
– https://scikit-learn.org/stable/modules/sgd.html.

2.3. Прості нейронні мережі.

Прості нейронні мережі є базовими моделями глибокого навчання, які використовуються для апроксимації функцій, класифікації даних та моделювання логічних операцій. Одним із класичних прикладів задачі, що демонструє можливості багатошарових нейронних мереж, є реалізація логічної функції «виняткове АБО» (XOR), яка не може бути коректно подана одношаровим персептроном (рис. 2).

Навчання багатошарових нейронних мереж, як правило, здійснюється за допомогою алгоритму зворотного розповсюдження помилки (Backpropagation), який дозволяє коригувати вагові коефіцієнти мережі на основі величини похибки на виході.

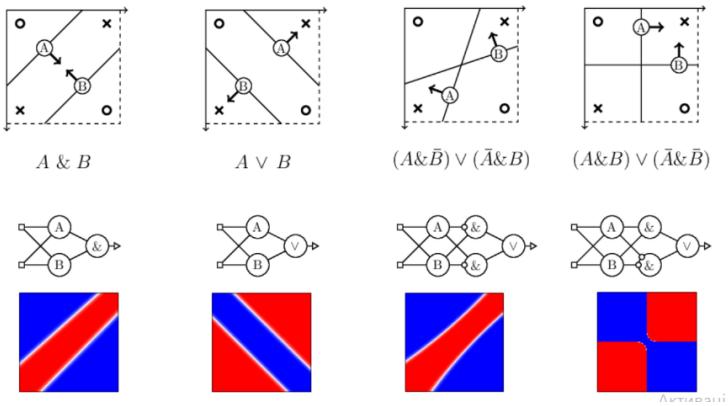

Рис. 2. Виняткове АБО (XOR)

Алгоритм зворотного розповсюдження помилки (BackProp).

Вхід: вибірка X^l, архітектура$(H_l)_{l=1}^L$, параметри η, λ;

Вихід: вектор ваг $\omega = (W^0, W^1, .., W^L)$ всіх шарів;

Кроки алгоритму:

1. Ініціалізувати всі ваги;
2. *Повторювати*

   випадково обрати об'єкт $x_i \in X^l$;

   прямий хід: для всіх $l = 1, .., L, h = 1, .., H_l$

$$\Sigma_{ih}^l := \sum_{k=0}^{H_{l-1}} \omega_{kh}^l \, x_{ik}^{l-1}; x_{ih}^l := \varphi_h^l(\Sigma_{ih}^l); s_{ih}^l := (\varphi_h^l)' \, \Sigma_{ih}^l;$$

$$\varepsilon_{ih}^L := \frac{\partial \mathcal{L}_i(\omega)}{\partial x_h^L}, h = 1, .., H_L$$

   зворотний хід: для всіх $l = L, .., 2, k = 0, .., H_{l-1}$

$$\varepsilon_{ik}^{l-1} = \sum_{h=0}^{H_l} \varepsilon_{ih}^l s_{ih}^l \, \omega_{kh}^l$$

   градієнтний крок: для всіх $l = 1, .., L, k = 0, .., H_{l-1}, h = 1, .., H_l$

$$\omega_{kh} := \omega_{kh} - \eta \varepsilon_{ih}^l s_{ih}^l x_{ik}^{l-1};$$

3. Продовжувати навчання, *поки* значення Q та/або ваги ω не зійдуться.

Побудова нейронної мережі.

Розглянемо приклад побудови простої нейронної мережі засобами бібліотеки Keras.

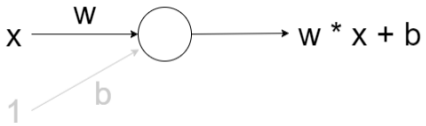

Рис. 3. Проста нейронна мережа

Лістинг 1 – Приклад простої нейронної мережі в Keras.

```
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import SGD, Adam
```

```
model = Sequential()
model.add(Dense(50, input_dim=1, activation='tanh',
bias_initializer='glorot_normal'))
model.add(Dense(1, input_dim=50, activation='relu'))
sgd=Adam(lr=0.001, clipnorm=0.001)
model.compile(loss='mean_squared_error', optimizer=sgd)
model.fit(x, y, epochs=1000, verbose = 0)

plt.scatter(x, y, color='black', antialiased=True)
plt.plot(x, model.predict(x), color='red', linewidth=2,
antialiased=True)
plt.show()
```

У наведеному прикладі створюється двошарова нейронна мережа для апроксимації функції. Перший шар містить 50 нейронів з функцією активації tanh, а вихідний шар – один нейрон з функцією активації relu. Для навчання використовується оптимізатор Adam.

Завдання.

1. Налаштувати параметри нейронної мережі (кількість нейронів, кількість шарів, функції активації, метод оптимізації тощо) для апроксимації заданої функції.
2. Реалізувати логічні функції І, АБО, виняткове АБО засобами бібліотеки Keras.
3. Порівняти результати роботи мережі при різних налаштуваннях параметрів.

Додаткове завдання (дослідження параметрів нейронної мережі).

1. Ознайомитися з базовим скриптом Regression, який реалізує навчання багатошарової нейронної мережі (MLP) для штучно згенерованої задачі регресії.
2. Запустити навчання моделі з різними оптимізаторами: SGD; RMSProp; Adam.

Зафіксувати значення метрик:

– середньоквадратична помилка (MSE);

– середня абсолютна помилка (MAE);

– коефіцієнт детермінації (R^2).

3. Провести серію експериментів, змінюючи параметри моделі:

– кількість шарів (depth = 1, 3, 5);

– кількість нейронів у шарі (width = 64, 128, 256);

– наявність або відсутність Batch Normalization;

– наявність або відсутність Dropout;

– коефіцієнт Dropout (0.1, 0.3, 0.5).

4. Порівняти отримані результати:

– проаналізувати зміну метрик при використанні різних оптимізаторів;

– оцінити вплив Batch Normalization та Dropout на узагальнюючу здатність моделі (за значенням R^2 на тестовій вибірці).

5. Зробити висновки щодо впливу архітектури мережі та параметрів навчання на якість моделі.

2.4. Розпізнавання зображень.

Розпізнавання зображень є однією з ключових задач сучасного машинного навчання та комп'ютерного зору. Людина здатна розпізнавати об'єкти незалежно від їх положення, масштабу, кута огляду або незначних спотворень. Подібні властивості намагаються відтворити за допомогою нейронних мереж.

Класифікація зображень елементів одягу (Fashion MNIST).

У даному прикладі використовується набір даних Fashion MNIST, який містить 70 000 зображень елементів одягу, представлених у відтінках сірого та розподілених на 10 класів.

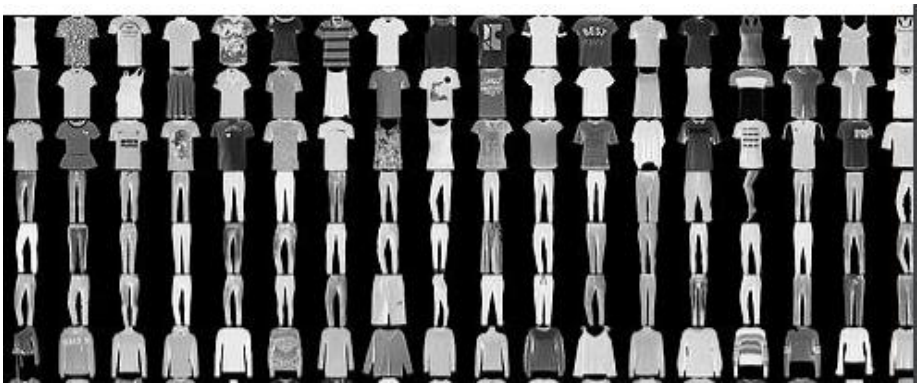

Рис. 4. Fashion MNIST.

Набір даних Fashion MNIST запропоновано як сучасну альтернативу класичному набору MNIST, який традиційно використовується як базовий приклад у задачах машинного навчання та комп'ютерного зору. MNIST містить зображення рукописних цифр у форматі, аналогічному до представлення зображень одягу у Fashion MNIST.

Ми скористаємося 60 000 зображеннями для тренування мережі та 10000 зображеннями для перевірки точності навчання та класифікації зображень.

Завантажуючи набір даних, ми отримуємо метадані, навчальний набір даних і тестовий набір даних.

- ✓ Модель навчається на наборі даних із `train_dataset`
- ✓ Модель оцінюється на наборі даних із `test_dataset`

Перед навчанням виконується попередня обробка даних:

– нормалізація значень пікселів (поділ на 255);

– перетворення міток у формат one-hot encoding;

– приведення даних до необхідної розмірності.

Побудова нейронної мережі потребує налаштування шарів, а потім збирання моделі з функціями оптимізації та втрат.

Базовим структурним елементом нейронної мережі є шар. Кожен шар приймає на вхід певне подання даних і перетворює його у нове подання. Послідовне поєднання кількох шарів дозволяє отримати представлення, придатне для розв'язання поставленої задачі.

Під час побудови моделей глибокого навчання основна увага приділяється конструюванню архітектури мережі, тобто визначенню зв'язків між шарами. Більшість шарів, зокрема таких як tf.keras.layers.Dense, містять параметри, які налаштовуються в процесі навчання моделі.

Архітектура нейронної мережі.

Базова модель складається з трьох шарів:

- ✓ вхідного Flatten – цей шар перетворює зображення розміром 28x28 пікселів у 1D-масив розміром 784 (28*28). На цьому шарі ми не маємо жодних параметрів для навчання, оскільки цей шар займається лише перетворенням вхідних даних.
- ✓ прихований шар Dense – щільний шар зі 128 нейронів. Кожен нейрон (вузол) приймає на вхід усі 784 значення з попереднього шару, змінює вхідні значення згідно з внутрішніми вагами та зсувами під час тренування та повертає єдине значення на наступний шар.
- ✓ вихідний шар Dense – softmax-шар, що складається з 10 нейронів, кожен із яких відповідає певному класу елемента одягу. Як і в попередньому шарі, кожен нейрон приймає на вхід значення всіх 128 нейронів попереднього шару. Параметри шару (ваги та зсуви) налаштовуються в процесі навчання таким чином, щоб вихідні значення належали інтервалу [0,1] і інтерпретувалися як ймовірності належності зображення до відповідних класів. Сума вихідних значень усіх 10 нейронів дорівнює 1.

Перед тим, як тренувати модель, варто ще виконати кілька налаштувань. Ці налаштування виконуються під час складання моделі при виклику методу compile:

- ✓ функція втрат – алгоритм вимірювання того, наскільки далеко знаходиться бажане значення від спрогнозованого.
- ✓ функція оптимізації – алгоритм «підлаштування» внутрішніх параметрів (ваг та зсувів) моделі для мінімізації функції втрат;
- ✓ метрики – використовуються для моніторингу процесу тренування та тестування.

Тренування відбувається за допомогою методу model.fit:

- ✓ Надсилає train_dataset на вхід моделі.
- ✓ Модель навчається зіставляти вхідне зображення з міткою.
- ✓ Параметр epochs = 5 обмежує кількість тренувань до 5 повних навчальних ітерацій з набору даних, що у результаті дає нам тренування на 5 * 60 000 = 300 000 прикладах.

Лістинг 2 – Приклад класифікації зображень (MLP та CNN).

```
from keras.models import Sequential
```

```
from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
class_names = ['Футболка/топ', 'Шорти', 'Светр','Плаття','Плащ',
'Сандалі', 'Сорочка', 'Кросівок','Сумка', 'Черевик']
x_train = x_train / 255
x_test = x_test / 255

y_train_cat = tf.keras.utils.to_categorical(y_train, 10)
y_test_cat = tf.keras.utils.to_categorical(y_test, 10)
x_train = np.expand_dims(x_train, axis=3)
x_test = np.expand_dims(x_test, axis=3)
```

```
#Модель повнозв'язної нейромережі
model = Sequential([
    Flatten(input_shape = (28,28,1)),
    Dense(100,activation = 'relu'),
    Dense(10,activation = 'softmax')
    ])

model.compile(optimizer = 'adam',
              loss = 'categorical_crossentropy',
              metrics = ['accuracy'])
model.fit(x_train,y_train_cat, epochs = 5, batch_size = 32)
```

```
#Модель згорткової нейромережі (CNN)
model = Sequential([
    Conv2D(32,(3,3),padding='same', activation = 'relu',
input_shape = (28,28,1)),
    MaxPooling2D((2,2),strides=2),
    Conv2D(32,(3,3),padding='same', activation = 'relu'),
    MaxPooling2D((2,2),strides=2),
    Flatten(),
    Dense(100, activation = 'relu'),
    Dense(10,activation = 'softmax')
    ])

model.compile(optimizer = 'adam',
              loss = 'categorical_crossentropy',
              metrics = ['accuracy'])
model.fit(x_train,y_train_cat, epochs = 5, batch_size = 32)
```

Завдання.

1. Реалізувати нейронні мережі (MLP, CNN) для розпізнавання цифр MNIST dataset.

2. Поекспериментувати з різними моделями та подивіться як змінюватиметься точність. Зокрема, можна змінити такі параметри:

- параметр epochs;
- кількість нейронів у прихованому шарі (10-500), і подивіться, яким чином змінюватиметься точність прогнозу моделі;

- додати додаткові шари між flatten-шаром і кінцевим dense-шаром, проведіть експерименти з кількістю нейронів на цьому шарі;
- поекспериментувати з кількістю згорткових блоків, їх параметрами;
- поекспериментувати з методом оптимізації, функцією втрат, функціями активації, тощо;
- не нормалізувати значення пікселів та подивіться, що з цього вийде.

3. Проаналізувати, як зміна параметрів впливає на точність моделі.

4. Зробити висновки щодо ефективності різних архітектур нейронних мереж для задачі класифікації зображень.

2.5. Задача класифікації. Згорткові нейромережі (CNN).

Згорткові нейронні мережі (Convolutional Neural Networks, CNN) є одним із найефективніших підходів для обробки та класифікації зображень. Вони дозволяють автоматично виділяти характерні ознаки об’єктів незалежно від їх положення, масштабу та орієнтації.

Розглянемо задачу класифікації кольорових зображень котів і собак на основі набору даних Microsoft Asirra. Кожне зображення в цьому наборі даних має мітку 1 або 0, якщо на зображенні відповідно собака або кішка.

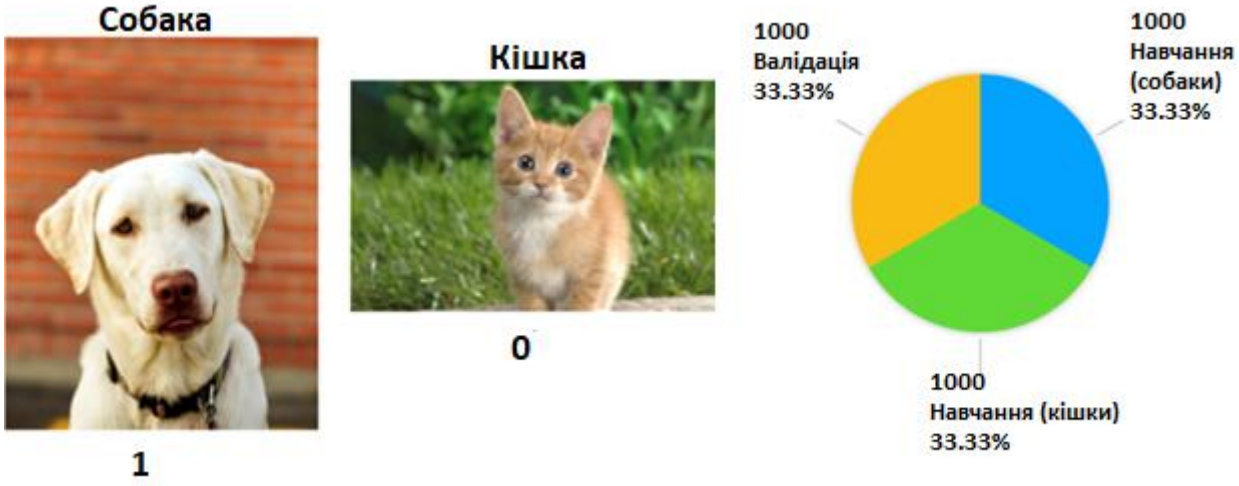

Рис. 5. Зображення котів та собак (Microsoft Asirra).

Перед тим як зображення можуть бути використані як вхідні дані для нашої мережі, їх необхідно перетворити до тензорів зі значеннями з плаваючою комою. Список кроків, які необхідно зробити для цього:

1. Прочитати зображення з диска.

2. Декодувати вміст зображень і перетворити на потрібний формат з урахуванням RGB-профілю.

3. Перетворити до тензорів зі значеннями з плаваючою комою.

4. Провести нормалізацію значень тензора з інтервалу від 0 до 255 до інтервалу від 0 до 1, так як нейронні мережі краще працюють з маленькими вхідними значеннями.

На щастя всі ці операції можуть бути виконані з використанням tf.keras.preprocessing.image.ImageDataGenerator-класу.

Вирішуючи завдання класифікації зображень, ми завжди вдаємося до одного з варіантів уніфікації вхідних даних – приведення розмірів зображень до єдиних значень (resizing).

Вдамося до зміни розмірів всіх зображень до розмірів 150 пікселів за висотою та 150 пікселів за шириною. Перетворюючи зображення до єдиного розміру ми, тим самим, гарантуємо, що на вхід нейронної мережі надходитиме картинка потрібного розміру і при передачі у flatten-шар ми отримаємо одновимірний масив такого ж розміру.

Архітектура CNN.

Модель складається з послідовності згорткових і підвибіркових шарів:

– кілька блоків Conv2D + MaxPooling2D;

– шар Flatten;

– повнозв'язний шар Dense (512 нейронів);

– вихідний шар Dense (2 нейрони, softmax).

Як і раніше, ми скористаємося оптимізатором adam. Як функцію втрат використаємо sparse_categorical_crossentropy. Також хочемо на кожній навчальній ітерації стежити за точністю моделі, тому передаємо значення accuracy у параметр metrics.

Для того щоб розібратися і зрозуміти, яким чином згорткові нейронні мережі працюють з кольоровими зображеннями, нам варто заглибитися в те, як саме мережі працюють взагалі.

Висота та ширина 3D-масиву буде визначена висотою та шириною зображення, а глибина (depth) визначається кількістю колірних каналів зображення.

Більшість кольорових зображень можуть бути представлені трьома каналами – червоним (red), зеленим (green) і синім (blue).

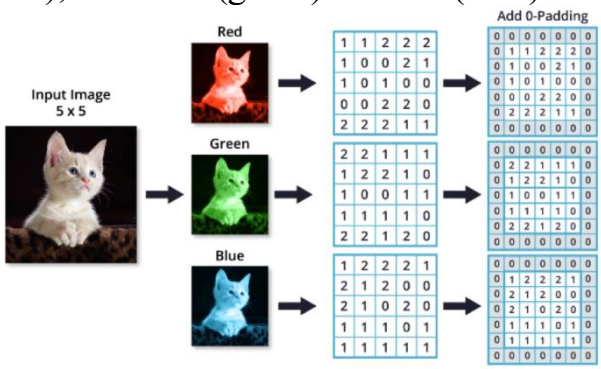

Рис. 6. RGB-зображення.

Зображення, що складаються з червоного, зеленого та синього каналів, називаються RGB-зображеннями. Об'єднання цих трьох каналів у результаті дає кольорове зображення. У кожному з RGB-зображенні кожен канал представлений окремим двовимірним масивом пікселів.

Оскільки кількість каналів у нас дорівнює трьом, то і в результаті у нас будуть три двовимірні масиви.

Так як навчальні блоки будуть надходити з генератора (ImageDataGenerator), ми скористаємося методом fit_generator замість раніше використовуваного методу fit.

Аугментація (Data Augmentation) – це метод штучного розширення навчального набору даних на основі існуючих прикладів з метою підвищення ефективності моделей глибокого навчання за обмеженого обсягу даних. Цей підхід є однією з форм регуляризації, яка сприяє покращенню узагальнювальної здатності моделі.

Тут ми використовували об'єкт Keras ImageDataGenerator для збільшення даних для випадкових перетворень (зміни розміру, повороту, інше) зображень. Кожен новий пакет даних випадково коригується відповідно до параметрів, наданих ImageDataGenerator.

Лістинг 3 – Класифікація зображень котів і собак (CNN).

```
from keras.models import Sequential
from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
_URL = 'https://storage.googleapis.com/mledu-
datasets/cats_and_dogs_filtered.zip'
zip_dir = tf.keras.utils.get_file('cats_and_dogs_filterted.zip',
origin=_URL, extract=True)
base_dir = os.path.join(os.path.dirname(zip_dir),
'cats_and_dogs_filtered')
train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')

train_cats_dir = os.path.join(train_dir, 'cats')
train_dogs_dir = os.path.join(train_dir, 'dogs')
validation_cats_dir = os.path.join(validation_dir, 'cats')
validation_dogs_dir = os.path.join(validation_dir, 'dogs')
batch = 100
img_size = 150
train_datagen = ImageDataGenerator(rescale = 1./255)
train_generator = train_datagen.flow_from_directory(
                        train_dir,
                        target_size =(img_size,img_size),
                        batch_size = batch,
                        class_mode = 'binary'    #)
test_datagen = ImageDataGenerator(rescale = 1./255)

validation_generator = test_datagen.flow_from_directory(
                        validation_dir,
                        target_size = (img_size,img_size),
                        batch_size = batch,
                        class_mode = 'binary')
model = Sequential([
    Conv2D(16,(3,3), activation = 'relu', input_shape =
(img_size,img_size,3)),
    MaxPooling2D(2,2),
    Conv2D(32,(3,3), activation = 'relu'),
```

```
    MaxPooling2D(2,2),
    Conv2D(64, (3,3), activation = 'relu'),
    MaxPooling2D(2,2),
    Flatten(),
    Dense(512, activation= 'relu'),
    Dense(2,activation = 'softmax')
])
model.compile(optimizer = 'adam',
              loss = 'sparse_categorical_crossentropy',
              metrics = ['accuracy'])
history = model.fit_generator(
                    train_generator,
                    steps_per_epoch = 20,
                    epochs = 15,
                    validation_data = validation_generator,
                    validation_steps = 5,)
```

Завдання.

1. Покращити точність моделі на тестових даних (CNN_cats_dogs.ipynb), змінюючи: кількість згорткових шарів; кількість фільтрів; розмір ядра згортки; параметри оптимізації.
2. Реалізувати аугментацію даних та оцінити її вплив на якість моделі.
3. Ознайомитися з попередньо навченими моделями з tf.keras.applications, зокрема: VGG16; VGG19; ResNet.
4. Порівняти результати власної моделі та моделей з попереднім навчанням.
5. Зробити висновки щодо ефективності згорткових нейронних мереж для задач класифікації зображень.

2.6. Прогнозування символів і слів рекурентною мережею. Embedding шар.

Рекурентні нейронні мережі (Recurrent Neural Networks, RNN) застосовуються для обробки послідовностей даних, зокрема текстів. Вони дозволяють враховувати попередній контекст при формуванні прогнозу.

Розглянемо задачу прогнозування наступного символу або слова на основі попередньої послідовності.

Як навчальну вибірку візьмемо заготовлений файл з короткими висловлюваннями. Потім потрібно визначитися: як представляти вхідні дані при роботі з текстовою інформацією? Припустимо, що спочатку ми маємо набір деяких висловлювань. Ми можемо переглядати цей текст (послідовно брати з нього символи inp_chars) і прогнозувати наступний. Щоб було простіше вирішувати це завдання, залишимо в тексті лише символи англійських літер та символи пропуску.

Далі, нам треба вирішити: у якому форматі подавати ці символи на вхід рекурентної мережі? Загальний вигляд нашої мережі (розгорнутої у часі) буде наступним (рис.7).

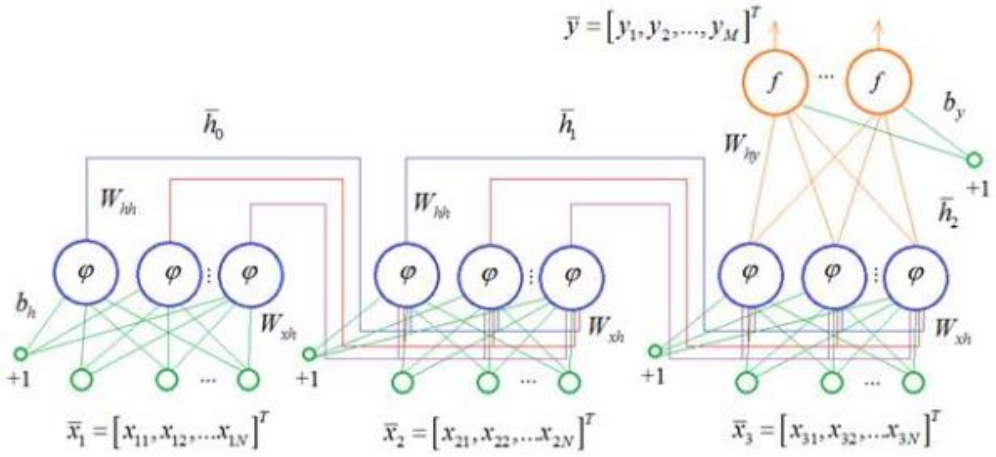

Рис. 7. Рекурентні нейронні мережі.

Тут послідовно подаються три символи (inp_chars = 3), а потім на виході формується прогноз наступного (четвертого) символу. Маємо рекурентну мережу виду – Many to One.

Що собою представляють вхідні вектори і вихідний вектор? Перше, що спадає на думку – це кожному символу поставити у відповідність деяке число і ці числа подавати на вхід мережі.

Для формування одного вихідного вектора на вхід потрібно подати вектори inp_chars у форматі One-hot encoding (OHE). Настав час створити таку навчальну вибірку. Ми її формуватимемо за допомогою інструмента tf.keras.preprocessing.text.Tokenizer, який робить «розумний» парсинг (розклад на складові елементи) зазначеного тексту.

Далі створимо рекурентну мережу за допомогою Keras відповідно до архітектури малюнка вище.

Лістинг 4 – Прогнозування символів (SimpleRNN).

```
token = Tokenizer(num_words = 27, char_level=True)
token.fit_on_texts(text)
inp_char = 3
data = token.texts_to_matrix(text)
n = data.shape[0]-inp_char
X = np.array([data[i:i+inp_char, :] for i in range(n)])
Y = data[inp_char:]
model = Sequential([InputLayer(input_shape=(inp_char,num_char)),
                    SimpleRNN(500, activation = 'tanh'),
                    Dense(num_char, activation = 'softmax')
                    ])
model.compile(optimizer = 'adam', loss =
'categorical_crossentropy', metrics = ['accuracy'])
history = model.fit(X, Y, batch_size = 32, epochs = 50)
```

У більш складному випадку аналізуються не окремі символи, а слова. Для цього текст перетворюється у послідовність індексів відповідно до словника.

Після цього виконується one-hot кодування або формується відповідний тензор навчальних даних.

Лістинг 5 – Прогнозування слів.

```
token = Tokenizer(num_words = 10000, filters = '!-"—
#$%&amp;()*+,-./:;<=>?@[\\]^_`{|}~\t\n\r«»',lower = True, split
= ' ', char_level = False)
token.fit_on_texts([text])
dist = list(token.word_counts.items())
data = token.texts_to_sequences([text])
result = tf.keras.utils.to_categorical(data[0],
num_classes=maxWords)
inp_words = 3
n = result.shape[0]-inp_words
X = np.array([result [i:i+inp_words, :] for i in range(n)])
Y = result [inp_words:]
model =
Sequential([InputLayer(input_shape=(inp_words,maxWords)),
                    SimpleRNN(500, activation = 'tanh'),
                    Dense(maxWords, activation = 'softmax')
                    ])
model.compile(optimizer = 'adam', loss =
'categorical_crossentropy', metrics = ['accuracy'])
history = model.fit(X, Y, batch_size = 32, epochs = 50)
```

Однак у такій реалізації є один істотний недолік: вхідний тензор, що ми отримали, займає в пам'яті дуже багато місця. Уявіть, якщо вирішується реальне завдання з числом слів 20 000. Тому фахівці з нейронних мереж запропонували альтернативний підхід – використання спеціального вхідного шару, який отримав назву Embedding.

У чому його суть? Embedding-шар: приймає на вхід індекси слів; відображає їх у вектори меншої розмірності; дозволяє зменшити обсяг пам’яті та прискорити навчання.

Фактично він реалізує компактне подання слів у вигляді щільних векторів.

Лістинг 6 – Прогнозування слів з використанням Embedding.

```
token = Tokenizer(num_words = 10000, filters = '!-"—
#$%&amp;()*+,-./:;<=>?@[\\]^_`{|}~\t\n\r«»',lower = True, split
= ' ', char_level = False)
token.fit_on_texts([text])
dist = list(token.word_counts.items())
data = token.texts_to_sequences([text])
result = np.array(data[0])
inp_words = 3
n = result.shape[0]-inp_words
X = np.array([result [i:i+inp_words] for i in range(n)])
Y = tf.keras.utils.to_categorical(result[inp_words:],
num_classes=maxWords)
```

```
model = Sequential([Embedding(maxWords,256, input_length =
inp_words),
                    SimpleRNN(256, activation = 'tanh'),
                    Dense(maxWords, activation = 'softmax')
                    ])
model.compile(optimizer = 'adam', loss =
'categorical_crossentropy', metrics = ['accuracy'])
history = model.fit(X, Y, batch_size = 32, epochs = 50)
```

Завдання.

1. Реалізувати модель рекурентної нейронної мережі для прогнозування тексту українською мовою.

2. Провести експерименти, змінюючи: довжину вхідної послідовності; розмір словника; розмірність Embedding-шару; кількість нейронів у RNN.

3. Порівняти результати моделей: з one-hot кодуванням; з використанням Embedding.

4. Зробити висновки щодо ефективності використання Embedding-шару для задач обробки тексту.

2.7. Сентимент-аналіз текстів (LSTM, GRU).

Сентимент-аналіз є однією з важливих задач обробки природної мови, яка полягає у визначенні емоційного забарвлення тексту, зокрема віднесенні висловлювання до позитивного або негативного класу (наприклад, kaggle/ Positive and Negative Sentences).

Для побудови моделі сентимент-аналізу необхідно підготувати навчальну вибірку, що складається з текстів позитивного та негативного змісту. Оскільки довжини текстових фрагментів можуть відрізнятися, їх потрібно привести до однакового розміру. Для цього короткі послідовності доповнюються нулями, а довгі – обрізаються до заданої довжини.

Змінна max_text_len визначає максимальну довжину текстової послідовності в словах. У результаті формується двовимірний тензор розмірності (batch_size, max_text_len), що містить індекси слів відповідно до побудованого словника.

Для підготовки текстових даних використовуються:

– Tokenizer – для перетворення текстів у послідовності індексів;

– pad_sequences – для вирівнювання довжини послідовностей.

Після формування вибірки тексти перемішуються, що сприяє більш ефективному навчанню моделі.

Модель на основі LSTM.

Для аналізу текстових послідовностей часто використовують шар LSTM (Long Short-Term Memory), який дозволяє враховувати довготривалі залежності в тексті.

Лістинг 7 – Сентимент-аналіз із використанням LSTM.

```
token = Tokenizer(num_words=10000, filters='!-"—#$%&amp;()*+,-
./:;<=>?@[\\]^_`{|}~\t\n\r«»',
                          lower=True, split=' ',
char_level=False)
token.fit_on_texts(texts)
dist = list(token.word_counts.items())
data = token.texts_to_sequences(texts)
max_text_len = 50
data_pad = pad_sequences(data, maxlen=max_text_len)
X = data_pad
Y = np.array([[1, 0]]*count_true + [[0, 1]]*count_false)
inds = np.random.choice(X.shape[0], size = X.shape[0],replace =
False)
X = X[inds]
Y = Y[inds]
model = Sequential([Embedding(maxWords, 128, input_length =
max_text_len),
                    LSTM(128, activation = 'tanh', dropout=0.5,
return_sequences=True),
                    LSTM(64, activation = 'tanh'),
                    Dense(2, activation = 'softmax')
])
model.compile(optimizer='adam', loss =
'categorical_crossentropy', metrics = ['accuracy'])
history = model.fit(X, Y, batch_size=32, epochs=10)
```

Модель на основі GRU.

Альтернативою LSTM є шар GRU (Gated Recurrent Unit), який має простішу структуру та в багатьох випадках забезпечує подібну якість результатів при менших обчислювальних витратах.

Лістинг 8 – Сентимент-аналіз із використанням GRU.

```
model = Sequential([Embedding(maxWords, 128, input_length =
max_text_len),
                    GRU(128, activation = 'tanh', dropout=0.5,
return_sequences=True),
                    GRU(64, activation = 'tanh'),
                    Dense(2, activation = 'softmax')
])
model.compile(optimizer='adam', loss =
'categorical_crossentropy', metrics = ['AUC'])
history = model.fit(X, Y, batch_size=32, epochs=10)
```

Наведені приклади мають демонстраційний характер. Якість класифікації значною мірою залежить від обсягу та якості навчальної вибірки. За невеликого обсягу даних результати можуть бути нестабільними, тому наведені моделі слід розглядати як ілюстрацію загального підходу до розв’язання задачі сентимент-аналізу.

Однією з проблем рекурентних мереж, зокрема LSTM і GRU, є схильність до перенавчання. Для зменшення цього ефекту доцільно:

– контролювати якість моделі на валідаційній вибірці;

– використовувати регуляризацію;

– застосовувати Dropout та Batch Normalization.

У бібліотеці Keras для рекурентних шарів можуть використовуватися такі параметри:

– dropout – для нейронів, пов’язаних із вхідним вектором;

– recurrent_dropout – для нейронів, пов’язаних із рекурентним станом.

Завдання.

1. Підготувати навчальну та тестову вибірки для задачі сентимент-аналізу.
2. Реалізувати моделі на основі LSTM та GRU для класифікації текстів на позитивні та негативні.
3. Оцінити якість моделей на тестовій вибірці за відповідними метриками.
4. Порівняти результати моделей LSTM і GRU.
5. Дослідити вплив параметрів dropout та recurrent_dropout на якість класифікації.
6. Зробити висновки щодо ефективності рекурентних нейронних мереж у задачах аналізу тексту.

2.8. Двоспрямовані рекурентні нейронні мережі.

У розглянутих раніше рекурентних нейронних мережах використовується каузальна структура, тобто стан мережі в момент часу (t) залежить лише від поточного входу (x(t)) та інформації з попередніх моментів часу (x(1), …, x(t-1)). У деяких випадках також може враховуватися інформація про попередні вихідні значення.

Проте в багатьох практичних задачах передбачення (y(t)) може залежати не лише від минулого, а й від майбутнього контексту. Наприклад, у задачах розпізнавання мовлення, рукописного тексту або аналізу природної мови правильна інтерпретація поточного елемента часто визначається як попередніми, так і наступними елементами послідовності.

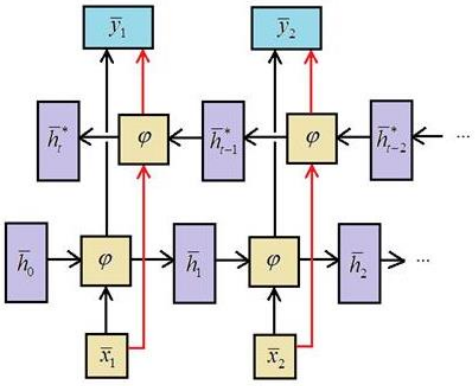

Рис. 8. Двоспрямовані рекурентні нейронні мережі.

Для розв’язання таких задач застосовуються *двоспрямовані рекурентні нейронні мережі (Bidirectional Recurrent Neural Networks)*. Вони поєднують дві рекурентні мережі:

– одну, що обробляє послідовність у прямому напрямку;

– іншу, що обробляє послідовність у зворотному напрямку.

Такий підхід дозволяє формувати вихідне представлення, яке враховує інформацію як із минулого, так і з майбутнього контексту.

У бібліотеці Keras для побудови двоспрямованих рекурентних мереж використовується клас: keras.layers.Bidirectional(...)

Як приклад розглянемо задачу регресії. Нехай задано синусоїдальний сигнал із шумом. Мета полягає у прогнозуванні окремого значення сигналу на основі кількох попередніх і кількох наступних спостережень.

Для цього формується тривимірний тензор вхідних даних, який дозволяє мережі враховувати положення кожного елемента відносно прогнозованого значення.

Лістинг 9 – Використання двоспрямованої GRU для задачі регресії.

```
N = 10000
data = np.array([np.sin(x/20) for x in range(N)]) +
0.1*np.random.randn(N)
plt.plot(data[:100])
n = 3
lenght = 2*n + 1
X =
np.array([np.diag(np.hstack((data[i:i+n],data[i+n+1:i+lenght])))
for i in range (N-lenght)])
Y = data[n:N-n-1]
model = Sequential([Input((lenght-1,lenght-1)),
                    Bidirectional(GRU(2)),
                    Dense(1,activation = 'linear')
])
model.compile(loss = 'mse', optimizer = Adam(0.01))
history = model.fit(X,Y,batch_size = 32, epochs = 10)
```

У наведеному прикладі використовується двоспрямований шар GRU, який обробляє вхідну послідовність у двох напрямках. Це дозволяє враховувати повний контекст при побудові прогнозу.

Завдання.

1. Реалізувати двоспрямовану рекурентну нейронну мережу для задачі аналізу послідовностей. Використати двоспрямовані RNN для підбору слів відповідно до контексту в українському тексті.

2. Порівняти результати односпрямованої та двоспрямованої рекурентних мереж.

3. Зробити висновки щодо доцільності використання двоспрямованих мереж у задачах обробки послідовностей.

2.9. Автокодувальник.

Автокодувальники (Autoencoders, AE) – це тип нейронних мереж, призначених для навчання ефективного представлення даних (кодування) з можливістю їх подальшого відновлення.

Основною особливістю автокодувальників є те, що розмірність вхідного та вихідного шарів збігається.

Автокодувальник складається з двох основних частин:

1. Енкодер: відповідає за стиск входу в latent-space. Представлений функцією кодування h = f(x);
2. Декодер: призначений для відновлення введення з latent-space. Подано функцією декодування h = f (x).

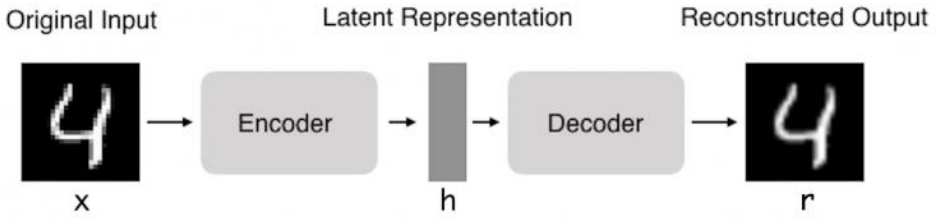

Рис. 9. Автокодувальник.

Типи автокодувальників.

Розглянемо основні типи автокодувальників:

1) Базовий (Vanilla Autoencoder)
2) Багатошаровий автокодувальник
3) Згортковий автокодувальник (Convolutional AE)
4) Регуляризований автокодувальник

Щоб проілюструвати типи автокодувальників, за допомогою структури Keras та датасету MNIST створено приклад кожного з них.

Лістинг 10 – Базовий автокодувальник (Vanilla AE).

```
inp = Input((28, 28, 1))
x = Flatten()(inp)
# Encoder
h = Dense(64, activation='relu')(x)
# Decoder
d = Dense(28*28, activation='sigmoid')(h)
out = Reshape((28, 28, 1))(d)
autoencoder = keras.Model(inp, out)
autoencoder.compile(optimizer='adam', loss='mse')
```

У своїй простій формі автокодувальник є мережею із трьох шарів, тобто нейронною мережею з одним прихованим шаром. Вхідні та вихідні дані збігаються, а модель навчається відновлювати вхід, використовуючи, зокрема, оптимізатор Adam та функцію втрат середньоквадратичної помилки.

Тут бачимо неповний автокодувальник, оскільки розмір прихованого шару (64) менше, ніж вхід (784). Накладене обмеження забезпечує навчання нейронної мережі компактному (стислому) поданню даних.

Якщо одного прихованого шару недостатньо, автокодувальник розширюють до більшої кількості.

Лістинг 11 – Багатошаровий автокодувальник.

```
inp = Input((28, 28, 1))
x = Flatten()(inp)
# Encoder
hidden_1 = Dense(128, activation='relu')(x)
h = Dense(64, activation='relu')(hidden_1)
# Decoder
hidden_2 = Dense(128, activation='relu')(h)
d = Dense(28*28, activation='sigmoid')(h)
out = Reshape((28, 28, 1))(d)
autoencoder = keras.Model(inp, out)
autoencoder.compile(optimizer='adam', loss='mse')
```

Лістинг 12 – Згортковий автокодувальник.

```
inp = Input(shape=(28, 28,1))
# Encoder
conv1_1 = Conv2D(16,(3, 3),activation='relu',padding='same')(inp)
pool1 = MaxPooling2D((2, 2), padding='same')(conv1_1)
conv1_2 = Conv2D(8,(3,3),activation='relu',padding='same')(pool1)
pool2 = MaxPooling2D((2, 2), adding='same')(conv1_2)
conv1_3 = Conv2D(8,(3,3),activation='relu',padding='same')(pool2)
h = MaxPooling2D((2, 2), padding='same')(conv1_3)
# Decoder
conv2_1 = Conv2D(8, (3, 3), activation='relu', padding='same')(h)
up1 = UpSampling2D((2, 2))(conv2_1)
conv2_2 = Conv2D(8, (3, 3),activation='relu',padding='same')(up1)
up2 = UpSampling2D((2, 2))(conv2_2)
conv2_3 = Conv2D(16, (3, 3), activation='relu')(up2)
up3 = UpSampling2D((2, 2))(conv2_3)
out = Conv2D(1,(3, 3),activation='sigmoid', padding='same')(up3)
autoencoder = keras.Model(inp, out)
autoencoder.compile(optimizer='adam', loss='mse')
```

Регуляризований автоенкодер.

Замість того, щоб обмежувати ємність моделі, зберігаючи кодер та декодер неглибокими, та використовувати короткий код, регуляризовані автокодувальники використовують функцію втрат. Вона додає моделі інші

властивості, крім копіювання входу на вихід. Насправді використовують два типи регуляризованого автоенкодера: розріджений (sparse) і шумопригнічувальний (denoising).

Sparse автокодувальник.

Sparse-автокодувальники зазвичай використовуються для навчання ознак, які можуть бути застосовані не лише для задач класифікації. Автокодувальник, регуляризований як sparse, реагує на специфічні характеристики набору даних, на якому він навчений, замість простого відтворення вхідних даних. Таким чином, у процесі навчання sparse-автокодувальника формується модель, яка, як побічний ефект, навчається виділяти корисні ознаки.

```
h = Dense(128, activation='relu', activity_regularizer=regulariz
ers.l1(10e-5))(x)
```

Denoising автокодувальник.

Замість додавання регуляризаційного штрафу до функції втрат можна побудувати автокодувальник, який навчається виділяти інформативні ознаки шляхом модифікації задачі реконструкції. Це досягається додаванням шуму до вхідних даних і навчанням моделі відновлювати початкове (чисте) зображення.

У такій постановці автокодувальник не просто копіює вхід, а змушений виділяти суттєві характеристики даних. У результаті енкодер формує більш узагальнене та стійке (робастне) представлення, яке краще відображає структуру даних і менш чутливе до шуму.

Завдання.

1. Реалізувати Denoising Autoencoder: додати шум до навчальної вибірки (x_train); сформувати зашумлену вибірку (x_augmentation); навчити модель відновлювати початкові дані.
2. Дослідити роботу автокодувальників на кольорових зображеннях (наприклад, набір даних cats and dogs).
3. Порівняти якість реконструкції для різних типів автокодувальників.
4. Зробити висновки щодо ефективності використання автокодувальників для задач обробки зображень.

2.10. Варіаційні автокодувальники (VAE).

Варіаційний автокодувальник (Variational Autoencoder, VAE) є узагальненням класичного автокодувальника, у якому прихований простір описується не фіксованими значеннями, а ймовірнісним розподілом. Це дозволяє не лише стискати дані, а й генерувати нові зразки, подібні до навчальної вибірки.

Спочатку задають розмірність прихованого простору, розмір пакета (batch size), а також допоміжну функцію для застосування шарів Dropout і BatchNormalization.

Далі формується мережа енкодера. Використовуючи повнозв'язні шари, модель обчислює два вектори: z_mean – вектор математичних сподівань; z_log_var – вектор логарифмів дисперсій.

Логарифм дисперсії використовується для зручності подальших обчислень. На основі цих двох векторів формується приховане представлення (h), яке моделюється як випадкова величина з нормальним розподілом. Для цього застосовується спеціальний шар Lambda, який реалізує параметризацію прихованого простору.

Після цього визначається мережа декодера, яка відновлює вихідні дані з прихованого представлення. На основі побудованих блоків формуються моделі енкодера, декодера та повного варіаційного автокодувальника.

Для навчання моделі використовується спеціальна функція втрат, що складається з двох частин:

– похибки реконструкції між вхідними та вихідними даними;

– відстані Кульбака–Лейблера, яка забезпечує наближення розподілу прихованих змінних до нормального.

Похибка реконструкції обчислюється як сума квадратів різниць між вхідним і відновленим зображенням. Відстань Кульбака–Лейблера обчислюється за векторами z_mean і z_log_var.

Лістинг 13 – Реалізація варіаційного автокодувальника

```
batch_size = 50
def dropout_a_batch(x):
  return Dropout(0.3)(BatchNormalization()(x))
inp = Input(shape =(28,28,1))
img = Flatten()(inp)
#Encoder
x = Dense(256,activation = 'relu')(img)
x = dropout_a_batch(x)
x = Dense(128,activation = 'relu')(x)
x = dropout_a_batch(x)

z_mean = Dense(2)(x)
z_log = Dense(2)(x)
def hidden(args):
  global z_mean, z_log
  z_mean, z_log = args
  N = K.random_normal(shape=(batch_size,2), mean = 0., stddev =
1.0)
  return K.exp(z_log/2) * N + z_mean

h = Lambda(hidden, output_shape = (2,))([z_mean, z_log])
#Decoder
input_dec = Input((2,))
x = Dense(128,activation = 'relu')(input_dec)
```

```
x = dropout_a_batch(x)
x = Dense(256,activation = 'relu')(x)
x = dropout_a_batch(x)
x = Dense(28*28, activation = 'sigmoid')(x)
out = Reshape((28,28,1))(x)
encoder = keras.Model(inp,h, name = 'encoder')
decoder = keras.Model(input_dec,out, name = 'decoder')
vae = keras.Model(inp,decoder(encoder(inp)), name = 'vae')
def vae_loss(x,y):
  x = K.reshape(x, shape=(batch_size,28*28))
  y = K.reshape(y, shape=(batch_size,28*28))
  loss = K.sum(K.square(x-y),axis=-1)
  kl_loss = - 0.5 * K.sum(1 + z_log - K.square(z_mean) -
K.exp(z_log),axis=-1)
  return loss + kl_loss
vae.compile(optimizer='adam', loss = vae_loss)
vae.fit(x_train,x_train,epochs = 5,
batch_size=batch_size,shuffle=True)
```

У наведеному прикладі реалізовано простий варіаційний автокодувальник, який формує компактний прихований простір і декодує його значення назад у зображення. Якість реконструкції залежить, зокрема, від розмірності прихованого простору. Якщо розмірність занадто мала, модель не здатна повністю відобразити всі особливості вхідних даних.

Одним із шляхів покращення якості моделі є збільшення розмірності прихованого простору. Інший підхід полягає у використанні додаткової інформації про класи об’єктів. Якщо для навчальної вибірки відомі мітки класів, їх можна подавати на вхід енкодера і декодера.

Така модель називається *умовним варіаційним автокодувальником (Conditional Variational Autoencoder, CVAE)*. Її структура подібна до звичайного VAE, але додатково використовує інформацію про клас об’єкта.

Завдання.

1. Дослідити вплив розмірності прихованого простору на якість реконструкції.
2. Побудувати Conditional Variational Autoencoder (CVAE).
3. Порівняти результати роботи VAE та CVAE.
4. Зробити висновки щодо використання варіаційних автокодувальників у задачах генерації та реконструкції зображень.

2.11. Генеративно-змагальні мережі (GAN).

Генеративно-змагальні мережі (Generative Adversarial Networks, GAN) – це клас моделей глибокого навчання, які використовуються для генерації нових даних, подібних до навчальної вибірки.

GAN складається з двох нейронних мереж:

– генератора (generator), який створює нові зразки даних;

– дискримінатора (discriminator), який намагається відрізнити реальні дані від згенерованих.

Навчання GAN відбувається як змагальний процес: генератор намагається “обманути” дискримінатор, а дискримінатор – правильно класифікувати зразки.

Для покращення якості генерації доцільно використовувати однорідні дані. Наприклад, з набору MNIST можна вибрати лише зображення цифри «7»:

```
x_train = x_train[y_train == 7]
y_train = y_train[y_train == 7]
```

Далі визначаються параметри навчання:

```
buffer_size = x_train.shape[0]
batch_size = 100
buffer_size = buffer_size // batch_size * batch_size
x_train = x_train[:buffer_size]
y_train = y_train[:buffer_size]
```

Формується набір даних:

```
train_dataset = tf.data.Dataset.from_tensor_slices(x_train) \
   .shuffle(buffer_size) \
   .batch(batch_size)Потім визначимо дві константи:
buffer_size = x_train.shape[0]
batch_size = 100
```

Модель генератора.

Генератор отримує на вхід випадковий вектор (шум) та формує зображення.

Лістинг 14 – Генератор GAN

```
hidden_dim = 2
generator = Sequential([Dense(7*7*256, activation = 'relu',
input_shape = (hidden_dim,)),
                        BatchNormalization(),
                        Reshape((7,7,256)),
                        Conv2DTranspose(128,(5,5), strides =
(1,1), padding='same', activation = 'relu'),
                        BatchNormalization(),
                        Conv2DTranspose(64,(5,5), strides =
(2,2), padding='same', activation = 'relu'),
                        BatchNormalization(),
                        Conv2DTranspose(1,(5,5), strides =
(2,2), padding='same', activation = 'sigmoid')
                        ])
```

Генератор на вхід отримуватиме вектор незалежних нормальних випадкових величин, розмірністю hidden_dim.

Потім, за допомогою шару Dense, він масштабується до розміру 7*7*256 елементів і в шарі Reshape перетворюється на тензор з розмірами:7x7x256. Далі виконується операція транспонованої згортки Conv2DTranspose.

Модель дискримінатора.

Дискримінатор визначає, чи є зображення реальним або згенерованим.

Лістинг 15 – Дискримінатор GAN.

```
discriminator = Sequential([Conv2D(64,(5,5),strides=(2,2),
padding='same',input_shape = (28,28,1)),
                            LeakyReLU(),
                            Dropout(0.3),
                            Conv2D(128,(5,5),strides=(2,2),
padding='same'),
                            LeakyReLU(),
                            Dropout(0.3),
                            Flatten(),
                            Dense(1)
                            ])
```

Отже, мережі визначено. Далі ми оголосимо дві функції для обчислення втрат генератора та дискримінатора:

```
cross_ent = tf.keras.losses.BinaryCrossentropy(from_logits=True)
def generator_loss(fake_output):
  return cross_ent(tf.ones_like(fake_output),fake_output)

def discriminator_loss(fake_output, real_output):
  real_loss = cross_ent(tf.ones_like(real_output),real_output)
  fake_loss = cross_ent(tf.zeros_like(fake_output),fake_output)
  return real_loss + fake_loss

gen_optimizer = tf.keras.optimizers.Adam(1e-4)
dis_optimizer = tf.keras.optimizers.Adam(1e-4)
```

Тепер у нас все готове, щоб сформувати процес навчання генератора та дискримінатора. Для цього скористаємося безпосередньо засобами Tensorflow 2.0 та визначимо функцію одного кроку навчання через декоратор tf.function:

Лістинг 16 – Крок навчання GAN.

```
@tf.function
def train_step(images):
  noise = tf.random.normal([batch_size,hidden_dim])
  with tf.GradientTape() as gen_tape, tf.GradientTape() as
dis_tape:
    art_img = generator(noise,training=True)

    fake_output = discriminator(art_img,training=True)
    real_output = discriminator(images,training=True)

    gen_loss = generator_loss(fake_output)
    dis_loss = discriminator_loss(fake_output, real_output)
```

```
    gen_grad =
gen_tape.gradient(gen_loss,generator.trainable_variables)
    dis_grad =
dis_tape.gradient(dis_loss,discriminator.trainable_variables)

    gen_optimizer.apply_gradients(zip(gen_grad,generator.trainab
le_variables))
    dis_optimizer.apply_gradients(zip(dis_grad,discriminator.tra
inable_variables))
  return gen_loss, dis_loss
```

Тепер, оголосимо функцію, яка запускатиме весь процес навчання мереж. На її вхід передаватимемо вибрану зображень, що навчається, розбитою по батчам, і число епох:

```
def train(dataset, epochs):
  history = []
  print_label = 10
  th = buffer_size // (batch_size*print_label)

  for epoch in range(1, epochs+1):
    print(f'{epoch}/{epochs}: ', end='')
    n = 0
    gen_loss_epoch = 0
    for img in dataset:
      gen_loss, dis_loss = train_step(img)
      gen_loss_epoch += K.mean(gen_loss)
      if( n % th == 0): print('=', end='')
      n += 1

    history += [gen_loss_epoch/n]
    print(': '+str(history[-1]))
  return history
```

Нарешті, запускаємо процес навчання та відображаємо графік зміни втрат для генератора за епохами:

```
epochs = 15
history = train(train_dataset,epochs)
plt.plot(history)
plt.grid(True)
```

Генеративно-змагальні мережі дозволяють отримувати нові зображення, які подібні до навчальної вибірки. Якість генерації залежить від архітектури мереж, параметрів навчання та обсягу даних.

Завдання.

1. Проаналізувати згенеровані зображення після навчання моделі.
2. Дослідити вплив розмірності вхідного шуму (hidden_dim) на якість генерації.
3. Змінити архітектуру генератора або дискримінатора та оцінити результати.

4. Зробити висновки щодо ефективності GAN для генерації зображень.

2.12. Самоорганізовані карти Кохонена (SOM).

Самоорганізовані карти Кохонена (Self-Organizing Maps, SOM) – це тип нейронних мереж, що використовуються для кластеризації, візуалізації та зниження розмірності даних без учителя.

Основна ідея SOM полягає у відображенні багатовимірних даних на двовимірну решітку нейронів із збереженням топологічної структури даних.

Принцип роботи SOM.

Карта складається з набору нейронів, розташованих у вигляді двовимірної сітки. Кожен нейрон має вектор ваг тієї ж розмірності, що й вхідні дані.

Процес навчання відбувається таким чином:

– на вхід подається вектор даних;

– визначається нейрон-переможець (Best Matching Unit, BMU);

– ваги нейрона-переможця та його сусідів коригуються;

– поступово карта “організовується”, відображаючи структуру даних.

Алгоритм навчання SOM.

1.Ініціалізувати ваги нейронів випадковими значеннями.

2. Для кожного вхідного вектора (x):

– знайти нейрон-переможець: $c = \text{argmin}_i |x - w_i|$

– оновити ваги нейронів: $w_i^{t+1} := w_i^t + \eta h_{ci}^t (x - w_i)$

де η – коефіцієнт навчання; h_{ci}^t – функція сусідства.

3. Повторювати процес до збіжності.

Функція сусідства визначає, наскільки сильно змінюються ваги нейронів, що знаходяться поруч із переможцем. Зазвичай використовується гаусівська функція:

$$h_{ci}^t = e^{-|r_c - r_i|^2 2\sigma^2}$$

Для реалізації самоорганізованих карт зручно використовувати бібліотеку MiniSom.

Лістинг 17 – Побудова SOM.

```
from minisom import MiniSom
import numpy as np
data = np.random.rand(100, 3)
som = MiniSom(x=10, y=10, input_len=3, sigma=1.0,
learning_rate=0.5)
som.random_weights_init(data)
som.train_random(data, num_iteration=100)
winner_coordinates = np.array([som.winner(x) for x in data])
```

SOM дозволяє візуалізувати структуру даних, наприклад:

– карта відстаней (U-Matrix);

– кластеризація даних;

– розподіл об’єктів по карті.

Завдання.

1. Реалізувати самоорганізовану карту Кохонена для заданого набору даних.

2. Провести кластеризацію даних за допомогою SOM.

3. Візуалізувати результати (наприклад, U-Matrix).

4. Порівняти результати SOM із іншими методами кластеризації (наприклад, k-means).

5. Зробити висновки щодо ефективності SOM.

3. САМОСТІЙНІ ЗАВДАННЯ З ДИСЦИПЛІНИ.

Самостійна робота – головний спосіб вивчення дисципліни, органічна частина навчального процесу. Вона допомагає глибоко засвоїти матеріал, закріпити знання, поглибити вміння та навички в пізнавальній діяльності, творчо мислити; виховує організованість і дисциплінованість, активність та ініціативу, настирливість у досягненні мети; сприяє виробленню власних прийомів і методів пізнання, вчить раціонально організовувати та контролювати робочий час.

При вивченні дисципліни студент повинен ознайомитися з навчальною програмою дисципліни, її структурою, формами і методами навчання, видами та методами контролю знань. Програма дисципліни є джерелом самоконтролю, особливо при підготовці до заліку.

3.1. Персептрон Розенблатта. Метод стохастичного градієнта SG.

У процесі самостійного опрацювання теми необхідно звернути увагу на такі ключові питання:

– постановка задач навчання з прецедентів;

– об’єкти та ознаки;

– основні поняття машинного навчання: модель алгоритмів, метод навчання, функція втрат, функціонал якості;

– принцип мінімізації емпіричного ризику;

– узагальнююча здатність та перехресна перевірка (cross-validation);

– базова модель нейронної мережі;

– градієнтні методи оптимізації;

– метод стохастичного градієнтного спуску (SGD) та його модифікації;

– біологічний нейрон та модель МакКаллока–Піттса як лінійний класифікатор;

– повнота двошарових нейронних мереж у просторі булевих функцій;

– алгоритм зворотного розповсюдження помилки;

– швидкі методи оптимізації: Поляка, Нестерова, AdaGrad, RMSProp, AdaDelta, Adam, Nadam;

– проблема вибуху градієнта та метод gradient clipping;

– метод випадкового вимкнення нейронів (Dropout);

– проблема перенавчання та «паралічу» мережі.

Питання і завдання для самостійного опрацювання.

1. Що таке модель МакКаллока-Піттса?
2. Описати метод стохастичного градієнтного спуску. Вивести градієнтний крок для квадратичної функції втрат і сигмоїдної функції активації.
3. Які основні недоліки методу SGD та способи їх подолання?
4. Що таке лінійний адаптивний елемент (ADALINE)?
5. Пояснити правило Хебба.
6. Що таке регуляризація (скорочення ваг)?
7. Обґрунтувати логістичну регресію: основні припущення; апостеріорна ймовірність класів; функція втрат.

3.2. Згорткові нейронні мережі (CNN) для зображень. Згортки для сигналів, текстів, графів, ігор.

У процесі самостійного опрацювання теми необхідно звернути увагу на такі ключові питання:

– обґрунтування глибоких нейронних мереж: виразні можливості та швидкість збіжності;

– роль надмірної параметризації в навчанні нейронних мереж;

– згорткові нейронні мережі (CNN) для обробки зображень;

– згортковий нейрон і операція згортки;

– архітектура CNN: згорткові шари, підвибірка (pooling), нормалізація, регуляризація;

– набір даних ImageNet та його роль у розвитку глибокого навчання;

– залишкові нейронні мережі (ResNet);

– застосування згорток для:

- сигналів;
- текстів;
- графів;
- ігрових задач.

Питання і завдання для самостійного опрацювання.

1. Що таке глибокі нейронні мережі та в чому їх переваги?
2. Пояснити принцип роботи основних шарів нейронної мережі: згортковий шар; повнозв’язний шар; шари підвибірки (max pooling, average pooling); шари нормалізації (batch normalization, layer normalization); Dropout.
3. Описати основні архітектури згорткових нейронних мереж: LeNet; AlexNet; VGG; Inception; ResNet; DenseNet; EfficientNet.

Практичне завдання.

Розв’язати задачу класифікації зображень із використанням згорткової нейронної мережі.

Використати набір даних:

https://www.kaggle.com/datasets/sid4sal/alpaca-dataset-small

У наборі даних виділено два класи: «альпака»; «не альпака».

Необхідно:

✓ Підготувати дані (завантаження, нормалізація, розбиття на train/test).
✓ Побудувати модель CNN.
✓ Навчити модель класифікувати зображення.
✓ Оцінити якість моделі (accuracy, loss).
✓ Проаналізувати результати та зробити висновки.

3.3. Рекурентні нейронні мережі. LSTM. Рекурсивні нейронні мережі.

У процесі самостійного опрацювання теми необхідно звернути увагу на такі ключові питання:

– рекурентні нейронні мережі (RNN) та їх призначення для обробки послідовностей;

– навчання рекурентних мереж: алгоритм Backpropagation Through Time (BPTT);

– проблема згасання та вибуху градієнта;

– мережі довготривалої пам’яті (LSTM);

– мережі Gated Recurrent Unit (GRU) та Simple Recurrent Unit (SRU);

– застосування рекурентних мереж у задачах обробки природної мови.

Питання і завдання для самостійного опрацювання.

1. Описати архітектури рекурентних нейронних мереж: RNN; LSTM; GRU.

2. Пояснити проблему згасання та вибуху градієнта.

3. Пояснити метод gradient clipping та його роль у навчанні рекурентних мереж.

4. Пояснити роль генеративних моделей у задачах комп’ютерного зору (GAN) та їх зв’язок із обробкою послідовностей.

Практичне завдання.

Семантична подібність – це задача визначення ступеня схожості двох речень за їх змістом.

Необхідно реалізувати модель для оцінки семантичної подібності речень із використанням трансформерів.

Використати корпус SNLI (Stanford Natural Language Inference) та приклад реалізації: https://keras.io/examples/nlp/semantic_similarity_with_bert/

3.4. Нейронна мережа Кохонена.

У процесі самостійного опрацювання теми необхідно звернути увагу на такі ключові питання:

– нейронні мережі Кохонена та їх призначення;
– конкурентне навчання;
– стратегії конкуренції:
  - WTA (Winner-Takes-All);
  - WTM (Winner-Takes-Most);
– самоорганізовані карти Кохонена (Self-Organizing Maps, SOM);
– застосування SOM для візуалізації та аналізу даних.

Питання і завдання для самостійного опрацювання.

1. Описати мережі Кохонена та правила конкуренції: жорстка конкуренція; справедлива конкуренція; м’яка конкуренція.
2. Розглянути алгоритм навчання.
3. Пояснити задачу квантування даних та її зв’язок із SOM.
4. Розглянути задачу багатовимірної візуалізації та роль самоорганізованих карт Кохонена у її розв’язанні.

Практичне завдання.

Створити двовимірну самоорганізовану карту Кохонена розміром (5x6) із гексагональною топологією.

Для навчання використати вибірку з 1000 випадкових двовимірних векторів, компоненти яких рівномірно розподілені в інтервалі ([-1; 1]).

3.5. Завдання із зображеннями. Моделювання мови.

У процесі самостійного опрацювання теми необхідно звернути увагу на такі ключові питання:

– згорткові нейронні мережі для розпізнавання зображень;
– задачі комп’ютерного зору;
– класифікація зображень;
– виділення об’єктів (object detection);
– відстеження об’єктів (object tracking);
– використання онлайн-навчання;
– мовні моделі та їх застосування;
– генерація текстів;
– машинний переклад на основі нейронних мереж.

Питання і завдання для самостійного опрацювання.

1. Описати застосування згорткових нейронних мереж у задачах розпізнавання зображень.

2. Розглянути архітектуру моделей типу encoder–decoder (RNN encoder–decoder).

Практичне завдання.

Побудувати модель нейронної мережі, на основі якої буде виконуватися переклад поданого тексту з мови оригіналу на інші. Для досягнення поставленої мети необхідно виконати такі завдання:

- ✓ сформувати вхідний математичний опис системи мультилінгвістичного перекладу на основі технології штучних нейронних мереж;
- ✓ обрати механізм кодування-декодування тексту;
- ✓ обрати механізм уваги; розробити моделі нейронних мереж для прямого перекладу та перекладу з використанням мови-посередника;
- ✓ розробити та програмно реалізувати алгоритмічне забезпечення системи мультилінгвістичного перекладу;
- ✓ провести тестування системи мультилінгвістичного перекладу.

3.6. Застосування алгоритмів глибокого навчання до розв'язання прикладних задач.

У процесі самостійного опрацювання теми необхідно звернути увагу на такі ключові напрями застосування алгоритмів глибокого навчання:

– використання нейронних мереж у медицині:
  - діагностика захворювань;
  - аналіз медичних зображень;

– діагностика несправностей складних технічних систем;

– застосування в соціально-економічних задачах:
  - прогнозування результатів виборів;
  - аналіз поведінки користувачів;

– застосування у фінансовій сфері:
  - кредитний скоринг;
  - прогнозування банкрутств;
  - прогнозування валютних курсів;
  - аналіз та прогнозування цінних паперів;
  - моделювання ринку житлової нерухомості;

– поєднання нейронних мереж із експертними системами.

Завдання для самостійного опрацювання.

1. Описати можливості застосування глибокого навчання у медицині.

2. Розглянути приклади використання нейронних мереж для діагностики технічних систем.

3. Проаналізувати застосування нейронних мереж у фінансовій сфері.

4. Описати підходи до прогнозування часових рядів за допомогою нейронних мереж.

Практичне завдання.

Обрати одну прикладну задачу (медицина, фінанси, технічні системи або інша) та:

- ✓ Сформулювати постановку задачі.
- ✓ Обрати тип нейронної мережі (CNN, RNN, трансформер тощо).
- ✓ Підготувати або знайти відповідний набір даних.
- ✓ Реалізувати модель.
- ✓ Провести навчання та оцінити якість результатів.
- ✓ Проаналізувати отримані результати та зробити висновки.

ЛІТЕРАТУРА.

1. Кочура Ю.П., Гордієнко Ю.Г. Нейронні мережі. [Електронний ресурс]: підруч. для здобувачів ступеня бакалавра/магістра за спец. 123 «Комп'ютерна інженерія» та 121 «Інженерія програмного забезпечення». – Київ: КПІ ім. Ігоря Сікорського, 2024. – 113 с. – (Серія «Штучний інтелект для індустрії»).

2. Новотарський М.А., Нестеренко Б.Б. Штучні нейронні мережі: обчислення. – Київ: Ін-т математики НАН України, 2004. – 408 с.

3. Субботін С. О. Нейронні мережі : теорія та практика: навч. посіб. – Житомир : Вид. О. О. Євенок, 2020. – 184 с.

4. Терейковський І. А., Бушуєв Д. А., Терейковська Л. О. Штучні нейронні мережі: Базовіположення. Навчальний посібник. Київ: КПІ ім. Ігоря Сікорського, 2022. – 123 с.

5. Mehlig B. Machine learning with neural networks. – Department of Physics University of Gothenburg, Göteborg, Sweden, 2021. – 241 p.

6. Richert W., Coelho L. P. Building Machine Learning Systems with Python. – Packt, 2013. – 290 p.

7. Haykin S. Neural Networks and Learning Machines. – Prentice Hall, 2009. – 906 p.

8. Chollet F. Deep Learning with Python 1st Edition. – Manning, 2017. – 384 p.

Електронне навчальне видання комбінованого використання
Можна використовувати в локальному та мережному режимі

Карєва Валерія Віталіївна

ШТУЧНІ НЕЙРОННІ МЕРЕЖІ

Методичні вказівки
до виконання практичних завдань та
самостійного вивчення теми «Штучні нейронні мережі» з дисципліни
«Алгоритми аналізу даних та штучні нейронні мережі»
для здобувачів вищої освіти другого (магістерського) рівня
за спеціальністю F1 «Прикладна математика»

В авторській редакції

Підписано до розміщення 21.05.2026. Гарнітура Times New Roman.
Ум. друк. арк. 1,5. Обсяг 1,205 Мб. Зам. № 242/26.

Харківський національний університет імені В. Н. Каразіна,
61022, м. Харків, майдан Свободи, 4.
Свідоцтво суб'єкта видавничої справи ДК № 3367 від 13.01.2009
Видавництво ХНУ імені В. Н. Каразіна