НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

«На правах рукопису»

УДК 004.056.5

«До захисту допущено»

Завідувач кафедри
Сергій СТІРЕНКО
(підпис) (ім'я, прізвище)
"____" _____ 2022 р.

Магістерська дисертація

зі спеціальності: *121. Інженерія програмного забезпечення*
(код та назва напряму підготовки або спеціальності)

Спеціалізація: *Інженерія програмного забезпечення комп'ютерних систем*

на тему: Спосіб обробки мультимодальних даних на основі методів штучного інтелекту

Виконав: студент II курсу, групи ІП-03мн
(шифр групи)

Гордієнко Нікіта Юрійович _____
(прізвище, ім'я, по батькові) (підпис)

Науковий керівник к.т.н, доцент Роковий О.П
.
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2022 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет (інститут) *Інформатики та обчислювальної техніки*

(повна назва)

Кафедра *Обчислювальної техніки*

(повна назва)

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою

Спеціальність *121. Інженерія програмного забезпечення*

(код і назва)

Спеціалізація *Інженерія програмного забезпечення комп'ютерних систем*

(код і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри

Сергій СТІРЕНКО

(підпис) (ініціали, прізвище)

« » 2022 р.

ЗАВДАННЯ

на магістерську дисертацію студенту

Гордієнко Нікіта Юрійович

(прізвище, ім'я, по батькові)

1. Тема дисертації Спосіб обробки мультимодальних даних на основі методів штучного інтелекту

Науковий керівник дисертації к.т.н, доцент. Роковий О.П.

(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від « 26» квітня 2022 р. № НС/88/2022

2. Строк подання студентом дисертації _____

3. Об’єкт дослідження методи аналізу мультимодальних даних ЕЕГ зібраних з різних джерел

4. Предмет дослідження створення та порівняння систем для аналізу ЕЕГ даних на основі методів штучного інтелекту

5. Перелік завдань, які потрібно розробити: Розглянути основні новітні практики, що застосовуються для аналізу даних різного роду. Обрати найкращі підходи для аналізу даних з мультиканальних аналогових сенсорів, що залежить від часу. Розглянути основні способи підготовки даних до застосування методів аналізу. Створити систему аналізу з розглянутими механізмами та практиками. Проаналізувати продуктивність та ефективність системи, порівняти їх між собою.

6. Консультанти розділів дисертації:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль			
1			
2			
3			
4			

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів дисертації	Примітка
1	Затвердження теми роботи	17.09.2021	
2	Складання і узгодження технічного завдання	28.10.2021	
3	Написання вступної частини та огляду рішень	20.12.2021	
4	Розробка способу	01.01.2022	
5	Програмна реалізація	01.02.2022	
6	Аналіз отриманих результатів	15.02.2022	
7	Попередній захист та проходження нормоконтролю		
8	Подання МД рецензенту		
9	Захист		

Студент _____ Гордієнко Н.Ю.

(підпис) (ініціали, прізвище)

Науковий керівник дисертації _____ Роковий. О.П

(підпис) (ініціали, прізвище)

РЕФЕРАТ

на магістерську дисертацію

виконану на тему:

Спосіб обробки мультимодальних даних на основі методів штучного інтелекту

студентом: Гордієнко Нікітою Юрійовичем

Робота складається із вступу та чотирьох розділів. Загальний обсяг роботи: 114 аркушів основного тексту, 33 ілюстрацій, 12 таблиць. При підготовці використовувалася література з 73 різних джерел.

Актуальність. З розвитком технологій електроенцефалографії та розвитком нових застосувань для досліджень мозку, зростає необхідність розробки методів аналізу даних, на основі новітніх практик штучного інтелекту, що використовуються в інших сферах, такі як глибинне навчання, штучні нейронні мережі. Нові застосування дають змогу розробляти нові додатки для корисного використання та знаходження хвороб. Нові методи обробки даних можуть зменшити кількість ресурсів необхідних для нових пристроїв, зменшити час навчання та збільшити точність аналізу. Це дозволить створювати більш якісні застосування та зробити їх більш доступними. Отже використання методів штучного інтелекту, а саме штучних нейронних мереж, у даній сфері фактично може відкрити нові можливості для створення більш компактних та масових пристроїв на основі технологій електроенцефалографії.

Мета і завдання дослідження. Метою магістерської роботи є розробка покращень та порівняння модифікацій методів аналізу даних з аналогових сенсорів з декількох каналів.

Для досягнення поставленої магістерською роботою мети було поставлено й вирішено наступні завдання:

- Розглянути основні новітні практики, що застосовуються для аналізу даних різного роду.
- Обрати найкращі підходи для аналізу даних з мультиканальних аналогових сенсорів, що залежить від часу.

- Розглянути основні способи підготовки даних до застосування методів аналізу.
- Створити систему аналізу з розглянутими механізмами та практиками.
- Проаналізувати продуктивність та ефективність системи, порівняти їх між собою.

Об'єктом даного дослідження є методи аналізу мультимодальних даних ЕЕГ зібраних з різних джерел.

Предметом даної роботи є створення та порівняння систем для аналізу ЕЕГ даних на основі методів штучного інтелекту.

Методи досліджень. Для досягнення поставлених в магістерський роботі задач, було використано методи машинного навчання та створення штучних нейронних мереж. Наукова новизна проведеного дослідження забезпечена наступними пунктами:

- Було запропоновано використання рекурентних, згорткових та повнозв’язних шарів у одній архітектурі для кращого використання просторових та часових зв’язків.
- Були використані механізми уваги для покращення точності аналізу даних електроенцефалографії.
- Розроблено декілька унікальних моделей архітектур штучних нейронних мереж, які були порівнянні між собою.
- Тренування проведено на хмарному провайдері Kaggle.

Особистий внесок здобувача. Магістерське дослідження є самостійно виконаною роботою, в якій відображено особистий авторський підхід та особисто отримані теоретичні та прикладні результати, що відносяться до вирішення задачі мозкової активності за допомогою штучних нейронних мереж.

Формулювання мети та завдань дослідження проводилось спільно з науковим керівником.

Практична цінність. Отримані результати можуть бути вільно використані у майбутніх дослідженнях за напрямками:

- Штучні нейронні мережі для аналізу даних електроенцелографії.
- Дослідження зв'язків активності мозку і діяльності людини.
- Практичне використання нейро-комп'ютерного інтерфейсу.

Публікації. Результати дослідження, яке було виконано в рамках дисертації, було прийнято для опублікування, представлено на наступній конференції, та буде опубліковано в збірниках доповідей конференції:

- Hybrid Convolutional, Recurrent and Attention-based Architectures of Deep Neural Networks for Classification of Human-Computer Interaction by Electroencephalography / Nikita Gordienko, Oleksandr Rokovoy, Yuri Gordienko, Sergii Stirenko // 24TH INTERNATIONAL CONFERENCE ON HUMAN-COMPUTER INTERACTION (HCII2022), Gothenburg, Sweden, Lecture Notes in Artificial Intelligence (LNAI), Springer, 2022.
- Convolutional and Recurrent Neural Networks for Physical Action Forecasting by Brain-Computer Interface / K. Kostiukevych, S. Stirenko, N. Gordienko, O. Rokovyi, O. Alienin and Y. Gordienko // 11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), pp. 973-978, doi: 10.1109/IDAACS53288.2021.9660880, 2021.

Ключові слова. Штучні нейронні мережі, електроенцефалографія, інтерфейс мозок комп’ютер, класифікація даних електроенцефалографії, механізм уваги, згорткові мережі, рекурентні мережі.

ABSTRACT

for a master's thesis

made on the topic: Method of processing multimodal data based on artificial intelligence methods

by student: Gordiienko Nikita Yurievich

Master's Thesis. The work consists of an introduction and four chapters. Total amount of work: 114 sheets of the main text, 33 illustrations, 12 tables. Literature from 73 different sources was used in the preparation.

The urgency of the problem. With the development of electroencephalography technologies and the development of new applications for brain research, there is a growing need to develop data analysis methods based on the latest practices of artificial intelligence used in other areas, such as deep learning, artificial neural networks. New applications make it possible to develop new applications for the useful applications and detection of diseases. New data processing methods can reduce the amount of resources required for new devices, reduce training time and increase the accuracy of analysis. This will create better applications and make them more common. Thus, the use of artificial intelligence methods, especially artificial neural networks, in this area may actually open up new opportunities for the creation of more compact and mass devices based on electroencephalography technologies.

The purpose and objectives of the study. The aim of the master's thesis is to develop and compare modifications of data analysis methods from analog sensors from several channels.

To achieve the goal set by the master's thesis, the following tasks were set and solved:

- Consider the main new practices used to analyze data of various kinds.
- Choose the best approaches for time-dependent analysis of data from multi-channel analog sensors.
- Consider the main ways to prepare data for the use of analysis methods.
- Create an analysis system with the considered mechanisms and practices.

- Analyze the performance and efficiency of the systems, compare them with each other.

Object of the study. The object of this study are methods of analysis of multimodal EEG data collected from different sources.

Subject of study. The subject of this work is the creation and comparison of systems for EEG data analysis based on artificial intelligence methods.

Research methods and Scientific Novelty. To achieve the objectives set in the master's thesis, the methods of machine learning and the creation of artificial neural networks were used. The scientific novelty of the study is provided by the following points:

- It was proposed to use recurrent, convolutional and fully connected layers in one architecture to make better use of spatial and temporal correlations.
- Attention mechanisms have been used to improve the accuracy of electroencephalography data analysis.
- Several unique models of artificial neural network architectures have been developed and compared.
- The training was performed on the cloud provider Kaggle.

Personal contribution of the applicant. The master's research is a self-performed work, which reflects the personal author's approach and personally obtained theoretical and applied results related to solving the problem of brain activity using artificial neural networks.

The formulation of the purpose and objectives of the study was carried out jointly with the supervisor.

Practical value. The obtained results can be freely used in future research in the following areas:

- Artificial neural networks for electroenephalography data analysis.
- Research of connections between brain activity and human activity.
- Practical use of neuro-computer interface.

Publications. The results of the research, which was performed within the dissertation, were presented at the following conferences, and published in the proceedings of the conference:

- Hybrid Convolutional, Recurrent and Attention-based Architectures of Deep Neural Networks for Classification of Human-Computer Interaction by Electroencephalography / Nikita Gordienko, Oleksandr Rokovoy, Yuri Gordienko, Sergii Stirenko // 24TH INTERNATIONAL CONFERENCE ON HUMAN-COMPUTER INTERACTION (HCII2022), Gothenburg, Sweden, Lecture Notes in Artificial Intelligence (LNAI), Springer, 2022.
- Convolutional and Recurrent Neural Networks for Physical Action Forecasting by Brain-Computer Interface / K. Kostiukevych, S. Stirenko, N. Gordienko, O. Rokovyi, O. Alienin and Y. Gordienko // 11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), pp. 973-978, doi: 10.1109/IDAACS53288.2021.9660880, 2021.

Keywords. Artificial neural networks, electroencephalography, computer brain interface, electroencephalography data classification, attention mechanism, convolutional networks, recurrent networks.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 7

ВСТУП 10

РОЗДІЛ 1. ПРИРОДА ТА МЕТОДИ АНАЛІЗУ МОЗКОВОЇ АКТИВНОСТІ 12

1.1. Коротка інформація про електроенцефалографія 12

1.2. Процес вироблення токів мозком 13

1.2.1. Мікроскопічний рівень 14

1.2.2. Мезоскопічний рівень 14

1.2.3. Макроскопічний рівень 15

1.3. Огляд технічних засобів для вимірювання 16

1.4. Методи аналізу 19

1.4.1. Мультимодальні дані 19

1.4.2. Частотний аналіз 21

1.4.3. Часово-частотний аналіз 22

1.4.4. Вейвлети 23

1.4.5. Методи штучного інтелекту 25

1.5. Використання у різних сферах 26

ВИСНОВКИ ДО РОЗДІЛУ 1 29

РОЗДІЛ 2. АНАЛІЗ ЗАСОБІВ ОБРОБКИ МУЛЬТИМОДАЛЬНИХ ДАНИХ НА ОСНОВІ МЕТОДІВ ШТУЧНОГО ІНТЕЛЕКТУ 31

2.1. Штучний інтелект та машинне навчання 31

2.1.1. Навчання з учителем 34

2.1.2. Навчання без учителя 35

2.1.3. Напівавтоматичне навчання 35

2.1.4. Навчання з підкріпленням 36
2.1.5. Глибинне навчання 36
2.2. Згорткова штучна нейронна мережа 39
2.2.1. Згорткові шари 40
2.2.1.1. Рецептивне поле 40
2.2.1.2. Ваги 41
2.2.1.3. Особливості використання згорткових шарів 41
2.2.1.4. Характеристики згорткових шарів 43
2.2.1.4.1. Глибина 44
2.2.1.4.2. Крок 44
2.2.1.4.3. Заповнення 44
2.2.1.5. Підвибірка (Pooling layer) 44
2.2.1.6. Функції активації 46
2.2.1.6.1. Sigmoid 47
2.2.1.6.2. Tanh 48
2.2.1.6.3. ReLU 49
2.2.1.6.4. Softmax 49
2.2.1.7. Функції втрат 50
2.2.1.7.1. Середня квадратична помилка 50
2.2.1.7.2. Перехресна ентропія 51
2.2.1.7.3. Softmax 52
2.2.1.8. Зменшення розмірності 52
2.2.3. Повнозв’язний шар 53
2.2.4. Популярні архітектури згорткових мереж для задач класифікації на зображеннях 53

2.2.4.1. AlexNet 53
2.2.4.2. LeNet-5 55
2.2.4.3. VGG16 56
2.3. Рекурентна нейронна мережа 58
2.3.1. Рекурентна нейронна мережа проти нейронної мережі з прямим зв'язком 58
2.3.2. Довга короткострокова память 60
2.3.3. Вентильний рекурентний вузол 61
2.4. Трансформер 62
2.4.1. Механізм уваги 64
2.5. Датасети 66
2.5.1. Датасет ЕЕГ 67
Висновки до розділу 2 72
РОЗДІЛ 3. РОЗРОБКА МЕТОДІВ АНАЛІЗУ МУЛЬТИМОДАЛЬНИХ ДАНИХ ЕЕГ НА ОСНОВІ ШТУЧНИХ НЕЙРОННИХ МЕРЕЖ 74
3.1. Інструменти розробки та створення методів AI 74
3.2. Процес навчання моделі 75
3.3. Підготовка до розробки моделей 77
3.3.1. Аналіз структури ЕЕГ даних та підготовка даних 77
3.3.2. Нормалізація даних 78
3.3.3. Аналіз метаданих датасету 79
3.3.4. Створення генераторів 81
3.4. Огляд структури даних та можливих моделей використання кожної з моделей 81
3.4.1. Згорткові мережі 81
3.4.2. Рекурентні архітектури 82

3.4.3. Механізм уваги 83
3.5. Опис розроблених моделей та модифікацій 84
3.5.1. Рекурентні архітектури 84
3.5.2. Рекурентні архітектури з прихованими станами 84
3.5.3. Рекурентні архітектури з прихованими станами та механізмом уваги 84
3.5.4. Рекурентні архітектури зі згортковими шарами 85
3.5.5. Рекурентні архітектури зі згортковими шарами та прихованими станами 85
3.5.6. Рекурентні архітектури зі згортковими шарами та прихованими станами з механізмом уваги 85
3.5.7. Рекурентні архітектури зі згортоквими шарами до і після 86
3.5.8. Використання популярних згорткових архітектур 86
3.6. Методи порівняння моделей 87
Висновки до розділу 3 90
Розділ 4 92
АНАЛІЗ ТА ПОРІВНЯННЯ ОТРИМАНИХ РЕЗУЛЬТАТІВ 92
4.1. Аналіз отриманих результатів за різними модифікаціями між архітектурами 92
4.1.1. Рекурентні архітектури 92
4.1.2. Рекурентні архітектури з прихованими станами 93
4.1.3. Рекурентні архітектури з прихованими станами та механізмом уваги 94
4.1.4. Рекурентні архітектури зі згортковими шарами 96
4.1.5. Рекурентні архітектури зі згортковими шарами та прихованими станами 97

4.1.6. Рекурентні архітектури зі згортковими шарами та прихованими станами з механізмом уваги 99

4.1.7. Рекурентні архітектури зі згортковими шарами до і після 100

4.1.8. Використання популярних згорткових архітектур 102

4.2. Аналіз отриманих результатів за різними рекурентними архітектурами між модифікаціями 103

4.2.1. SimpleRNN 103

4.2.2. LSTM 104

4.2.3. GRU 105

4.4. Порівняння всіх моделей 106

Висновки до розділу 4 110

ВИСНОВКИ 112

ПЕРЕЛІК ПОСИЛАНЬ 115

ДОДАТОК А. Мережа на основі найбільш розповсюджені архітектури RNN. 121

ДОДАТОК Б. Мережа на основі найбільш розповсюджені архітектури RNN з використанням прихованих станів 122

ДОДАТОК В. Мережа на основі найбільш розповсюджені архітектури RNN з використанням прихованих станів та механізмом уваги. 123

ДОДАТОК Г. Мережа на основі розповсюджених архітектур RNN з використанням згорткових шарів до рекурентних 124

ДОДАТОК Д. Мережа на основі розповсюджені архітектур RNN з використанням згорткових шарів до рекурентних та використанням прихованих станів рекурентної мережі 125

ДОДАТОК Е. Мережа на основі розповсюджених архітектур RNN з використанням згорткових шарів до рекурентних та використанням

прихованих станів рекурентної мережі з використанням механізму уваги 126

ДОДАТОК Є. Мережа на основі розповсюджени архітектур RNN з використанням згорткових шарів до рекурентних та використанням згорткових шарів після рекурентної мережі з використанням прихованих станів перед повнозв'язними шарами 127

ДОДАТОК Ж. Результати моделей та їх модифікацій 128

ДОДАТОК З. Код програми 129

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

*RNN*_Dense - поєднання рекурентної нейронної мережі з повнозв’язними шарами.

*RNN*_Hidden_Attention_Dense - поєднання рекурентної нейронної мережі, прихованих станів та механізма уваги з повнозв’язними шарами.

*RNN*_Hidden_Dense - поєднання рекурентної нейронної мережі та прихованих станів з повнозв’язними шарами.

ЕЕГ - електроенцефалографія.

AI (англ. artificial intelligence) - штучний інтелект.

AlexNet - згорткова нейронна мережа, що застосовується для розпізнавання зображень.

Attention - механізм уваги.

AUC (англ. area under curve) - статистичний показник, площа, обмежена деякою кривою та віссю абсцис.

AvgPool (англ. average pooling) - середня підвибірка.

BCI (англ. brain computer interface) - нейро-комп'ютерний інтерфейс.

BERT (англ. Bidirectional Encoder Representations from Transformers) - двоспрямовані кодувальні представлення з трансформерів.

BPTT (англ. Backpropagation through time) - зворотне поширення в часі.

BRNN (англ. bidirectional recurrent neural networks) - двонаправлені рекурентні нейронні мережі.

CIFAR-10 - набір даних зображень.

CNN (англ. convolutional neural network, ConvNet) - згорткові нейронні мережі.

CNN_*RNN*_CNN_Dense - поєднання рекурентної нейронної мережі зі згортковими шарами до і після та повнозв’язними шарами.

CNN_*RNN*_Dense - поєднання рекурентної нейронної мережі з повнозв’язними шарами після і згортковими шарами до.

CNN_*RNN*_Hidden_Attention_Dense - поєднання рекурентної нейронної мережі, прихованих станів та механізма уваги з повнозв’язними шарами після і згортковими шарами до.

CNN_*RNN*_Hidden_Dense - поєднання рекурентної нейронної мережі та прихованих станів з повнозв’язними шарами після і згортковими шарами до.

CPU (англ. Central processing unit) - центральний процесор.

CUDA (англ. Compute Unified Device Architecture) - програмно-апаратна архітектура паралельних обчислень.

CUDNN (англ. CUDA® Deep Neural Network library) - бібліотека глибинної нейронної мережі створена командою NVIDIA.

CV (англ. cross validation) - техніка перехресної перевірки.

Dataset - набір даних.

Dense layer - повнозв’язний нейронний щар.

FPR (англ. false positive rate) - хибно позитивний відсоток.

Framework - програмна платформа.

FT (англ. Fourier transform) - перетворення Фур'є.

GA (англ. genetic algorithm) - генетичний алгоритм.

GAL (англ. Grasp and lift) - набір даних з ЕЕГ даних, які записувались при схоплення та підніманні предметів.

GPT (англ. Generative Pre-trained Transformer) - Породжувальний попередньо натренований трансформ ер.

GPU (англ. graphics processing unit) - графічний процесор.

GRU (англ. Gated recurrent units) - вентильні рекурентні вузли.

Hidden states - приховані стани.

ImageNet (англ. image network) - набір даних на основі зображень

LeNet-5 - згорткова нейронна мережа, що застосовується для розпізнавання зображень.

LSTM (англ. long short-term memory) - довга короткочасна пам'ять.

MaxPool (англ. maximum pooling) - максимальна підвибірка.

MDP (англ. Markov decision process) - марковські процеси вирішування.

MSE (англ. mean squared error) - середньоквадратична похибка.

NLP (англ. Natural language processing) - обробка природної мови.

One-hot encoding - унітарний код.

ONEIROS (англ. Open-ended Neuro-Electronic Intelligent Robot Operating System) - дослідницький проєкт створення відкритого операційної системи для машинного навчання.

PCA (англ. principal component analysis) - метод головних компонент/

ReLU (англ. rectified linear unit) - зрізаний лінійний вузол або випрямлений лінійний вузол.

RGB (англ. red, green, blue) - адитивна колірна модель.

RNN (англ. recurrent neural networks) - рекурентні нейронні мережі.

ROC (англ. receiver operating characteristic) - робоча характеристика приймача, графік, що дозволяє оцінити якість бінарної класифікації.

Sigmoid – сигмоїда.

Simple RNN (англ. simple recurrent neural networks) - архітектура рекурентної нейронної мережі у Keras.

Softmax - нормована експоненційна функція.

STFT (англ. Short-time Fourier transform) - віконне перетворення Фур'є або короткочасне перетворення Фур’є.

Tanh (англ. hyperbolic tangent) - гіперболічний тангенс.

TPR (англ. true positive rate) - істинно позитивний відсоток.

TPU (англ. tensor processing unit) - тензорний блок обробки.

VGG-16 - згорткова нейронна мережа, що застосовується для розпізнавання зображень.

ВСТУП

Останніми роками дуже інтенсивно розвиваються технології збору та аналізу біометричних даних великого обсягу. Сучасні засоби носимої електроніки (wearable electronics), наприклад розумні годинники дозволяють збирати біометричні дані різного типу про фізичну діяльність людини: параметри рухомої діяльності (ходьбу), серцево-судинної діяльності (параметри серцебиття людини), тощо. Після їх обробки сучасними методами користувачі мають змогу дізнатись про потенційні проблеми зі здоров'ям такі як вади ходи, проблеми з серцем тощо та можуть порадити звернутись до лікаря. З'являються нові більш складні і водночас більш доступні сенсори, наприклад, сенсори на основі електроенцефалографії (ЕЕГ), які дозволяють отримувати інформацію про стан здоров'я набагато більшого обсягу. У останні роки нові інтерфейси мозок комп'ютер (BCI) стають більш масовими та можуть більш широко застосовуватись для отримання інформації про стан здоров'я з багатьох джерел інформації на основі мультимодальних даних.

Останні дослідження мозку за допомогою використання альфа хвиль, які є результатом активності мозку, є темою багатьох досліджень задля майбутнього використання у практичних цілях. Деякі наукові роботи присвячені накопиченню даних та створенню наборів даних під час різних видів активності, методам точного розмічення та аналізу цих даних. Накопичені дані дають змогу розробляти методи обробки та аналізу цих даних для подальшого корисного застосування у вигляді застосунків та пристроїв. Разом з тим існує проблема у покращенні методів аналізу та розпізнавання даних альфа хвиль утворених мозком, наприклад на основі наявних та нових методів штучного інтелекту. Існує багато стандартних методів штучного інтелекту для вирішення деяких вже стандартних задач (комп'ютерний зір, розпізнавання об'єктів, генерація тексту тощо). Але поки не існує стандартних методів штучного інтелекту для вирішення проблеми аналізу даних ЕЕГ (альфа хвиль) з різних регіонів мозку для подальшого застосування у медицині та поліпшення наявних технологій обробки медичних даних ЕЕГ.

Тема є актуальною тому що нові методи обробки даних можуть зменшити кількість ресурсів необхідних для нових пристроїв, зменшити час навчання та збільшити точність аналізу. Це дозволить створювати більш якісні застосування та зробити їх більш доступними. ЕЕГ дані збираються з високою частотою з великою кількістю електродів (каналів), що створює великі набори даних. Статистичний аналіз не є ефективним, тому що дані отримані з різних каналів є схожими, а дані що містять корисну інформацію наявні у малій кількості відносно зібраних даних. Використання методів штучного інтелекту у даній сфері дає змогу використати цю корисну інформацію з невеликої кількості корисних даних і може дозволити передбачувати рухи на основі попередньої активності мозку. Якість аналізу і класифікації дій за допомогою даних про мозкові хвилі ускладнюється наявними шумами створеними іншою активністю мозку. Наявні просторові зв’язки між даними отриманими з електродів у різних частинах мозку та їх часові зв’язки можуть бути використані різними видами штучних нейронних мереж та їх поєднаннями для отримання більш точного результату. Поєднання згорткових, рекурентних, повнозв’язних штучних нейронних мереж з механізмами уваги можуть використовуватись як інноваційні методи аналізу мультимодальних даних ЕЕГ.

Отримані результати можуть бути вільно використані у майбутніх дослідженнях залежності активності мозку від дій, що виконує пацієнт. Також розроблені моделі можуть бути використані у нових компактних нейро-комп'ютерних інтерфейсах для аналізу даних у реальному часі. Отримані результати можуть бути застосовані для розробки нових унікальних архітектур штучних нейронних мереж для більш ефективного аналізу просторових та часових зв’язків даних електроенцефалографії.

РОЗДІЛ 1
ПРИРОДА ТА МЕТОДИ АНАЛІЗУ МОЗКОВОЇ АКТИВНОСТІ

1.1. Коротка інформація про електроенцефалографія

Електроенцефалограма (ЕЕГ) - це тест, який використовується для оцінки електричної активності вашого мозку. Клітини мозку спілкуються один з одним за допомогою електричних імпульсів. ЕЕГ може бути використаний для виявлення потенційних проблем, пов’язаних з діяльністю мозку.

ЕЕГ відстежує та записує моделі мозкових хвиль. Маленькі плоскі металеві диски, які називаються електродами, прикріплені до шкіри голови за допомогою дротів. Електрод – це провідник, по якому входить або виходить електричний струм. Електроди передають інформацію від мозку пацієнта до комп'ютера, який записує та аналізує дані. Електроди вимірюють електричні імпульси створені мозком і посилають сигнали на комп’ютер, який записує результати, що будуть використані у майбутньому [1].

Електричні імпульси в записі ЕЕГ дозволяють лікарям швидко оцінити, чи є аномальні моделі. Порушення можуть бути ознакою судом або інших мозкових розладів .

ЕЕГ використовується з 1929 року для виявлення проблем в електричній активності мозку, які пов’язані з певними порушеннями мозку. Вимірювання, дані ЕЕГ, використовуються для підтвердження або виключення різних станів, включаючи [2]:

- судомні розлади (наприклад, епілепсія)
- травма голови
- енцефаліт (запалення головного мозку)
- пухлина мозку
- енцефалопатія (захворювання, що викликає дисфункцію мозку)
- порушення сну
- інсульт
- деменція

Коли пацієнт знаходиться в комі, може бути проведена ЕЕГ, щоб визначити рівень його мозкової активності. Тест також можна використовувати для контролю активності під час операції на мозку.

Спеціалісти проводять ЕЕГ в лікарнях, кабінетах лікарів і лабораторіях. Тест зазвичай триває приблизно від 30 до 60 хвилин. Під час тесту між електродами та шкірою проходить невелика кількість струму. У деяких випадках людині можуть пройти 24-годинну ЕЕГ. Ці ЕЕГ використовують відео для фіксації нападів. ЕЕГ може показати відхилення, навіть якщо судоми не виникають під час тесту [3].

Також існують дослідження та розробки для допомоги людям з інвалідністю за допомогою пристроїв, що зчитують мозкову активність та дозволяють керувати зовнішніми контролерами чи аналізувати поведінку [4]. Деякі розробки, навпаки стимулюють нейрони для отримання бажаного результату.

1.2. Процес вироблення токів мозком

Нейронні коливання (також мозкові хвилі) - моделі нервової активності в центральній нервовій систем, що повторюються у певному ритмі. Нервова тканина генерує коливальну активність багатьма способами. Головними інструментами є реакції в окремих нейронах або взаємодія між нейронами. Коливання можуть бути виявлені у формі мембранного потенціалу або у формі ритмічних моделей потенціалів. Воним можуть викликати осциляційну активацію постсинаптичних нейронів. Синхронна активація багатьох нейронів у вигляді нейронних ансамблів викликає макроскопічні коливання, які можна спострегати за допомогою ЕЕГ даних. Така активність у групах нейронів створена зворотнім зв’язком між нейронами і вона створює синхронізацію моделей активації. Взаємодія між нейронами може викликати коливання з частотою, відмінною від частоти спрацювання окремих нейронів через їх створену групово активність[5].

. Широко визнані три різні рівні: мікромасштаб (активність окремого нейрона), мезомасштаб (активність локальної групи нейронів) і макромасштаб (активність різних областей мозку)[6] Коливальну діяльність можна спостерігати на всіх цих масштабах по всій області неровової системи.

1.2.1. Мікроскопічний рівень

Нейрони генерують потенціали дії в результаті зміни потенціалу електричної мембрани. Нейрони можуть генерувати безліч потенціалів дії в послідовності, утворюючи так звані ланцюги спайків. Ці потяги є основою для нейронного кодування та передачі інформації в мозку. Шипоподібні потяги можуть утворювати всілякі моделі, наприклад, ритмічні стрибки та розриви, і часто демонструють коливальну активність.[7] Коливальну активність в окремих нейронах можна також спостерігати при підпорогові флуктуації мембранного потенціалу. Ці ритмічні зміни мембранного потенціалу не досягають критичного порогу і тому не призводять до потенціалу дії. Вони можуть бути результатом постсинаптичних потенціалів від синхронних входів або внутрішніх властивостей нейронів.

Нейронні спайки можна класифікувати за моделями їхньої активності. За збудливістю нейрони можна розділити на I і II клас. Нейрони класу I можуть генерувати потенціали дії з як завгодно низькою частотою залежно від вхідної сили, тоді як нейрони класу II генерують потенціали дії в певному діапазоні частот, який відносно нечутливий до змін вхідної сили.[8] Нейрони II класу також більш схильні відображати підпорогові коливання мембранного потенціалу.

1.2.2. Мезоскопічний рівень

Група нейронів також може генерувати коливальну активність. Завдяки синаптичним взаємодіям моделі спрацьовування різних нейронів можуть стати синхронізованими, і ритмічні зміни електричного потенціалу, викликані їх потенціалами дії, будуть суміщатися (конструктивна інтерференція). Тобто

синхронізовані моделі спрацьовування призводять до синхронізованого введення в інші області кори, що викликає коливання локального поля з великою амплітудою. Ці великомасштабні коливання можна також виміряти за межами шкіри голови за допомогою ЕЕГ. Електричні потенціали, які генеруються окремими нейронами, занадто малі, щоб їх можна було отримати за межами шкіри голови, а активність ЕЕГ завжди відображає підсумовування синхронної активності тисяч або мільйонів нейронів, які мають подібну просторову орієнтацію [5].Нейрони в нейронному ансамблі рідко всі спрацьовують в один і той же момент, тобто повністю синхронізовані. Натомість ймовірність спрацьовування ритмічно модулюється так, що нейрони, швидше за все, запускатимуться одночасно, що викликає коливання їх середньої активності. Таким чином, частота великомасштабних коливань не обов’язково повинна відповідати схемі спрацювання окремих нейронів. З іншої сторони, якщо велика група нейронів ритмічно модулює хвилі на загальній частоті, вони з великою ймовірність будуть генерувати коливання в середньому полі [7]. Нейронні ансамблі можуть генерувати коливальну активність ендогенно через локальні взаємодії між збудливими та гальмівними нейронами. Зокрема, гальмівні інтернейрони відіграють важливу роль у створенні синхронності нейронного ансамблю, створюючи вузьке вікно для ефективного збудження та ритмічно модулюючи швидкість спрацьовування збудливих нейронів [9].

1.2.3. Макроскопічний рівень

Нейронні коливання також можуть виникати внаслідок взаємодії між різними ділянками мозку, пов’язаними через структурний коннектом, структурою зв'язків у нервовій системі організм. Важливу роль тут відіграють затримки часу. Оскільки всі ділянки мозку двонаправлено пов’язані, ці зв’язки між ділянками мозку утворюють петлі зворотного зв’язку. Позитивні петлі зворотного зв’язку, як правило, викликають коливальну активність, де частота обернено пропорційна часу затримки. Таламокортикальні випромінювання –

це зв’язки між корою і таламусом. Таламокортикальні випромінювання є гарним прикладом такої петлі зворотного зв’язку. Мережа на основі цих зв’язків здатна утворювати активність у формі коливань. Ця активність називається повторюваним таламо-корковим резонансом [10]. Вона є важливим елементом у процесі генерації альфа хвиль головним мозком. У моделі мережі всього мозку з реалістичною анатомічною зв’язністю та затримками поширення між областями мозку хвилі бета частот створені внаслідок часткової синхронізації підмножин областей мозку, які коливаються в гамма-діапазоні (генеруються на мезоскопічному рівні) [11].

1.3. Огляд технічних засобів для вимірювання

У зв’язку зі збільшеним використанням і попитом на високоякісні пристрої ЕЕГ, зараз існує безліч компаній, які можуть задовольнити конкретні потреби користувачів ЕЕГ. Пристрої відрізняються кількістю каналів, розміром (стаціонарний чи портативний пристрій), пропоновані попередньо визначені показники та ціною. Багато пристроїв у верхній частині цінового діапазону є особливо передовими, дослідницькими пристроями, які забезпечують неймовірну чутливість, а також велику кількість датчиків. Найбільш поширеними є дешеві рішення, які частіше за все використовуються у розвагах, тренувальних іграх для реабілітації тощо. Через свою низьку якість та точність вони не можуть бути використані у дослідженнях мозку, але підходять для простих задач. Апарати з більшою кількістю каналів та більшою частотою вимірювань, зазвичай мають вищу цінову категорію і можуть використовуватись у наукових роботах та експериментах, через свою доступність та легкість у використанні. Діагностичні апарати мають найбільшу цінову категорію. Зазвичай вони стаціонарні, вимагають використання професіоналами і використовуються у медичних дослідженнях та діагностиці через свою точність.

Одним з пристроїв є MindFlex [12]. Це ЕЕГ апарат, основна задача якого є використання технології ThinkGear, розробки NeuroSky для мозкових

тренувань під час реабілітації. Гарнітура бездротова підключається до блоку керувань, що підвищує мобільність пацієнта (рис. 1.1). Пристрій живиться за допомогою трьох 1,5 В батарейок. Металевий електрод, що знаходиться на чолі, виявляє мозкові хвилі. Інший електрод на мочці є нульовою точкою. Обчислення та аналіз на блоці обробки сигналів може визначити рівень концентрації та уваги. Дані передаються за допомогою бездротових протоколів. Деякі ігри використовують цей пристрій як контролер (наприклад, MindFlex Duel метою якої є використання активності мозку за допомогою думок та емоції для перемоги над суперником [13]).

Рис. 1.1. Вигляд бездротовиї гарнітури MindFlex з 2 електродами, одним на чолі іншим на мочці

На ринку існують інші мобільні пристрої, що використовувались для досліджень [14]. Однією з таких систем з відкритим кодом та можливістю для модифікацій є Ultracortex Mark 4 з розширенням Cyton Board та Daisy від OpenBCI (рис. 1.2.) [15]. Ця ситтема може використовувати одночасно 8 або 16 каналів в залежності від обраної частоти. На вибір є 2 режими робити замірів використовуючи або 8-каналів з частотою 250 ГГц, або 16-каналів з частотою 125 ГГц для збору даних ЕЕГ. Однією з переваг системи є відносно низька вартість за систему з відкритим кодом та простором для модифікацій. Для передачі даних з гарнітури використовується протокол Bluetooth, що також підвищує мобільність пацієнта. Канали можна розміщувати вільно на каркасі, роблячи заміри в різних регіонах мозку. У системі використані сухі електроди, які зменшують час для підготовки експерименту. З недоліків є невелика

точність через використання технології сухих електродів. Порівняння переваг та недоліків сухих електродів на відміну від вологих є у наступних дослідженнях [16].

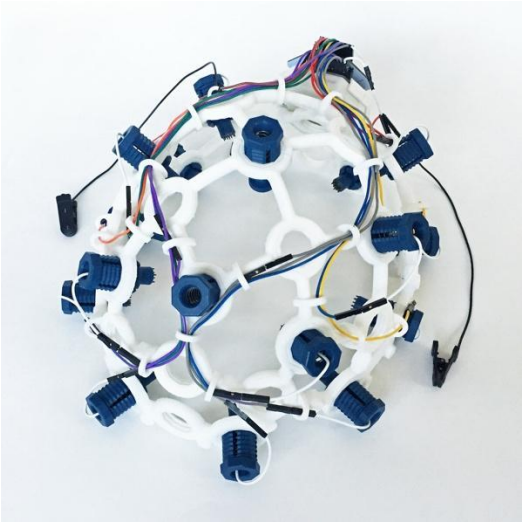


Рис. 1.2. Гарнітура ЕЕГ Ultracortex Mark 4 з каракасом на якому можуть бути розташовані до 16 електродів у різних регіонах мозку

Новий універсальний лабораторний підсилювач «все в одному» від компанії Brain Products (Рис. 1.3). Пристрій actiCHamp Plus пропонує масштабоване та гнучке рішення для лабораторних записів ЕЕГ. actiCHamp Plus можна комбінувати з усіма активними та пасивними електродними системами - в т.ч. R-Net і живиться від міцної літій-іонної батареї PowerUnit [17].

Переваги, які висвітлює виробник:

- Сумісний з усіма активними та пасивними електродами
- Можливість масштабування для записів високої щільності
- Проста синхронізація для кількох способів запису
- Потужний літій-іонний акумулятор, тобто можливе використання і без постійного живлення
- Інтеграція з EEGLAB, MATLAB®, LSL, OpenViBE
- Масштабовані рішення
- Модульність для запису від 32 до 160 каналів

- 8 додаткових каналів AUX для включення додаткових фізіологічних датчиків
- Частота дискретизації до 100 кГц

Це є професіональне спеціалізоване рішення для використання у лабораторіях спеціалістами. Компанія виробляє різні види електродів та інших датчиків і сенсорів, що сумісні з апаратом.

Рис. 1.3. Пристрій ЕЕГ компанії Brain Products

1.4. Методи аналізу

Для аналізу даних ЕЕГ використовуються різні методи. Порівняння різних видів аналізу будуть наявні у цьому розділі. Деякі з методів є застарілими (такі як, частотний аналіз), а деякі навпаки новітніми (такі як машинне навчання та нейронні мережі).

1.4.1. Мультимодальні дані

Дані доступні в широкому діапазоні модальностей, від візуальних даних у вигляді зображень і відео, мовних даних у вигляді тексту та мови, аудіо даних, таких як музика та звуки, а також інших сенсорних даних, таких як запах, смак. Крім того, дані часто надходять у формі, яка перетинає кілька модальностей, таких як відеоматеріал, який зазвичай включає принаймні зображення та звук (мову, музику), а також текст у вигляді субтитрів, які

можуть використовувати мову, що відрізняється від мови яка використовується для аудіо даних.

Інші параметри також надають дані в різних режимах, як-от відчуття людини, в якому, наприклад, дані про вираз обличчя у вигляді зображень можна об’єднати зі слуховими (мова, звук), тактильними (дотик) або іншими сенсорними даними.

Дані зібрані з різних джерел визначаються як мультимодальні, так як вони мають різну природу і доповнюють один одного. Злиття даних з різних датчиків - це поєднання даних, отриманих з різних джерел, таким чином, що отримана інформація має меншу невизначеність, ніж коли ці джерела використовувалися окремо. Зменшення невизначеності означає можливість більш точного, більш повного або більш надійного аналізу. Наприклад, стереоскопічне бачення в результаті якого обчислення інформації про глибину за допомогою комбінування двовимірних зображень з двох камер при трохи різних точках огляду дає більш повну картину [18].

Існує декілька видів поєднання мультимодальних даних, більш відома як синтез. Прямим ссинтезом називають злиття і поєднання даних сенсорів із датчиків однакових або датчиків однакового типу, м’яких датчиків та значень вже записаних сенсорних даних. Непрямий синтез може використовувати для поєднання дані про навколишнє середовище чи вхідні дані від користувача. Цей процес також називають мультисенсорним синтезом і належить до синтезу інформацію. Таким чином дані ЕЕГ є мультимодальними через збір даних з множини електродів, знання про час навколишнього середовища та вхідні дані користувача про експеримент.

На прикладі ЕЕГ різними датчиками можуть бути час з годинника. В професіональних лабораторіях одночасно з ЕЕГ можуть використовуватися інші пристрої для вимірювання серцевого ритму, тиску, рівня кисню в крові, акселерометри тощо. Тоді мультимодальними даними всі ці дані, а також введені користувачем дані. Користувач чи спостерігач може доповнити дані власними спостереженнями (наприклад, виділивши різні види активності

пацієнта та занести ці дані у базу даних. Також додаткові даних з акселерометрів на користувачі можуть використовуватись. Вони можуть бути корисними для більш точного розмічення даних для подальшого аналізу методами штучного інтелекту.

1.4.2. Частотний аналіз

Дані ЕЕГ можуть бути представлені у різних формах. Може використовуватись форма часових послідовностей. Також може бути корисним розглянути дані у частотному вимірі для цього сигнал розкладають на синуси та косинуси. Мета частотного виміру полягає в описі сигналу за допомогою розкладання на синусоїди різної частоти. Тобто кожний сигнал можна представити як суперпозицію трьох синусоїд різної частоти.

Перетворення Фур’є (FT) — перетворення, яке розкладає отриману функцію на окремі частотні компоненти. Прикладом застосування є розкладання форми хвилі музичного акорду в термінах інтенсивності його складових висот [19].

Результатом перетворення Фур’є функції є комплексна функція, що представляє комплексні синусоїди, за допомогою яких створена вихідна функція. Для кожної частоти величина (абсолютне значення) комплексного значення представляє амплітуду складової комплексної синусоїди з цією частотою, а аргумент комплексного значення представляє зміщення фази цієї комплексної синусоїди. Якщо частоти немає, перетворення має значення 0 для цієї частоти. Перетворення Фур’є не обмежується функціями часу, але область початкової функції зазвичай називається часовою областю. Теорема про інверсію Фур’є забезпечує зворотне перетворення Фур’є, яке синтезує вихідну функцію з її представлення в частотній області.

Основна формула перетворення Фур'є має вигляд:

$$F(w) = \int_{-\infty}^{\infty} f(t)e^{-iwt}dt \quad (1.1)$$

1.4.3. Часово-частотний аналіз

Перетворення Фур’є розкладає сигнал на певні частотні компоненти. Це є одним з недоліків такого підходу для аналізу даних. Час є важливою характеристикою, яка втрачається у такому підході. Під час перетворення Фур’є сигнал розглядається як нерухомий і як результат активність різних частот вважається постійною протягом усієї довжини сигналу. Але природа багатьох сигналів вказує на те що час є важливим фактором і функції змінюються з часом. Наприклад, такі поширені та зрозумілі речі як музика чи мова. Схожим за природоє є і сигнал зібраний за допомогою ЕЕГ. Тому перетворення Фур’є є неефективним. Процеси у мозку швидко змінюються у часі і не мають постійною функції. Наприклад, мозок може реагувати на зовнішні подразники іншими регіонами викликаючи нову реакцію.

Негативний ефект цього недоліку можна зменшити розділивши сигнал на певні частини. Таким чином буде подолана нестача часової характеристики при перетворенні Фур’є. Обчисливши спектр потужності для кожного зі шматків можна фокусуватись на різних часових проміжках даних. Також у більш продвинутих алгоритмах використовують спеціальне вікно, що рухається з якимось кроком. Такий підхід називається короткочасним перетворенням Фур'є (STFT) або віконним перетворенням Фур'є.

Це перетворення, на основі перетворення Фур’є, застосовується для визначення синусоїдальних компонент частоти та фазового вмісту сигналу, який залежить і змінюється у часі [20]. При застосуванні сигнал поділяється на менші за розміром сегменти сигналу, що мають однакову довжину. Для кожнго з цих сегментів обчислюється перетворення Фур’є окремо. Таким чином ортимано спектр Фур’є на кожному з сегментів. На основі цих спектрів будується графік як функція часу, яка називається спектрограмою.

Одним з недоліків такого підходу є те, що вікно має фіксований розмір отже роздільна здатність є фіксованою. Обрання ширини віконної функції залежить від того як представлений сигнал. Більш широке вікно дає кращу частотну роздільну здатність, але більш низьку роздільну здатність за часом.

Навпаки, менше вікно погіршує частотну роздільну здатність, але дає хорошу роздільну здатність за часом. Перетворення з використаннями вікон таких розмірів називаються вузькосмуговими та широкосмуговими перетвореннями відповідно. Така залежність та неможливість отримати хорошу роздільну здатність за цими характеристиками одночасно називається принципом невизначеності аналізу сигналів. Частота і час не можуть бути зроблені довільно малими одночасно через свою залежність. Точна локалізація у часі та частоті взаємовиключна, оскільки для визначення частоти потрібно кілька точок даних.

Це одна з причин створення вейвлет-перетворення та аналізу з багатьма роздільною здатністю, які можуть дати хорошу роздільну здатність за часом для високочастотних подій і хорошу роздільну здатність по частоті для низькочастотних подій, комбінація, найкраще підходить для багатьох реальних сигналів.

Щоб визначити кількісний розподіл частоти в конкретний момент часу і проаналізувати його динаміку обчислюють ентропію спектру потужностей на графіку. Ентропія, як міра випадковості інформаційного сигналу, є корисною через те що випадкові сигнали є непередбачуваними. В випадкових сигналах кожна точка дає нову інформацію про сигнал. З іншою сторони, чим більш передбачуваний сигнал тим більше він впорядкований і тим менше нової корисної інформації несуть нові дані [21].

1.4.4. Вейвлети

Вейвлет — це хвилеподібне коливання з амплітудою, яка починається з нуля, збільшується або зменшується, а потім повертається до нуля один або кілька разів. Вейвлети називаються «короткими коливаннями». Встановлено таксономію вейвлетів, засновану на кількості та напрямку їх імпульсів. Вейвлети наповнені специфічними властивостями, які роблять їх корисними для обробки сигналів.

Як математичний інструмент, вейвлети можна використовувати для вилучення інформації з багатьох різних типів даних, включаючи аудіосигнали та зображення. Для повного аналізу даних необхідні набори вейвлетів. «Комплементарні» вейвлети розкладають сигнал без пропусків або накладень, так що процес розкладання математично оборотний. Таким чином, набори додаткових вейвлетів корисні в алгоритмах стиснення/декомпресії на основі вейвлетів, де бажано відновити вихідну інформацію з мінімальними втратами. Формально це подання є представленням вейвлет-серії функції, що інтегрується в квадрат, або відносно повного ортонормованого набору базисних функцій, або надповного набору або кадру векторного простору для Гільбертового простору квадратних інтегрованих функцій. Це досягається за допомогою когерентних станів.

З математичної точки зору вейвлет це функція за допомогою якої можливий частотний аналіз (рис. 1.4). Зазвичай для аналізу використовуються вейвлет-коефіцієнти такі як масштаб (Scale), час (Time), рівень (Amplitude). Обчислюються коефіцієнти за допомогою інтегрального перетворення. Відмінністю від спектрограм отриманих перетворенням Фур'є є те що за допомогою вейвлетів можна визначити чіткий зв'язок спектра від часу. Вейвлет-перетворенням є перетворення, що допомагає розкласти функцію у термінах коливань за часом і частотою. Якщо отриманий вейвлет є локальним у просторі, це означає що його енергія знаходиться у скінченому інтервалі [22].

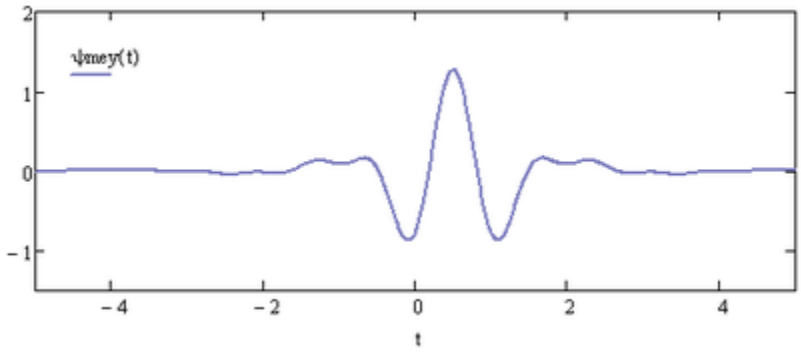


Рис. 1.4. Вейвлет Мейера

Метод вейвлентів кращий за STFT. Через те що неможливо отримати кращу роздільну здатність одночасно за часом і частотою, вікно константного розміру що використовується під час обчислень STFT не може дати точних результатів. Як пропозиція, почали використовуватись вікна різного розміру в залежності від обраної частоти за допомогою обчислення косинусної функції і її стиснення або розтягення в залежності від частоти. У результаті розмір вікна змінюється в залежності від масштабу.

Це і є основною ідеєю вейвлеті: використовувати різні розміри вікон для різних частот. Це виконується за допомогою перетворення (розтягення та стиснення) функції, яку називать материнським вейвлетом. Функція для материнського вейвлету може бути довільною. Існує деяка база основних материнських вейвлетів, які мають свої переваги у різних за стосунках.

1.4.5. Методи штучного інтелекту

Активно вивчається можливість застосування машинного навчання для аналізу даних зібраних за допомогою ЕЕГ. Машинне навчання довело свою ефективність у багатьох галузях і були розроблені основні техніки та найбільш ефективні методи аналізу для різних видів даних [23].

Глибинне навчання або Deep Learning [24], набирають популярності та знаходить свій застосунок у різних сферах через збільшення обчислювальної здатності новітніх комп’ютерних систем. Однією зі сфер є аналіз мозкової активності. Деякі дослідження мають на меті аналіз цих даних для виявлення хвороб. Деякі для застосування даних для забезпечення додаткових можливостей людям (як контролер). Такі техніки напівавтоматичного комп'ютерного аналізу даних мають потенціал через комплексність свого підходу, різноманітність задач що можуть виконувати і як результат можуть замінити людський аналіз ЕЕГ даних [25].

1.5. Використання у різних сферах

Нижче розглянуті роботи у яких досліджена чи використана активність мозку на основі мозкових хвиль у різних сферах життя та наведені деякі з цих досліджень. у яких використовуються як і лабораторні ЕЕГ з точним і детальним збором даних, так і більш прості пристрої ЕЕГ для використання у звичайному житті.

У одному з досліджень вивчається електрична активність мозку за допомогою ЕЕГ для розуміння процесів, які лежать в основі вправного виконання [26]. За думкою дослідників, сумнозвісною проблемою ЕЕГ є те, що справжні церебральні дані часто забруднені артефактами нецеребрального походження. Такі артефакти, як правило, посилюються, коли суб'єкт рухається, а це означає, що отримання достовірних даних під час вправ за своєю суттю проблематично. У цій статті обговорюється, як емпіричні дослідження загалом вирішують проблему артефакту руху шляхом прийняття альтернативних парадигм, які уникають запису під час фактичного фізичного навантаження. Крім того, обговорюються конкретні проблеми, які створює рух для отримання надійних даних ЕЕГ, а також практичні та обчислювальні методи протистояння цим проблемам. Нарешті, оскільки запис ЕЕГ у спорті часто підкріплюється бажанням оптимізувати продуктивність, також представлено короткий огляд ЕЕГ-біологічного зворотного зв'язку та досліджень максимальної продуктивності. Знання практичних аспектів запису ЕЕГ разом з появою нових технологій і все більш складними моделями обробки пропонують багатообіцяючий підхід до мінімізації, якщо, можливо, не повністю обійти проблему отримання надійних даних ЕЕГ під час руху. Отже дослідження у цьому напрямку зможуть допомогти створити ЕЕГ пристрої які можна буде використовувати при активних діях, наприклад під час активних вправ.

Наступне дослідження спрямоване на розпізнавання емоцій за допомогою ЕЕГ [27]. За думкою дослідників розпізнавання емоцій може здійснюватися за допомогою тексту, мови, виразу обличчя або жесту. У цій статті вони

зосередилися на розпізнаванні «внутрішніх» емоцій за сигналами ЕЕГ. Вони пропонують алгоритм кількісної оцінки основних емоцій на основі фрактальної розмірності в реальному часі з використанням моделі емоцій збудження-валентності. Було запропоновано та реалізовано два експерименти індукції емоцій із музичними стимулами та звуковими стимулами з бази даних International Affective Digitalized Sounds. Алгоритм реального часу був запропонований, реалізований та протестований для розпізнавання шести емоцій, таких як страх, розчарування, сум, щасливий, приємний та задоволений. Додатки реального часу були запропоновані та реалізовані у віртуальних 3D середовищах. Емоції користувача розпізнаються та візуалізуються в режимі реального часу на його/її аватарі, додаючи ще один так званий «емоційний вимір» до інтерфейсу комп'ютера людини. Запропоновано та впроваджено сайт музичної терапії з підтримкою ЕЕГ. Музика, яка звучить для пацієнтів, допомагає їм впоратися з такими проблемами, як біль і депресія. Було розроблено та впроваджено веб-програвач на основі ЕЕГ, який може відображати музику відповідно до поточних емоцій користувача.

Одна зі статей використовує дані ЕЕГ як новий метод аутентифікації особи [28]. Додаток розроблений авторами статті дозволяє блокувати та розблоковувати комп'ютер за допомогою інтерфейсу. Мозкові хвилі людини використовуються в режимі реального часу як пароль для розблокування та блокування екрана. У роботі дані збираються з гарнітури Emotiv за 14 каналами.

У наступній статті, автори описують свою систему інтерфейсів, за допомогою якої можна керувати зовнішніми пристроями тільки за допомогою альфа-хвиль які генеруються в результаті руху очей [29]. За думкою дослідників, така система може допомагати керувати протезами, роботизованими озброєннями та зовнішніми пристроями за допомогою альфа-мозкових хвиль та ритму Mu. Реакції мозку на рух очей у формі мозкових

хвиль збиралися та оброблялися для управління робототехнічними системами та управлінням розумним будинком.

ВИСНОВКИ ДО РОЗДІЛУ 1

ЕЕГ стає все більш поширеним тестом, який можна проводити не тільки у медичних закладах. За допомогою нього можна визначати активність мозку людини на основі показань активності мозкових хвиль. Ці дані можуть бути корисними як у медичній сфері та діагностиці, так і у прикладних застосунках. Основним компонентом кожного дослідження є ЕЕГ апарат, що використаний, бо від нього залежить точність, кількість каналів, форм фактор, а отже і результати дослідження. Наразі немає поширених застосувань окрім діагностики та реабілітації, але нові дослідження пропонують більш масове використання пристроїв на основі технології ЕЕГ.

Якщо розглянути природу зібраних даних, то можна відзначити наступні особливості:

- вони є мультимодальними. ЕЕГ дані є сукупністю замірів з електродів, що розташовані у різних регіонах. Додатково збираються дані про поточний час вимірювання та додаються вхідні дані від дослідника про опис дій користувача. Також можуть використовуватись інші прилади для більш точного визначення кореляцій активності мозку та іншими видами активності.
- дані збираються у великій кількості. Зазвичай частота проведення замірів є досить великою. У розглянутих пристроях частота дискретизації могла досягати 100кГц. Кількість каналів може досягати 64. Таким чином кожен експеримент це дані великого обсягу. Для їх ефективного аналізу варто використовувати новітні технології, методи та техніки, що можуть використати це як перевагу.
- Діяльність мозку має нелінійний характер, може сильно змінюватись у часі та в залежності від зовнішніх факторів, і також має просторові зв’язки між різними групами нейронів. Отже мають бути кореляції між даними зібраними з різних частин мозку. Ці кореляції у вигляді певних патернів можуть залежати від активності користувача. Комплексність даних та складність аналізу робить аналітичні спостереження на базі

досвіду неефективними. До того ж природа деяких процесів у мозку залишається невивченою. Через це чисельні методи аналі даних на основі даних великого обсягу, таких як штучний інтелект та машинне навчанн, можуть бути більш ефективними при дослідженні данних мозкових хвиль.

Тому потрібно вивчати техніки, що можуть бути найбільш ефективними для аналізу мультимодальних даних ЕЕГ. Такі методи аналізу можуть використовуватись у за стосунках та при діагностиці й дослідженнях.

РОЗДІЛ 2
АНАЛІЗ ЗАСОБІВ ОБРОБКИ МУЛЬТИМОДАЛЬНИХ ДАНИХ НА ОСНОВІ МЕТОДІВ ШТУЧНОГО ІНТЕЛЕКТУ

2.1. Штучний інтелект та машинне навчання

Штучний інтелект (AI) — це інтелект, який можуть використовувати машини для аналізу. Дослідження AI було визначено як область вивчення інтелектуальних агентів, яка відноситься до будь-якої системи, яка сприймає навколишнє середовище і виконує дії, які максимізують її шанси на досягнення поставлених цілей [30].

Раніше цей термін використовувався для машин, що здатні імітувати і демонструвати когнітивні навички, які можуть порівнюватись з людськими, і здатні «навчатись» та «вирішувати проблеми». Після цього це визначення було відкинуто основними дослідниками AI, які вирішили описувати та розглядати AI з боку раціональності та раціональних дій, що не буде обмежувати формулюванням та визначенням інтелекту.

AI використовується у різних додатках з різною метою. Наприклад, широко використовуються включають просунуті веб-пошукові системи (Google), системи рекомендацій в інформаційному просторі (які використовуються YouTube, Amazon і Netflix з різною метою), системи аналізу людської мови (наприклад, Siri та Alexa), самокеровані автомобілі (наприклад, Tesla), автоматизоване прийняття рішень і змагання на найвищому рівні в стратегічних ігрових системах (таких як шахи та го). Оскільки машини стають все більш спроможними, завдання, які вважаються такими, що потребують «інтелекту», часто вилучаються з визначення AI. Наприклад, оптичне розпізнавання символів часто виключається з речей, які вважаються штучним інтелектом, ставши рутинною технологією.

Машинне навчання — це напрямок комп'ютерних наук, який вивчає комп’ютерні алгоритми, які за допомогою аналізу даних набувають певного досвіду, який може бути використаний для аналізу нових даних [31]. Цей

напрямок є частиною напрямку з вивчення штучного інтелекту. Алгоритми машинного навчання використовуються для побудови моделей з використанням вибіркових даних, які називають навчальними дані. Далі створені та навчені моделі можуть бути використані у застосунках, щоб робити прогнози або приймати рішення без необхідності програмування явного алгоритму прийняття рішень. Алгоритми машинного навчання використовуються в додатках у різних сферах. Найбільш відомими напрямками для досліджень є автопілот для автівок (наприклад, розробки компаній Tesla та Volvo), редагування відео та фотографій (так звані deepfake), алгоритми рекомендацій музики та відео (від Google та Spotify), розпізнавання голосу та аудіо (Siri від Apple та Shazam) тощо. Також є досить вагомі інноваційні застосування у сферах енергетики, медицини, кіберзахисту та навіть програмування.

Напрямок машинного навчання пов’язаний з обчислювальною статистикою, метою якої є статистичний аналіз та передбачення на основі статистичного аналізу. Потрібно відзначити, що машинне навчання не є підмножиною статистичного навчання. Вивчення математичної оптимізації надає методи, теорію та прикладні області у сфері машинного навчання. Як наукова діяльність, машинне навчання було започатковане як результат дослідження штучного інтелекту.

Основними принципами машинного навчання є навчання і прогнозування на основі пасивних спостережень. На відміну від машинного навчання, системи штучного інтелекту мають передбачати взаємодію агента з навколишнім середовищем для навчання та дій, які максимізують його шанси на успішне досягнення цілей [32].

Метою машинного навчання є аналіз даних. Також розрізняють інтелектуальний аналіз даних. Це міждисциплінарна підгалузь інформатики та статистики із загальною метою вилучення інформації з метою перетворення в зрозумілу форму та структуру для подальшого використання. Об'єктом вивчення є процес вилучення та виявлення закономірностей у великих наборах

даних із застосуванням методів та практик, що використовуються та вивчаються у таких сферах як машинне навчання, статистика та системи баз даних.

Задачею цієї підгалузі є вивчення способів напівавтоматичного або автоматичного аналізу великої кількості даних для знаходження раніше невідомих залежностей та інформації. Розрізняють наступні види аналізу:

- кластерний аналіз - аналіз груп записів,
- виявлення аномалій - аналіз записів, що не підпадають під певний шаблон чи правила
- послідовний аналіз шаблонів вибір асоціативних правил - знаходження залежностей чи шаблонів у даних

Різниця між інтелектуальним аналізом даних і аналізом даних в звичному розумінні полягає в тому, що звичайний аналіз даних використовується для перевірки гіпотез щодо набору даних. Наприклад, часто аналізується продуктивність певних кампаній на основі зібраних даних в незалежності від кількості цих даних. Але ці методи аналізу даних є неефективними або неможливими на даних великого обсягу. Машинне навчання та статистичні моделі використовуються під час інтелектуального аналізу для виявлення таємних або прихованих закономірностей у великому обсязі даних. Тобто такому обсягу даних, де методи звичайного аналізу є неефективними і недоцільними.

Одній ті ж самі методи аналізу використовуються у машинному навчанні та інтелектуальному аналізі даних. Відмінністю є те що метою машинного навчання є передбачення результату зробленого на відомих властивостях, отриманих із навчальних даних, а метою інтелектуального аналізу даних є виявлення невідомих властивостей цих даних. Інтелектуальний аналіз даних може застосовувати методи та техніки машинного навчання, але для іншого реузультату. Машинне навчання також використовує методи інтелектуального аналізу даних (наприклад, як «навчання без учителя» або як етап попередньої обробки для підвищення точності).

Розрізняють наступні основні види машинного навчання:

- навчання під наглядом (supervised learning) або навчання з учителем,
- навчання без нагляду (unsupervised learning) або навчання без учителя,
- напівконтрольоване навчання (semi-supervised learning) або напівавтоматичне навчання,
- навчання з підкріпленням (reinforecement learning).

2.1.1. Навчання з учителем

При створенні алгоритмів навчання з учителем створюють математичну модель набору даних, яка містить як вхідні дані, так і бажаний результат у вигляді мітки [33]. Дані діляться на декілька категорій, одна з яких навчальні дані, що складаються з навчального прикладу та очікуваного результату. Кожен навчальний приклад має певні вхідні дані і бажаний вихід, також відомий як контрольний сигнал. У математичній моделі кожен навчальний приклад представлений деякими даними у формі масива або вектора. Навчальні дані представлені матрицею певної розмірності. Основним процесом під час навчання є оптимізація цільової функції під час якого алгоритми навчання визначають функцію, яка може бути використана для більш коректного прогнозування результату на базі отриманих даних. Оптимальна функція дозволяє алгоритму правильно визначити результат для вхідних даних. Для визначення ефективності та перевірки функції використовуються інші набори даних, які не використовувались при навчанні. Так звані валідаційні та тестові набори. Вважається, що алгоритм, який покращує точність своїх результатів або прогнозів з часом, навчився виконувати це завдання. Також можуть бути використані інші метрики для порівняння моделей (час навчання, ресурси необхідні для використання, час на визначення результату тощо).

Алгоритми навчання з учителем включають активне навчання, класифікацію та регресію. Алгоритми класифікації використовуються для вирішення задач класифікації, коли кількість класів для вхідних та вихідних даних заздалегідь відома. Алгоритми регресії використовуються, коли вихідні дані можуть мати

будь-яке числове значення в межах діапазону, тобто мають нескінченний діапазон значень.

2.1.2. Навчання без учителя

Алгоритми навчання без учителя використовують набір даних, який містить лише вхідні дані. Цей вид алгоритмів має на меті знайти структуру в даних та зв'язки в даних. Наприклад, створюють кластери чи групи подібних даних. Таким чином, алгоритми навчаються на даних, які не мають ніяких позначень, міток на відміну від навчальних даних для алгоритмів навчання з учителем. Замість того, щоб реагувати на зворотний зв’язок від моделі (наприклад, точність визначення результату під час навчання з урахуванням бажаних вихідних даних), алгоритми навчання без учителя визначають спільні риси в даних і реагують на наявність або відсутність таких рис у кожній новій частині даних. Центральне застосування неконтрольованого навчання є в області оцінки щільності в статистиці, наприклад, знаходження функції щільності ймовірності [34]. Алгоритми навчання без учителя охоплює інші області, що включають узагальнення та пояснення особливостей даних.

2.1.3. Напівавтоматичне навчання

Напівавтоматичне навчання має спільні риси і з навчанням без учителя (у якому навчальні дані не мають жодних бажаних вихідних даних), і з навчанням з учителем (у якому навчальні дані мають бажані вихідні дані). У цьому виді, у навчальних даних деякі навчальні приклади не мають міток. Дослідники виявили, що немарковані дані, якщо вони використовуються разом з невеликою кількістю маркованих даних, можуть значно покращити точність навчання. При навчанні зі слабким контролем навчальні мітки можуть бути виставлені з додатковим шумом, наявними у невеликій кількості або недостатньо точно виставлені. Однак ці мітки часто дешевше та швидше можна зробити саме для цього виду алгоритмів ніж для навчання з учителем, що призводить до більш ефективних навчальних наборів [35].

2.1.4. Навчання з підкріпленням

Навчання з підкріпленням — це область машинного навчання, метою якої є створення програмних агентів, які здатні виконувати певні дії для досягення максимального результату у певному середовищі. Під час навчання агента навчають отримувати кращий результат даючи нагороду за покращення. Через свою загальність ця галузь активно вивчається в багатьох дисциплінах теорія ігор, теорія управління, дослідження операцій, теорія інформації, оптимізація на основі моделювання, багатоагентні системи, статистика та генетичні алгоритми. Середовище зазвичай представлене у вигляді процесу прийняття рішень Маркова (MDP). Також часто використовуються методи динамічного програмування для більш ефективного запам’ятовуваввння результату [36]. Точна модель MDP не потрібна для використання алгоритмів навчання з підкріпленням. Цей тип аналізу використовується коли створення точної моделі MDP не є можливим. Такий аналіз зараз широко застосовується автономних транспортних засобах або в навчанні штучного інтелекту, який може грати проти людини.

2.1.5. Глибинне навчання

Глибинне навчання (також відоме як глибинне структуроване навчання) є частиною більш широкого сімейства методів машинного навчання, заснованих на штучних нейронних мережах з репрезентативним навчанням. Навчання може бути з учителем, з частковим наглядом або без учителя [37].

Розрізняють наступні основні види моделей

- дерева рішень;
- регресійний аналіз;
- генетичний алгоритм;
- штучні нейронні мережі.

Навчання дерева рішень використовує метод дерева рішень як модель прогнозування для переходу від спостережень за елементом, що знаходиться у гілках дерев, до висновків про цільове значення елемента, представленого у

листках. Це є одним з варіантів прогнозованого моделювання, який може використовуватись у таких сферах як статистика, аналіз даних, машинне навчання. Існує декілька віидів моделей дерев. Якщо цільова змінна приймає дискретний набір значень, то такі дерева називають деревами класифікації. В такому виді листки представляють мітки класів, а гілки представляють ознаки за якими можна визначити клас. Дерева в яких цільова змінна приймає безперервні значення з нескінченного набору називають деревами регресії. Дерева прийняття рішень можуть бути корисними для наочного і явного представлення процесу прийняття рішень та самих рішень. У сфері аналізу даних дерево рішень описує дані, але результуюче дерево класифікації може бути вхідним для прийняття рішень [38].

Регресійний аналіз, метоб якого є визначення зв’язку між вхідними даним та пов’язаними з ними ознаками, охоплює велику різноманітність статистичних методів. Найбільш простою та поширеною є лінійна регресія. Під час лінійної регресії визначається формула, що визначає певну криву, яка найкраще відповідає заданим даним відповідно до математичного критерію, наприклад, найменшої відстані [39].

Генетичний алгоритм (GA) — це алгоритм пошуку, яка імітує процес природного відбору, використовуючи такі методи, як мутація та кросовер, для створення нових генотипів у надії знайти найоптимальніші рішення для даної проблеми [40].

Штучна нейронна мережа — це штучна мережа, що складається зі штучних вузлів, які підраховують та зберігають свій внутрішній стан, який називають вагами. Ця парадигма обробки інформації натхнена способами обробки даних біологічними нейронними системами. Вузли мають зв'язки схожі на ті, що наявні у мозку між нейронами [41]. Штучний нейрон, отримуючи сигнал, оброблює його і може передавати цей сигнал іншим штучним нейронам з якими у нього наявні зв’язки. У найбільш популярних реалізаціях сигнал, який передається між нейронами, представлений дійсним числом, а для обчислення виходу кожного з нейронів використовується певна нелінійна функція суми

його вхідних даних. Зв’язки та можливість передавати дані від нейрона до нейрона називається «ребром». Внутрішній стан ребра та кожного нейрона визначаються за допомогою ваг. Ваги коригуються під час навчання для отримання найбільш правильного результату. Вага нейрона може посилювати вплив нейрона або зменшувати його. Часто використовуються такі функції, що сигнал надсилається нейроном тільки у випадку якщо були досягнуті певні порогові значення. Нейрони об’єднуються у шари. Різні види шарів здатні виконувати різні види перетворень у своїх входах. Сигнал посліовно поширюється від шару до шару, від першого (вхідного) до останнього (вихідного). Також можливе багаторазове проходження цих шарів для отримання додаткової інформації.

У сфері штучного інтелекту штучні нейронні мережі успішно застосовуються для розпізнавання мовлення, аналізу зображень та адаптивного керування, для створення програмних агентів (у комп’ютерних та відеоіграх) або автономних роботів.

Глибинне навчання є новим сімейством машинного навчання з використанням штучних нейронних мереж та різних видів навчання.

Різні архітектури глибинного навчання, такі як глибинні нейронні мережі, глибинне навчання з підкріпленням, рекурентні нейронні мережі та згорткові нейронні мережі, застосовується в таких областях, як комп’ютерний зір, розпізнавання мовлення, обробка природної мови, машинний переклад тощо. Тобто сімейство моделей глибинного навчання набуло популярностей у всіх сферах. Вони мають перевагу при аналізі даних великого обсягу. Глибинне навчання — це сучасна варіація, яка стосується необмеженої кількості шарів обмеженого розміру, що дозволяє практичне застосування та оптимізовану реалізацію, зберігаючи теоретичну універсальність. У глибинному навчанні шари також можуть бути неоднорідними та значно відхилятися від біологічно обґрунтованих моделей заради ефективності, можливості навчання та зрозумілості, звідки і виникає «структурована» частина.

2.2. Згорткова штучна нейронна мережа

У глибинному навчанні згорткова нейронна мережа (CNN, або ConvNet) — це клас штучних нейронних мереж, який найчастіше використовується для аналізу візуальних зображень через свою здатність знаходити зв'язки між даними у просторі [42]. Вони мають застосування в розпізнаванні зображень і відео, системах рекомендацій, класифікації зображень, сегментації зображень, аналізі медичних зображень, обробці природною мовою, інтерфейсах мозок-комп’ютер та фінансових тимчасових рядах.

CNN є регуляризованими версіями багатошарових персептронів. Багатошарові персептрони зазвичай означають повністю пов’язані мережі, тобто кожен нейрон одного шару з’єднаний з усіма нейронами наступного шару. «Повна зв’язність» цих мереж робить їх схильними до перенасичення даними. Типові способи регуляризації або запобігання перенасиченню є наступними:

- штрафні параметри під час тренування (наприклад, зниження ваги)
- обрізання зв'язку (пропущені з'єднання, відключення тощо) (dropout).

CNN використовують інший підхід до регуляризації. Використовуються переваги ієрархічного шаблону в даних і збирають шаблони зростаючої складності, використовуючи менші та простіші візерунки, відображені в їх фільтрах.

Згорткові мережі були натхненні біологічними процесами [43], оскільки структура зв’язку між нейронами нагадує організацію зорової кори тварин. Окремі нейрони кори реагують на подразники лише в обмеженій області поля зору, відомому як рецептивне поле. Рецептивні поля різних нейронів частково перекриваються таким чином, що охоплюють все поле зору.

CNN використовують відносно мало попередньої обробки порівняно з іншими алгоритмами класифікації. Це означає, що мережа вчиться оптимізувати фільтри (або ядра) за допомогою автоматизованого навчання, тоді як у традиційних алгоритмах ці фільтри створюються вручну. Ця незалежність

від попередніх знань і людського втручання у виділення ознак є великою перевагою.

Мережа такого типу складається з вхідного шару, прихованих шарів і вихідного шару. У будь-якій нейронній мережі з прямим зв’язком будь-які середні шари називаються прихованими, оскільки їхні вхідні та вихідні дані маскуються функцією активації та кінцевою згорткою.

2.2.1. Згорткові шари

У згортковій нейронній мережі прихованими шарами є шари, які виконують згортки. Зазвичай це включає шар, який виконує точковий добуток ядра згортки з вхідною матрицею шару. Коли ядро згортки переміщається по вхідній матриці для шару, операція згортки створює карту ознак, яка, у свою чергу, є вхідними даними для наступного шару. Далі йдуть інші шари, такі як шари об’єднання, шари повного зв’язку та шари нормалізації.

Згортковий шар є основним для мережі CNN. Параметри шару складаються з набору доступних для навчання фільтрів (або ядер), які мають невелике рецептивне поле (поле сприйняття), але поширюються на всю глибину вхідного об’єму. Під час прямого проходу кожен фільтр згортається по ширині та висоті вхідного об’єму, обчислюючи крапковий добуток між записами фільтра та вхідними даними, створюючи двовимірну карту активації цього фільтра. В результаті мережа вивчає фільтри, які активуються, коли вона виявляє певний тип об’єкта в деякому просторовому положенні на вході [44].

2.2.1.1. Рецептивне поле

У нейронних мережах кожен нейрон отримує вхідні дані з деякої кількості місць у попередньому шарі. У згортковому шарі кожен нейрон отримує вхідні дані лише з обмеженої області попереднього шару, яка називається рецептивним полем нейрона (Рис. 2.1). Зазвичай площа являє собою певну матрицю. Тоді як у повністю зв’язаному шарі рецептивним полем є весь попередній шар. Таким чином, у кожному згортковому шарі кожен нейрон

отримує вхідні дані з більшої області входу, ніж попередні шари. Це пов’язано з повторним застосуванням згортки, яка враховує значення певних даних, а також його даних, що знаходяться поруч.

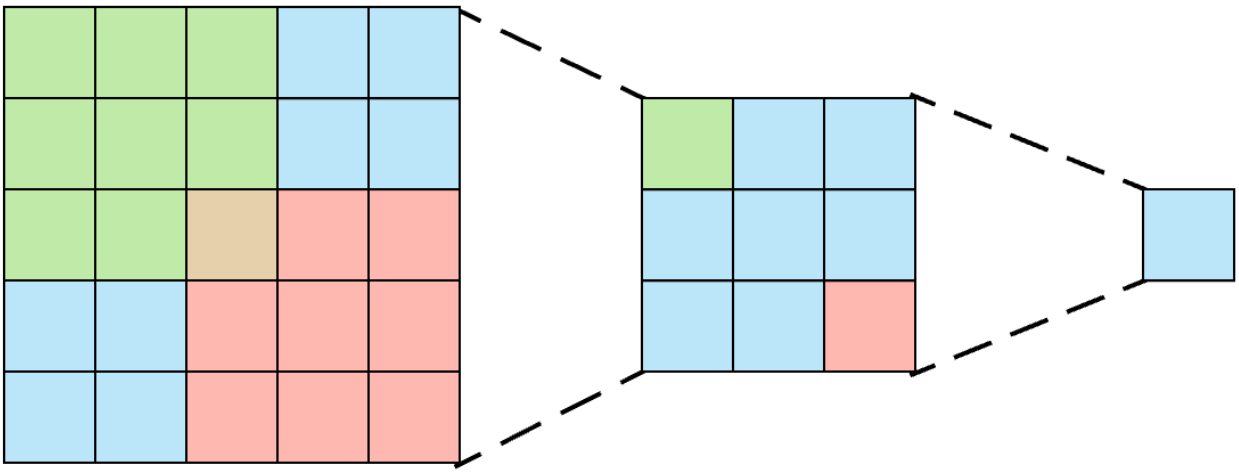


Рис. 2.1. Приклад використання рецептивного поля

2.2.1.2. Ваги

Кожен нейрон нейронної мережі обчислює вихідне значення, застосовуючи конкретну функцію до вхідних значень, отриманих із сприйнятливого поля в попередньому шарі. Функція, яка застосовується до вхідних значень, визначається вектором ваг і зміщенням (зазвичай дійсними числами). Навчання складається з ітеративного коригування ваг.

Вектор ваг і зміщення називаються фільтрами і представляють особливі характеристики вхідних даних (наприклад, конкретну форму). Відмінною особливістю CNN є те, що багато нейронів можуть використовувати один і той же фільтр. Це зменшує обсяг пам’яті, оскільки одне зміщення та один вектор ваг використовуються для всіх сприйнятливих полів, які спільно використовують цей фільтр, на відміну від того, що кожне рецептивне поле має власне зміщення та векторне зважування.

2.2.1.3. Особливості використання згорткових шарів

У CNN вхідним є тензор із розмірністю: (кількість вхідних даних) x (висота вхідних даних) x (ширина вхідних даних) x (кількість каналів вхідних даних).

Після проходження через згортковий шар зображення абстрагується до карти об’єктів, або картою активації, з формою: (кількість вхідних даних) х (висота карти об’єктів) х (ширина карти об’єктів) х (канали карти об’єктів).

Згорткові шари згортають вхідні дані і передають його результат наступному шару. Це схоже на реакцію нейрона в зоровій корі на певний стимул [45]. Кожен згортковий нейрон обробляє дані тільки для свого рецептивного поля. Хоча повністю підключені нейронні мережі з прямим зв’язком можна використовувати для вивчення функцій і класифікації даних, ця архітектура, як правило, непрактична для більших вхідних даних, таких як зображення з високою роздільною здатністю. Такі моделі потребують значної кількості обчислювальних ресурсів через дуже велику кількість використаних нейронів. Навіть у не глибинній архітектурі, через великий розмір вхідних зображень, де кожен піксель є відповідним елементом входу потребує багато ресурсів. Наприклад, повністю підключений шар для зображення розміром 100 х 100 має 10 000 ваг для кожного нейрона другого шару. Натомість згортка зменшує кількість вільних параметрів, що дозволяє розробляти мережі з більшою кількості шарів, тобто більш глибинні мережі [46]. Наприклад, незалежно від розміру зображення, для використання області мозаїки 5 х 5, кожна з яких має однакові спільні ваги, потрібно лише 25 параметрів, які можна вивчати. Використання регуляризованих ваг для меншої кількості параметрів дозволяє уникнути проблем зникнення градієнтів і вибухових градієнтів, які спостерігаються під час зворотного поширення в традиційних нейронних мережах.

Зворотне розповсюдження — це алгоритм контролю за навчанням, який використовується у штучних нейронних мережах. Основою цього алгоритму є техніка градієнтного спуску. На основі функції помилки та результатів штучної нейронної мережі, обчислюється градієнт функції помилки від ваг у мережі. Градієнт обчислюються проходячи у зворотньому напряму через мережу. Таким чином градієнт останніх ваг отримується першим, а градієнт перших ваг останнім. Часткові результати обчислення градієнту з одного шару повторно

використовуються під час обчислення градієнтів інших шарів (попередніх). Ця техніка дозволяє ефективно підраховувати градієнт для кожного шару [47].

Оскільки алгоритм зворотного поширення просувається вниз від вихідного шару до вхідного, градієнти часто стають все меншими і наближаються до нуля, що в кінцевому підсумку залишає вагу початкового або нижнього шарів майже незмінним. В результаті градієнтний спуск ніколи не сходиться до оптимального. Це відомо як проблема зникаючих градієнтів.

Навпаки, в деяких випадках градієнти продовжують збільшуватися і збільшуватися в міру виконання алгоритму зворотного поширення. Це, у свою чергу, спричиняє дуже велике оновлення ваги та призводить до розбіжності градієнтного спуску. Цю проблему називають вибухаючими градієнтами.

2.2.1.4. Характеристики згорткових шарів

Під час роботи з високорозмірними вхідними даними (наприклад, зображення з високою роздільною здатністю) підключення усіх нейронів до всіх між кожними шарами не є доцільним. Така архітектура не здатна ефективно використовувати знання про природу даних. Згорткові мережі здатні використовувати просторову локальну кореляцію. Таким чином згорткові шарі можуть забезпечити більш розріджений шаблон зв’язку між нейронами сусідніх шарів позбавившись від великої кількості неважливих зв’язків, оскільки кожен нейрон наступного шару підключений лише до невеликої групи нейронів попереднього.

Три гіперпараметри контролюють розмір вихідного об’єму згорткового шару:

- глибина;
- крок;
- розмір заповнення.

2.2.1.4.1. Глибина

Глибина вихідного об’єму контролює кількість нейронів у шарі, які виконують обчислення у тій самій області вхідних даних. Ці нейрони вчаться активуватися для різних функцій у вхідних даних. Наприклад, якщо перший згортковий шар бере вихідне зображення, то різні нейрони вздовж виміру глибини можуть активуватися за наявності різних об’єктів, які вони можуть знайти. Наприклад, деякі нейрони можуть активуватись знайшовши схожу форму, інші знайшовши схожий колір.

2.2.1.4.2. Крок

Крок керує розподілом стовпців глибини по ширині та висоті. Якщо крок дорівнює 1, фільтри переміщаються на один піксель за раз. Це призводить до сильного перекриття сприйнятливих полів між стовпцями та до великих обсягів виходу. Для будь-якого цілого числа крок S означає, що фільтр переміщується на S одиниць за раз. Більший крок означає менше перекриття рецептивних полів і менші просторові розміри вихідних даних [48].

2.2.1.4.3. Заповнення

Іноді зручно заповнювати вхідні дані нулями (або іншими значеннями, такими як середнє значення регіону) на межах, щоб змінити розмір віхідних даних. Розмір цього заповнення є третім гіперпараметром. Заповнення забезпечує контроль просторового розміру вихідних даних. Зокрема, іноді бажано точно зберегти просторовий розмір вхідного об’єму, це зазвичай називають «однаковим» заповненням.

2.2.1.5. Підвибірка (Pooling layer)

Підвибірка — це техніка, яка була розроблена для зменшення залежності від точного позиціонування в картах об’єктів, які створюються згортковими шарами в CNN.

Внутрішні елементи CNN містять ядра/фільтри фіксованих розмірів, які називаються детекторами функцій. Як тільки ознаки на зображенні виявлені, інформацію щодо положення об’єкта на зображенні можна фактично не враховувати, що має додаткові переваги.

Згорткові мережі можуть включати локальні та/або глобальні рівні об’єднання разом із традиційними згортковими шарами. Об’єднання шарів зменшує розміри даних, об’єднуючи вихідні дані кластерів нейронів на одному шарі в один нейрон наступного шару. Локальне об’єднання об’єднує невеликі кластери різної розмірності. Це є формою нелінійної дискретизації. Існує два поширених типи об’єднання в популярному використанні: максимальний і середній. Максимальне об’єднання використовує максимальне значення кожного локального кластера нейронів на карті ознак, тоді як середнє об’єднання приймає середнє значення.

Середнє об’єднання або AvgPool – це варіант підвибірки, де середнє значення даних, які потрапляють у сприйнятливе поле одиниці в межах шару підвибірки, приймається як вихідний результат.

Максимальне об’єднання або MaxPool — це варіант підвибірки, де максимальне значення даних, які потрапляють у сприйнятливе поле одиниці в шарі підвибірки, береться як вихід.

Максимальне об’єднання є найбільш поширеним. Такий метод розбиває вхідне зображення на набір прямокутників і для кожного такого субрегіону виводить максимум (Рис. 2.2).

Інтуїтивно, точне розташування об’єкта є менш важливим, ніж його приблизне розташування відносно інших об’єктів. Це ідея використання об’єднання в згорткових нейронних мережах. Рівень об’єднання служить для поступового зменшення просторового розміру представлення, зменшення кількості параметрів, обсягу пам’яті та обсягу обчислень у мережі, а отже, для контролю обладнання необхідного для розрахунків. Це відомо як пониження вибірки. Зазвичай в архітектурі CNN періодично вставляється шар об’єднання між послідовними згортковими шарами (за кожним зазвичай слідує функція

активації, наприклад, рівень ReLU) [49]. Хоча рівні об’єднання сприяють локальній інваріантності трансляції, вони не забезпечують глобальну інваріантність перекладу в CNN, якщо тільки не використовується форма глобального об'єднання [50]. Шар об’єднання зазвичай працює незалежно на кожній глибині або зрізі вхідних даних і змінює його просторово.

Виявилося, що залежність від позиції окремих функцій є недоліком для побудови та розвитку мережі, яка може відносно добре працювати з вхідними даними, які зазнали певної форми перетворення. Ваги що навчаються на занадто залежних від позиції даних є небажаними, тому що самі об'єкти не повинні залежати від їх позиції. Можливість розпізнавання об'єкту не має залежати від його абсолютного розміщення. Інформація, яка має значення з точки зору розташування об’єктів, це відносне розташування об’єкта до інших об’єктів на карті об’єктів, а не абсолютне розташування. Щоб зменшити залежність від точного позиціонування функцій у мережах, вживається зменшення просторової роздільної здатності. Зменшення просторової роздільної здатності просто зменшує кількість пікселів у карті об’єктів, і в цьому випадку це досягається шляхом підвибірки [49].

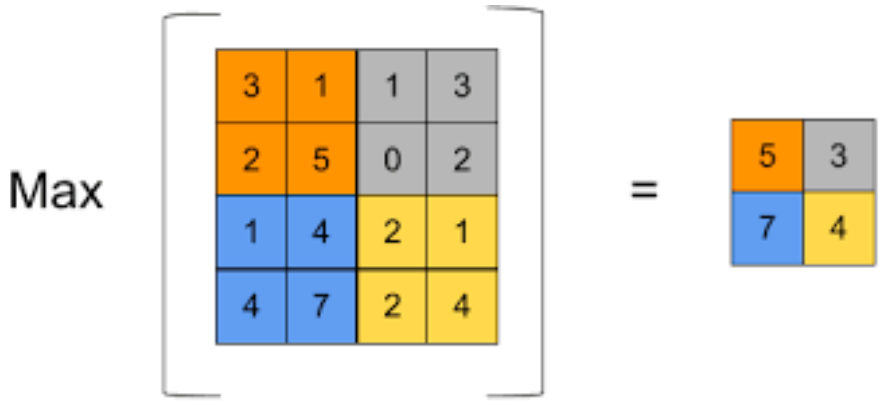


Рис. 2.2. Приклад операції максимального об’єднання, яка має вікно 2x2

2.2.1.6. Функції активації

Функція активації використовуються у нейронних мережах щоб регулювати вихід сигналу з нейрону або групи нейронів. Різні види функцій активацій мають свої переваги і недоліки в залежності від застосування. Вибір функції

може значно впливати на результати нейроннох мережі. Можливе використання різних функцій у різних частинах моделі. Технічно функція активації використовується в межах або після внутрішньої обробки кожного вузла в мережі, хоча мережі призначені для використання однієї і тієї ж функції активації для всіх вузлів у шарі.

Мережа може мати три типи шарів: вхідні шари, які отримують необроблені вхідні дані, приховані шари, які беруть вхідні дані з іншого шару і передають вихід на інший шар, і вихідні шари, які видають результат. Усі приховані шари зазвичай використовують ту саму функцію активації. Вихідний рівень зазвичай використовує функцію активації, відмінну від прихованих шарів, і залежить від типу передбачення, якого вимагає модель.

Функції активації також зазвичай диференційовані. Це необхідно з огляду на те, що нейронні мережі зазвичай навчаються за допомогою алгоритму зворотного поширення помилки, який вимагає похідної від помилки передбачення для оновлення ваг моделі.

2.2.1.6.1. Sigmoid

Sigmoid функція визначається наступним чином (2.1):

$$f(x) = \frac{1}{1 + e^{-(x)}} \quad (2.1)$$

Основна причина, чому використовується сигмовидна функція, полягає в тому, що вона існує між (0 до 1) (Рис. 2.3). Тому вона особливо використовується для моделей, де передбачається ймовірність як вихідний результат. Оскільки ймовірність чогось існує лише між діапазоном від 0 до 1, сигмовидна функція є правильним вибором. Ця функція диференційована, отже можна визначити нахил сигмовидної кривої в будь-яких двох точках.

Логістична сигмовидна функція може спричинити застрягання нейронної мережі під час навчання.

Функція softmax є більш узагальненою функцією логістичної активації, яка використовується для багатокласової класифікації.

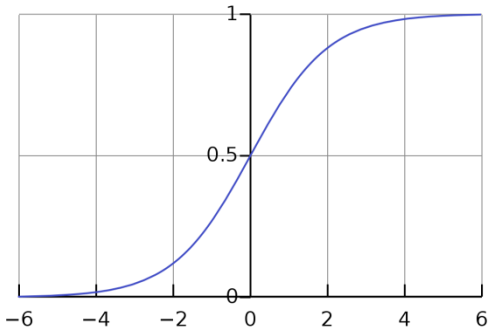


Рис. 2.3. Графік функції Sigmoid

2.2.1.6.2. Tanh

Тангенс функція визначається наступним чином (2.2):

$$f(x) = \frac{\left(e^{x} - e^{-x}\right)}{\left(e^{x} + e^{-x}\right)} \quad Tanh$$

(2.2)

Вона схожа на логістичну сигмовидну функцію. Діапазон функції tanh становить від (-1 до 1). tanh також сигмовидної форми (s - форма). Перевага полягає в тому, що негативні вхідні дані будуть відображені сильно негативними, а нульові входи будуть відображені поблизу нуля на графіку tanh (Рис. 2.4). Функція диференційована. Функція tanh в основному використовується для класифікації між двома класами.

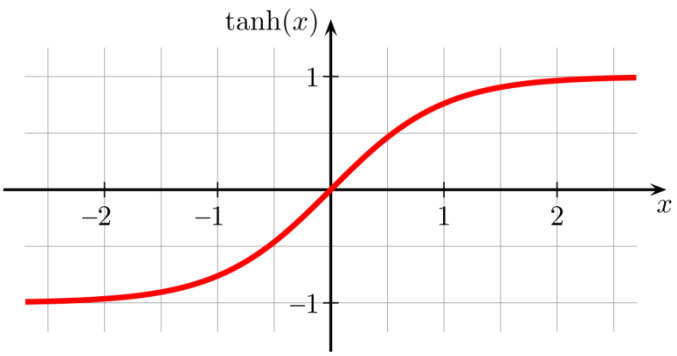


Рис. 2.4. Графік функції Tanh

2.2.1.6.3. ReLU

ReLU — це абревіатура від випрямленої лінійної одиниці, яка застосовує функцію активації без насичення (2.3).

$$\sigma(x) = \begin{cases} max(0, x) & , x >= 0 \\ 0 & , x < 0 \end{cases} \quad (2.3)$$

Він ефективно видаляє негативні значення з карти активації, встановлюючи їх на нуль [51]. Він вносить нелінійності до функції прийняття рішень і в загальну мережу, не впливаючи на сприйнятливі поля шарів згортки.

ReLU використовується майже у всіх згорткових нейронних мережах або глибинному навчанні. ReLU наполовину випрямлений (знизу). f(z) дорівнює нулю, коли z менше нуля, і f(z) дорівнює z, коли z вище або дорівнює нулю (Рис. 2.5).

ReLU часто віддають перевагу іншим функціям, оскільки він навчає нейронну мережу в кілька разів швидше без значного збитку для точності узагальнення [52].

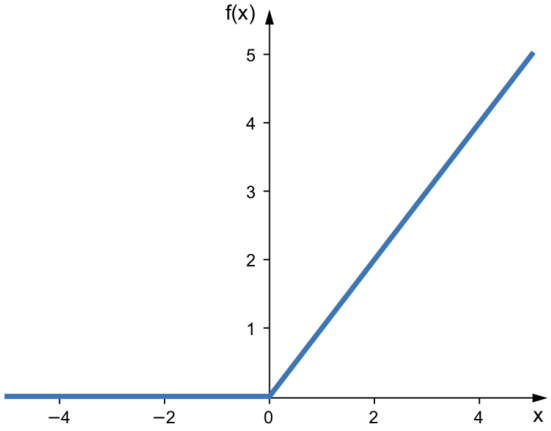


Рис. 2.5. Графік функції ReLU

2.2.1.6.4. Softmax

Softmax – це функція активації, яка масштабує числа чи вхідні дані до ймовірностей. Результатом Softmax є вектор (скажімо, v) з ймовірностями

кожного можливого результату (Рис. 2.6). Ймовірності у векторі v зводяться до одиниці для всіх можливих результатів або класів.

Математично Softmax визначається як (2.4):

$$s\left(x_i\right) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \quad (2.4)$$

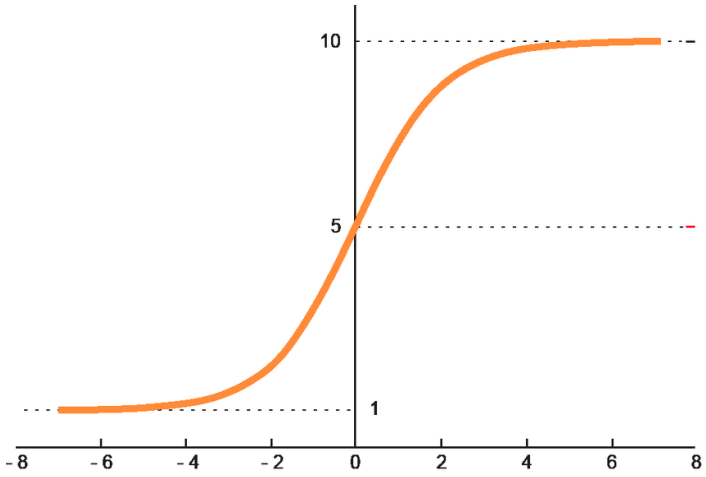


Рис. 2.6. Графік функції Softmax

2.2.1.7. Функції втрат

«Рівень втрат», або «функція втрат», визначає, як навчання штрафує відхилення між прогнозованим виходом мережі та бажаним результатом і використовується під час навчання під керівництвом. Залежно від конкретного завдання можна використовувати різні функції втрат.

2.2.1.7.1. Середня квадратична помилка

Середня квадратична помилка (MSE) – це найбільш поширена функція втрат (Рис. 2.7). Для обчислення MSE, підраховується різниця між прогнозами та істиною, береться у квадрат та вирівнюється усереднене по всьому набору даних (2.5).

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \quad (2.5)$$

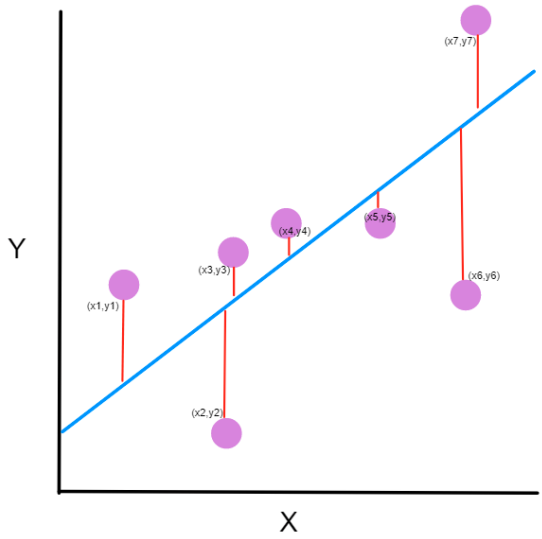


Рис. 2.7. Графік функції MSE

2.2.1.7.2. Перехресна ентропія

Функція втрат перехресної ентропії вимірює продуктивність моделі класифікації, вихідним результатом якої є значення ймовірності від 0 до 1 (Рис. 2.8). Втрата перехресної ентропії збільшується, оскільки прогнозована ймовірність відрізняється від фактичної мітки. Ідеальна модель мала б втрати 0. Оскільки прогнозована ймовірність наближається до 1, втрати повільно зменшується. Однак, оскільки прогнозована ймовірність зменшується, логарифмічні втрати швидко збільшуються (2.6). Функція втрат перехресної ентропії штрафує обидва типи помилок. Функція втрат перехресної ентропії використовується для прогнозування N незалежних значень ймовірності.

$$L(\hat{y}, y) = -\sum_{k}^{K} y^{(k)} \log \hat{y}^{(k)} \quad (2.6)$$

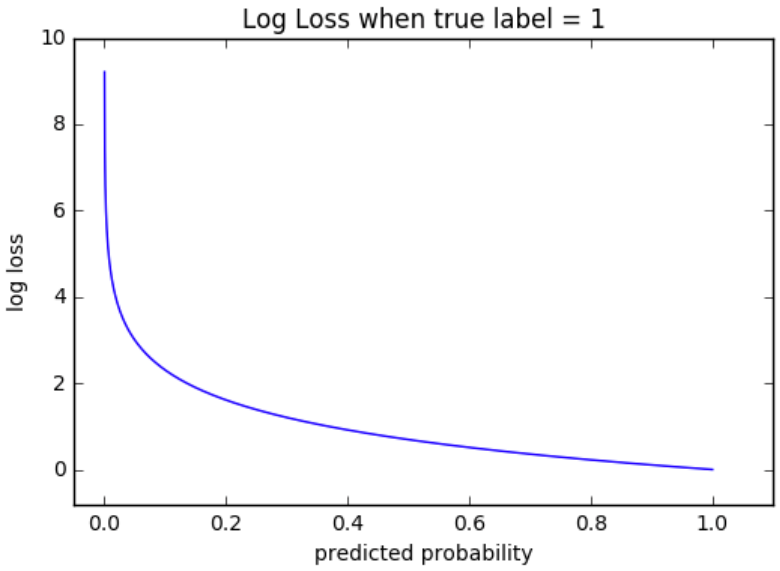


Рис. 2.8. Графік функції перехресної ентропії

2.2.1.7.3. Softmax

Функція втрат Softmax є функцією активації Softmax поєднаною з функцією втрат перехресної ентропії (Рис. 2.9). Функція активації Softmax підраховує ймовірність для кожного класу. Ці ймовірності в сумі дають 1. Функція втрат перехресної ентропії – це сума від'ємного логарифма ймовірностей. Вони обидва зазвичай використовуються разом у класифікаціях. Функція втрат Softmax використовується для передбачення одного класу K взаємовиключних класів.

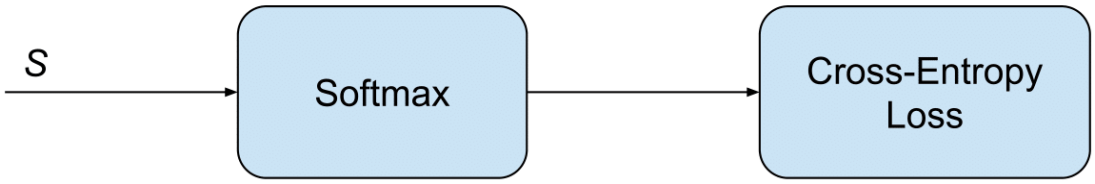


Рис. 2.9. Приклад використання функції втрат Softmax

2.2.1.8. Зменшення розмірності

Зменшення розмірності — це процес зменшення кількості випадкових величин шляхом отримання набору головних змінних. Іншими словами, під час процесу зменшення розмірності зменшується розмірність набору ознак залишаючи тільки головні з них. У більшості методів непотрібні ознаки

усуваються. Одним з методів є аналіз головних компонентів (PCA). Він передбачає перетворення даних таким чином, щоб розмірність зменшилась а усі дані збереглись. Наприклад, таким чином можна перетворити дані 3D у 2D зберігаючи всі вихідні змінні [53]. Так за гіпотезою різноманіття припускається, що масиви даних високої розмірності лежать уздовж низько вимірних. Методи зменшення розмірності роблять це припущення, що веде до області багаторазового навчання та багатообразної регуляризації.

2.2.3. Повнозв'язний шар

Після кількох згорткових і максимальних шарів об'єднання остаточна класифікація виконується за допомогою повнозв'язних шарів. Нейрони в повнозв'язному шарі мають зв'язки з усіма активаціями в попередньому шарі, як це видно в звичайних (без згорткових) штучних нейронних мережах. Таким чином, їх активації можна обчислити як афінне перетворення з множенням матриці з подальшим зміщенням (за допомогою векторного додавання вивченого або фіксованого зміщення) [54].

Повнозв'язні шари з'єднують кожен нейрон в одному шарі з кожним нейроном в іншому шарі. Це те саме, що традиційна багатошарова персептронна нейронна мережа. Після перетворення матриці у вектор, цей вектор проходить через повнозв'язні шар для класифікації.

2.2.4. Популярні архітектури згорткових мереж для задач класифікації на зображеннях

2.2.4.1. AlexNet

AlexNet - це архітектура CNN, яка складається з восьми шарів: п'яти згорткових шарів і трьох повнозв'язних шарів [55]. Особливостями мережі є (Рис. 2.10):

- Нелінійність ReLU. AlexNet використовує ReLU замість функції tanh, яка була стандартною. Перевага ReLU полягає в часі навчання. CNN, що використовує ReLU, зміг досягти 25% помилки в наборі даних CIFAR-10 в шість разів швидше, ніж CNN, що використовує tanh.

- Кілька графічних процесорів. AlexNet дозволяє тренуватися з кількома GPU, поміщаючи половину нейронів моделі на один GPU, а іншу половину на інший GPU. Це не тільки означає, що можна навчати більшу модель, але й скорочує час навчання.
- Об’єднання з перекриттям. У згортоквих нейроних мережах зазвичай не використовують перекриття нейронів. Однак, автори виявили, що перекриття допомагає зменшити похибку приблизно на 0,5%. До того ж вони виявили, що такі моделі з перекриттям зазвичай важче перенаситити ніж моделі що не використовують перекриття.

У AlexNet 60 мільйонів параметрів, що стало серйозною проблемою з точки зору перенасичення. Для зменшення перенасичення використовували два методи:

- Автори використовували трансформацію, що зберігає мітки, щоб зробити свої дані більш різноманітними. Зокрема, вони створили трансляції зображень і горизонтальні відбиття, що збільшило навчальний набір у 2048 разів. Вони також виконали аналіз основних компонентів (PCA) для значень пікселів RGB, щоб змінити інтенсивність каналів RGB, що зменшило помилку першої частини більше ніж на 1%.
- “Вимкнення” нейронів. За допомогою цієї техніки результат деяких нейронів просто не використовується. Ймовірність використання нейрона визначається заздалегідь. Таким чином при кожній ітерації використовується різна вибірка параметрів моделі. За допомогою цього кожен нейрон змушений спиратись на більш надійні функції, які можна використовувати з певними шумами. З недоліків це те що підвищується час навчання модеулі.

AlexNet переміг у конкурсі ImageNet 2012 року з показником помилок у топ-5 15,3%, у порівнянні з моделью на другому місці у якої ей показник 26,2%.

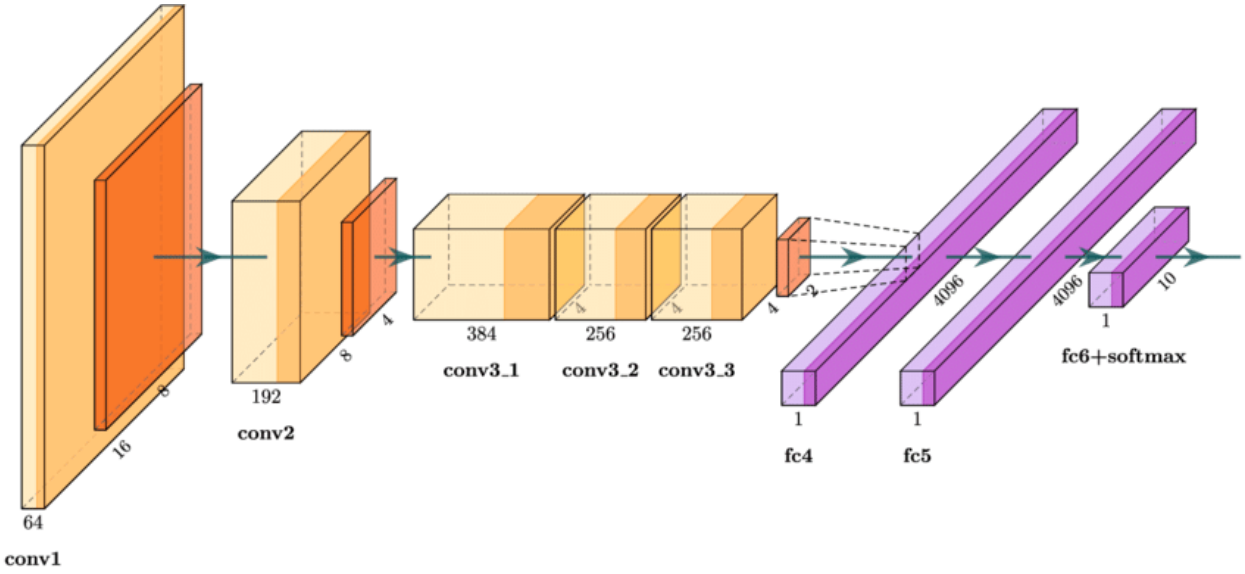


Рис. 2.10. Схема архітектури AlexNet [55]

2.2.4.2. LeNet-5

LeNet-5 - це архітектура CNN, яка має декілька рівнів (Рис. 2.11). Перший рівень — це вхідний рівень — він зазвичай не вважається рівнем мережі, оскільки на цьому рівні нічого не вивчається [56]. Вхідний шар створений так, щоб приймати дані розміром 32x32 і це розміри зображень, які передаються в наступний шар. Наприклад ,зображення набору даних MNIST мають розміри 28x28. Щоб розмір зображень MNIST відповідав вимогам вхідного шару, зображення розміром 28x28 заповнюються додатковими даними.

Для зображень у відтінках сірого, використаних у дослідницькій роботі, значення пікселів було нормалізовано від 0 до 255 до значень від -0,1 до 1,175. Нормалізація гарантує, що пакет зображень має середнє значення 0 і стандартне відхилення 1, переваги цього вбачаються в скороченні часу навчання.

Архітектура LeNet-5 використовує два значущі типи конструкцій шарів:

- Згорткові шари
- Шари підвибірки

У дослідницькій роботі та на зображенні нижче згорткові шари ідентифікуються як «Cx», а шари підвибірки ідентифікуються як «Sx», де «x» — послідовне положення шару в архітектурі. «Fx» використовується для

ідентифікації повнозв'язних шарів. Цей метод ідентифікації шару можна побачити на зображенні вище.

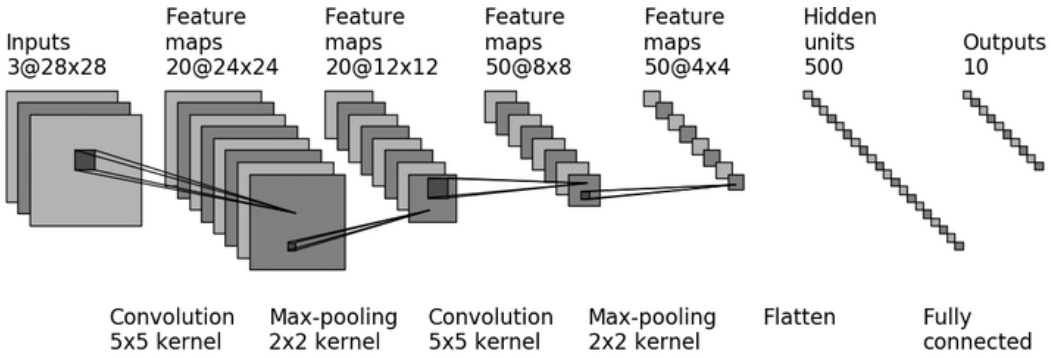


Рис. 2.11. Схема архітектури LeNet-5 [56]

Згортковий шар першого рівня C1 створює як вихідні 6 карт об'єктів і має розмір ядра 5x5. Ядро/фільтр – це ім'я вікна, яке містить значення ваги, які використовуються під час згортки значень ваги з вхідними значеннями. 5x5 також вказує на розмір локального рецептивного поля кожної одиниці або нейрона в згортковому шарі. Розміри шести карт об'єктів, які створює перший шар згортки, становлять 28x28.

Шар підвибірки «S2» слідує за шаром «C1». Шар «S2» зменшує вдвічі розмір карт об'єктів, які він отримує від попереднього шару; це зазвичай відомо як зниження дискретизації.

Шар «S2» також створює 6 карт об'єктів, кожна з яких відповідає картам об'єктів, переданим як вхідні дані з попереднього шару. Це посилання містить додаткову інформацію про шари підвибірки.

2.2.4.3. VGG16

VGG16 - це проста й широко використовувана архітектура згорткової нейронної мережі (CNN), яка використовується для ImageNet датасету [57].

Під час навчання вхідними даними для згорткових шарів є зображення фіксованого розміру 224 x 224 RGB (Рис. 2.12). Віднімання середнього значення RGB, обчисленого на навчальному наборі, з кожного пікселя є єдиною попередньою обробкою, що виконується перед навчанням моделі. Зображення пропускається через набір згорткових шарів, де обчислюються фільтри з дуже

малою розмірністю: 3 × 3, що є найменшим розміром для навчання поняття ліворуч/праворуч, вгору/вниз, центр і має використовується таке ж ефективне рецептивне поле, як 7 x 7. Він глибший, має більше нелінійності та має менше параметрів. В одній із конфігурацій також використовуються фільтри згортки 1 × 1, які можна розглядати як лінійне перетворення вхідних каналів за яким слідує нелінійність. Крок згортки та просторове заповнення згорткових шарів фіксується на 1 піксель для 3 x 3 згорткових шарів, що забезпечує збереження просторової роздільної здатності після згортки. П’ять шарів максимальної вибірки, які слідують за деякими зі згорткових шарів, допомагають у просторовому об’єднанні. Максимальне об’єднання виконується у вікні 2×2 пікселів із кроком 2.

Використовуються три повністю зв'язані шари, які слідують за стеком згорткових шарів (вони мають різну глибину в різних архітектурах): перші два мають по 4096 каналів кожен, третій виконує 1000-сторонню класифікацію ILSVRC і, таким чином, містить 1000 каналів (один для кожного класу). Останнім шаром є шар soft-max. Конфігурація повністю підключених шарів однакова у всіх мережах.

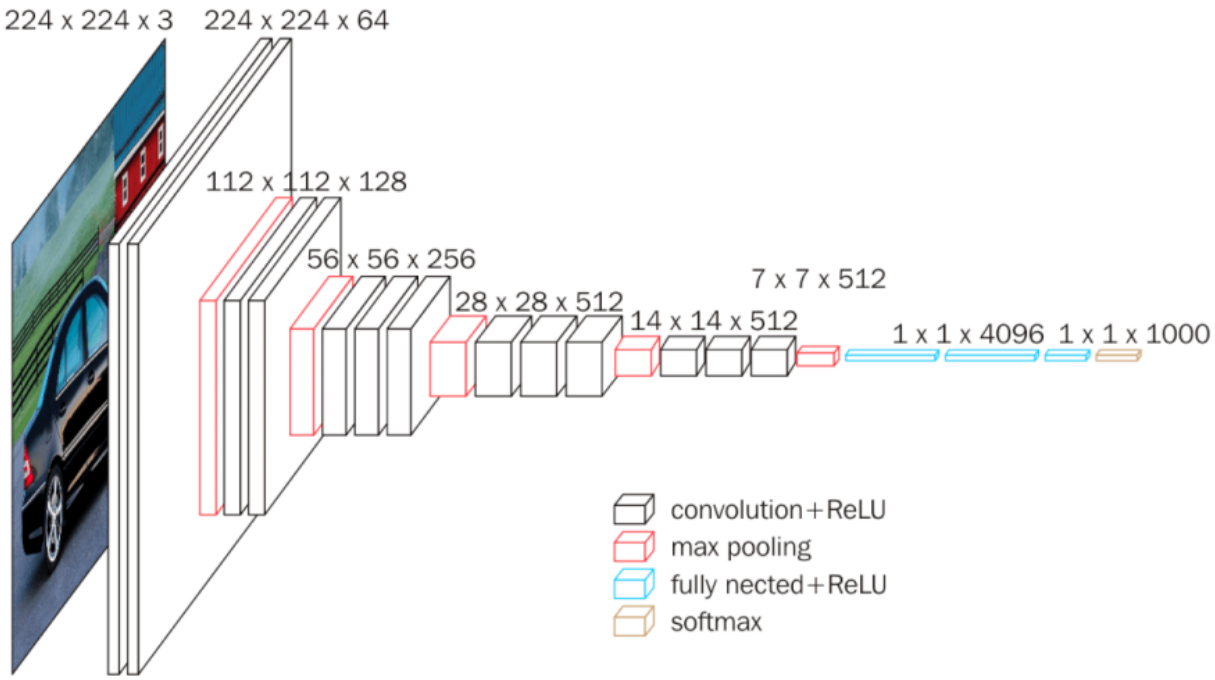


Рис. 2.12. Схема архітектури VGG16 [57]

2.3. Рекурентна нейронна мережа

Рекурентна нейронна мережа (RNN) — це тип штучної нейронної мережі, який використовується для аналізу послідовних даних або даних, які мають форму часових рядів [58]. Цей алгоритм є алгоритмом глибинного навчання і використовуються для вирішення задач у яких дані у формі послідовностей чи часових рядів. Найбільш відомими задачами є переклад, обробка природної мови (NLP), розпізнавання мовлення, тощо. Вони відрізняються від інших типів мереж можливістю “запам'ятовувати” попередні дані та використовувати її для покращення навчання. Рекурентні нейронні мережі беруть інформацію з попередніх вхідних даних, щоб впливати на поточний вхід і вихід. Інші види мереж припускають, що дані що передаються в мережу незалежні один від одного. У рекурентних нейронних мережах результат залежить не тільки від входів, а й від результатів попереднього опрацювання. Хоча майбутні дані також можуть бути корисними для визначення результату даної послідовності, односпрямовані рекурентні нейронні мережі не можуть врахувати такі дані в своїх прогнозах.

2.3.1. Рекурентна нейронна мережа проти нейронної мережі з прямим зв'язком

«Згорнута» RNN представляє всю нейронну мережу, тобто опрацьовану всю послідовність даних. «Розгорнутий» вигляд представляє окремі шари або часові кроки нейронної мережі під час навчання. Кожен шар відповідає одному елементу у даних. Попередні вхідні дані, будуть представлені як прихований стан на наступному етапі часу, щоб передбачити у послідовності наступний елемент.

Також відмінністю рекурентних нейроних мереж від інших видів мереж є те що вони мають спільні параметри на кожному з рівнів мережі. У мережах прямого зв’язку кожен з нейронів має різні ваги. Нейрони у рекурентних нейроних мережах містять однаковий параметр. Для полегшення навчання з

підкріпленням ці ваги також коригуються під час зворотнього поширення та градієнтного спуску.

Алгоритм зворотного поширення через час (Backpropagation through time, BPTT) використовується у рекурентних нейроних мережах для обчислення градієнтів. Цей алгоритм має відмінності від звичного зворотнього поширення. Основними принципами BPTT є ті сами принципи зворотнього поширення: під час навчання моделі обчислюються помилки у порядку від вихідного рівня до вхідного. Ці обчислення допомагають при корегування параметрів мережі. BPTT на відміну від звичайного підходу сумує помилки на кожному кроці часу. Мережі з прямим зв’язком, оскільки мають різні параметри на рівнях, не підсумовують параметри.

Через це рекурентні нейронні мережі можуть бути чутливими до проблем вибухаючи та зникаючих градієнтів. Такі проблеми пов’язані з розміром кроку градієнта. Менше значення градієнту може призвести до продовження корегування параметрів поки градієнт зовсім не зникне. Вибухаючі градієнти можуть зростати занадто швидко і призвести до нестабільності моделі. У випадку вибухаючи градієнтів ваги зростають до занадто великих значень, що робить неможливим наступні підрахунки. Обидві ситуації блокують подальше навчання пожелі. Рішенням таких проблем є зменшення мережі задля спрощення архітектури рекурентної нейронної мережі.

Тут є просте множення вхідних даних і попереднього результату, що проходить через функцію активації Tanh (Рис. 2.13).

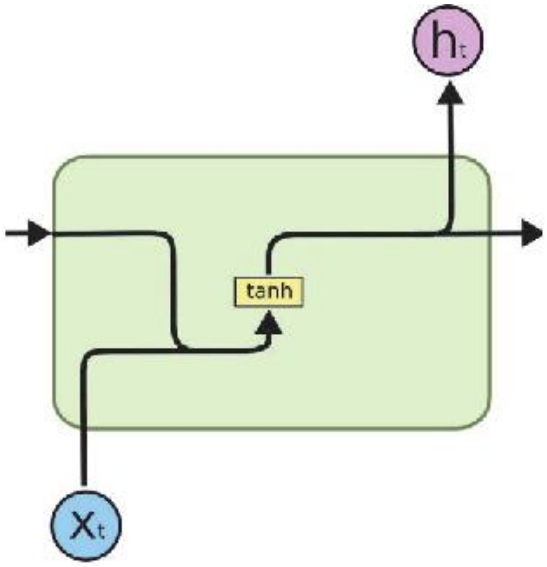


Рис. 2.13. Схема архітектури Simple RNN [58]

Двонаправлені рекурентні нейронні мережі (BRNN) [59]: це варіант архітектури мережі RNN. На відміну від рекурентних нейронних мереж з одним напрямком, корисна інформація може надходити як від попередніх так і від майбутніх станів. Це підвищує точність прогнозів, але може використовуватись не у всіх за стосунках. Вони не можуть бути використані одразу під час збору даних, оскільки дані з майбутнього ще не є доступними.

2.3.2. Довга короткострокова память

Довга короткострокова пам'ять (Long short term memory, LSTM) - це популярна архітектура RNN, яку представили Зепп Хохрайтер і Юрген Шмідхубер як вирішення для проблеми зникаючого градієнта [60]. У своїй статті вони працювали над рішенням проблеми довгострокових залежностей. Проблемою, яку вони намагалися вирішити, було те що якщо попередній стан, який впливає на поточний прогноз знаходиться досить давно, модель RNN може бути не в змозі точно передбачити поточний стан, через відстань між елементами даних. Наприклад, у обробці текстів, контекст який був кількома реченнями раніше, ускладнює або унеможливлює для RNN зв’язання інформації. Щоб вирішити це, LSTM використовує три вентилі – оновлення,

забування та вихідний. Ці вентилі керують потоком інформації, необхідної для прогнозування виходу в мережу.

Використовується шлюз оновлення, забування та вихідний. Шлюз оновлення вирішує чи передавати попередній стан до наступного елементу рекурентної мережі чи ні. Інші шлюзи це додаткові математичні операції на входах. Таким чином, загалом LSTM представив 2 математичні операції з 2 новими наборами ваг (Рис. 2.14).

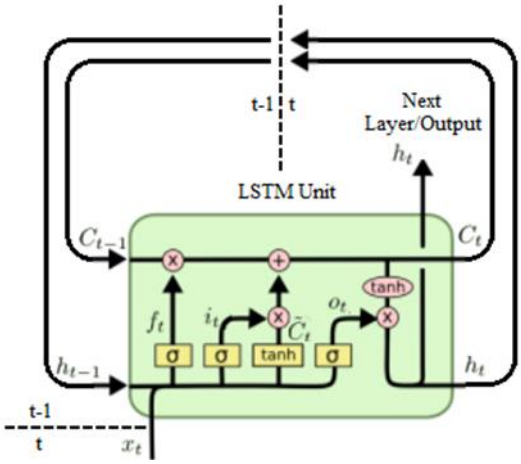


Рис. 2.14. Схема архітектури LSTM [63]

2.3.3. Вентильний рекурентний вузол

Вентильний рекурентний вузол (Gated recurrent unit, GRU) - це варіант RNN, який подібний до LSTM, оскільки він також працює для вирішення проблеми короткочасної пам’яті моделей RNN [61]. Замість використання інформації, яка регулює контекст, вона використовує приховані стани, і замість трьох вентилів має два — шлюз забування та шлюз оновлення. Подібно до вентилів у LSTM, елементи скидання та оновлення контролюють, скільки і яку інформацію зберігати.

Шлюз оновлення вирішує чи передавати попередній стан до наступного елементу рекурентної мережі чи ні. Шлюз забування — це додаткові математичні операції з новим набором ваг (Рис. 2.15).

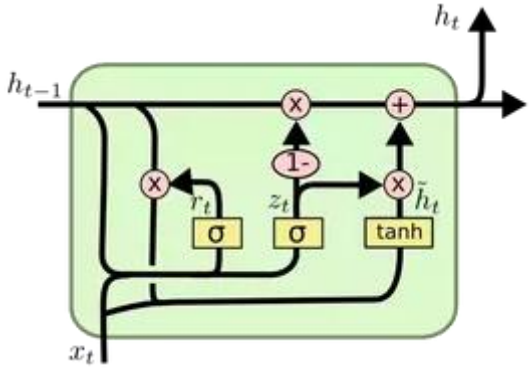


Рис. 2.15. Схема архітектури GRU [64]

2.4. Трансформер

Трансформер – це модель глибинного навчання, яка використовує механізм самоуваги, диференційно зважуючи значущість кожної частини вхідних даних. Він використовується в основному в областях обробки природної мови (NLP) і комп'ютерного зору (Рис. 2.16) [62].

Як і RNN, трансформер призначені для обробки послідовних вхідних даних, таких як природна мова, для таких завдань, як переклад і узагальнення тексту. Однак, на відміну від RNN, трансформер не обов'язково обробляють дані по порядку. Механізм уваги забезпечує контекст для будь-якої позиції у вхідній послідовності. Наприклад, якщо вхідними даними є речення природною мовою, трансформер не потрібно обробляти початок речення до кінця. Натомість він визначає контекст, який надає значення кожному слову в реченні. Ця функція забезпечує більше розпаралелювання, ніж RNN, і, отже, скорочує час навчання.

Трансформери були представлені в 2017 році командою Google Brain і все частіше стають моделлю вибору для проблем NLP, замінюючи моделі RNN, такі як LSTM. Додаткове розпаралелювання навчання дозволяє навчатися на більших наборах даних, ніж це було колись можливо. Це призвело до розробки попередньо підготовлених систем, таких як BERT [63] і GPT, які навчалися за допомогою великих наборів мовних даних, таких як Wikipedia Corpus і Common Crawl, і можуть бути добре- налаштований на конкретні завдання.

До трансформерів більшість найсучасніших систем NLP покладалися на RNN, такі як LSTM і GRU, з додатковими механізмами уваги. Трансформери

побудовані на цих технологіях уваги без використання структури RNN, підкреслюючи той факт, що використання лише механізмів уваги може відповідати продуктивності RNN з увагою.

Теоретично інформація з одного елементу може поширюватися довільно далеко вниз по послідовності, якщо в кожній точці стан продовжує кодувати контекстну інформацію про маркер. На практиці цей механізм хибний: проблема зникнення градієнта залишає стан моделі в кінці довгого речення без точної інформації про попередні маркери, яку можна використати. Залежність обчислень маркерів від результатів попередніх обчислень маркерів також ускладнює розпаралелювання обчислень на сучасному обладнанні глибинного навчання. Це може зробити навчання RNN неефективним.

Ці проблеми вирішувалися за допомогою механізмів уваги. Механізми уваги дозволяють моделі витягувати з стану в будь-якій попередній точці послідовності. Рівень уваги може отримати доступ до всіх попередніх станів і зважити їх відповідно до вивченої міри релевантності, надаючи відповідну інформацію про віддалені маркери.

Як приклад можна розглянути переклад людських мов, у яких контекст відіграє основне значення для визначення інформації в тексті. При перекладі фрази з англійської мови на українську, перше слово українське буде скоріш за все залежити від кількох перших слов у фразі англійською.

Однак у моделі LSTM для того щоб отримати перше слово української фрази модель отримує лише останній стан після обрабки англійського слова у вигляді вектора. Це вектор можливо містить усю необхідну інформацію для перекладу. Але на практиці цієї інформації часто буває недостатньо. Тому використовується механім уваги наступним чином: декодер має перелік сіх прихованих станів рекурентної нейронної мережі після обробки фрази англійською, а не лише останній. Після цього він може використати вагові коефіцієнти уваги, які вказують які елементи є важливими для аналізу фрази.

Додатковий механізм уваги підваищеє продуктивність рекурентних мереж. Наприклад, розробка архітектури трансформеру довела, що механізм уваги є

ефективним способом вилучення додаткової інформації. Вони використовують механізм уваги без використання рекурентних мереж і оброблюють всі елементи одночасно, а не передаючи їх один за одним, обчислюючи значення уваги між ними в послідовних шарах. Через те що в механізмі уваги використовується лише інформація про елементи нижчіх рівнів, їх можна обчислювати для всіх міток паралельно, пришвидшуючи обчислення та навчання.

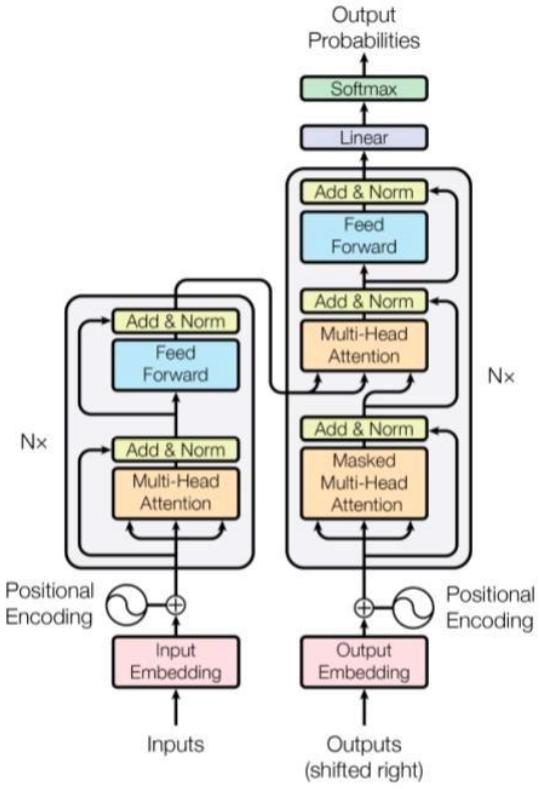


Рис. 2.16. Схема архітектури трансформер [63]

2.4.1. Механізм уваги

Механізм уваги був введений [64], щоб вирішити проблему вузького місця, яке виникає при використанні вектора кодування фіксованої довжини, де декодер має обмежений доступ до інформації, наданої до моделі. Вважається, що це стає особливо проблематичним для довгих або складних послідовностей, де розмірність їх представлення буде такою ж, як і для коротших або простіших послідовностей. Механізм уваги поділено на покрокові обчислення оцінок вирівнювання, ваг та вектора контексту (Рис. 2.17):

- Оцінка вирівнювання: модель вирівнювання приймає закодовані приховані стани, і попередній вихід декодера, щоб обчислити значення, яке вказує, наскільки добре елементи вхідної послідовності узгоджуються з поточним виходом у позиції. Модель вирівнювання представлена функцією, яка може бути реалізована нейронною мережею прямого зв'язку:
- Вага: Вага, обчислюються шляхом застосування операції softmax до попередньо обчислених балів вирівнювання.
- Вектор контексту: унікальний вектор контексту, подається в декодер на кожному кроці часу. Він обчислюється зваженою сумою всіх прихованих станів кодера:

RNN реалізовано як для кодера, так і для декодера в моделях трансформерах. Однак механізм уваги можна переформулювати в загальну форму, яку можна застосувати до будь-якого рішення задач з використанням послідовностей на вході до моделі.

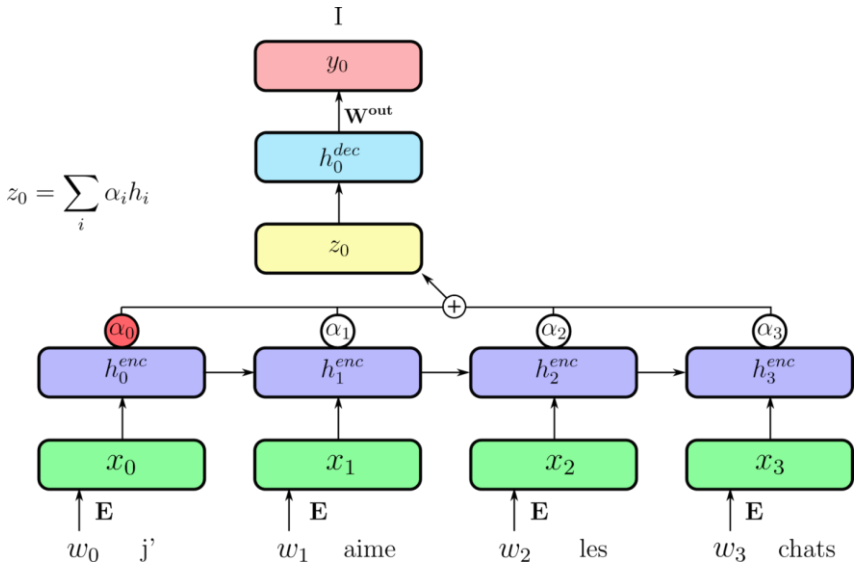


Рис. 2.17. Схема роботи механізму уваги на прикладі машинного перекладу [64]

Механізм загальної уваги використовує три основні компоненти, а саме запити, ключі, і значення. Якщо порівнювати ці три компоненти з механізмом уваги, тоді запит є аналогом до попередньому виводу декодера, значення є

аналогами закодованим входам. У механізмі уваги ключі та значення є одним і тим же вектором. У цьому випадку вектор може бути розглянутий як запит, що виконується до бази даних пар ключ-значення, де ключами є вектори та приховані стани є потрібним значенням.

Механізм загальної уваги виконує такі обчислення:

1. Кожен вектор запиту порівнюється з базою даних ключів для обчислення значення оцінки. Ця операція збігу обчислюється як крапковий добуток конкретного запиту, що розглядається, для кожного ключового вектора:
2. Оцінки передаються через операцію softmax для створення ваг:
3. Узагальнена увага потім обчислюється зваженою сумою векторів значень, де кожен вектор значення поєднується з відповідним ключем:

Найчастіше цей механізм використовується у машинному перекладі. У контексті машинного перекладу кожному слову у вхідному реченні буде приписувати власні вектори запиту, ключів і значень. Ці вектори генеруються шляхом множення представлення кодером конкретного слова, що розглядається, на три різні вагові матриці, які були б згенеровані під час навчання.

2.5. Датасети

Набір даних (або датасет) — це структуровані зібрані дані, що можуть бути використані для аналізу. У разі табличних даних набір даних відповідає одній або кільком таблицям бази даних, де кожен стовпець таблиці представляє певну змінну, а кожен рядок відповідає заданому запису відповідного набору даних. Набір даних перераховує значення для кожної зі змінних, таких як висота і вага об’єкта, для кожного члена набору даних. Набори даних також можуть складатися з колекції документів або файлів [65]. Характеристиками є кількість, розмірність і типи атрибутів або змінних, а також різні статистичні показники, застосовні до них, такі як стандартне відхилення та середнє значення.

Даними можуть бути числа з набору дійсних чисел чи певних дискретних значень, що відображають певну інформацію, наприклад, що представляють

етнічну приналежність людини. Однак також можуть бути відсутні значення, які потрібно якимось чином вказати. Іноді потрібно виконати певні техніки для підготовки датасету до навчання (нормалізація, зміна типів даних, тощо).

У статистиці набори даних зазвичай отримують із фактичних спостережень, отриманих шляхом вибірки статистичної сукупності, і кожен рядок відповідає спостереженням за одним елементом цієї сукупності. Набори даних можуть також створюватися за допомогою алгоритмів з метою тестування певних видів програмного забезпечення. Дані можуть бути згенеровані чи з певними перетвореннями для збільшення кількості даних.

MNIST — це велика база даних невеликих квадратних зображень у відтінках сірого розміром 28×28 пікселів із рукописними одинокими цифрами від 0 до 9 [66]. Загалом вона складається з 70 000 рукописних зображень цифр, причому навчальний набір містить 60 000 зображень, а тестовий набір — 10 000. Усі зображення позначаються відповідною цифрою, яку вони представляють. Всього існує 10 класів цифр (від 0 до 9).

ImageNet – це велика візуальна база даних, призначена для використання в дослідженнях програмного забезпечення для розпізнавання візуальних об’єктів [67]. Більше 14 мільйонів зображень були анотовані вручну проектом, щоб вказати, які об’єкти зображені, і щонайменше на одному мільйоні зображень також надано обмежувальні рамки. ImageNet містить понад 20 000 категорій.

2.5.1. Датасет ЕЕГ

Ці дані містять записи ЕЕГ суб’єктів, які виконують дії хапання та підняття (GAL) [68]. Для кожного сеансу GAL існує 6 подій:

- HandStart - початок руху руки до певного об'єкту дослдужваною особою.
- FirstDigitTouch - досліджувана особа вперше торкається певного об’єкта на столі.
- BothStartLoadPhase – стискання двома пальцями об’єкта досліджуваною особою.

- LiftOff – підняття двома пальцями об’єкта зі столу у повітря на певну висоту.
- Replace – повернення об’єкта на стіл досі тримаючи об’єкт двома пальцями.
- BothReleased - досліджувана особа має відпустити об’єкт і припинити його тримати двома пальцями.

Дані збираються з 32 каналів, які визначені міжнародними стандартами [69] (рис. 2.18), які описані в табл. 2.1.

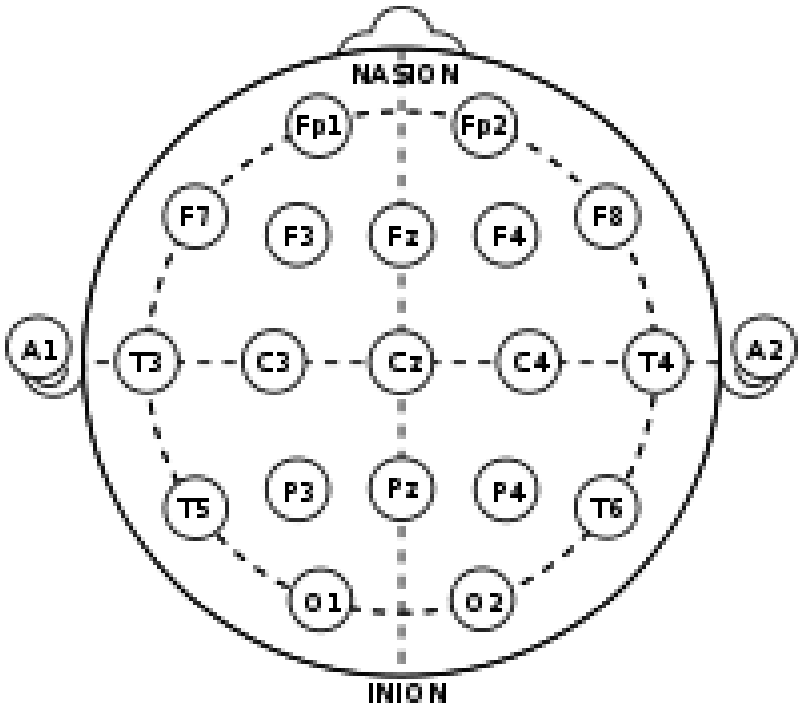


Рис. 2.18. Розміщення електродів ЕЕГ

Таблиця 2.1

Короткий опис найбільш поширених типів електродів

Електрод	**Опис**
FP2 FP1	Електроди поміщаються у префронтальній корі. Ця частина охоплює передню частину лобової частки. Вважається що є зв’язок між особистістю, волею людини до життя, та функціями префронтальної кори. Ця частина відповідає за планування складної когнітивної поведінки (такої як самовираження особистості) , прийняття рішень, соціальну поведінку та контроль певних аспектів мови.
C4 C3	Електроди розміщуються у центральній частині.
P8 P7	Електрод розміщується у тім'яній частині. Ця частина відповідає за інтеграцію сенсорної інформації. Інформація включає орієнтування у просторі і навігацію. Дані, такі як дотики, температура та біль, отримуються зі шкіри передаються через таламус до тім'яної частки. Також деякі області тім'яної частки залучені до обробки мови.
O2 O1	Електрод розміщується у потиличній частині. Ця частина відповідає за візуальну обробку і містить більшу частину області зорової кори.

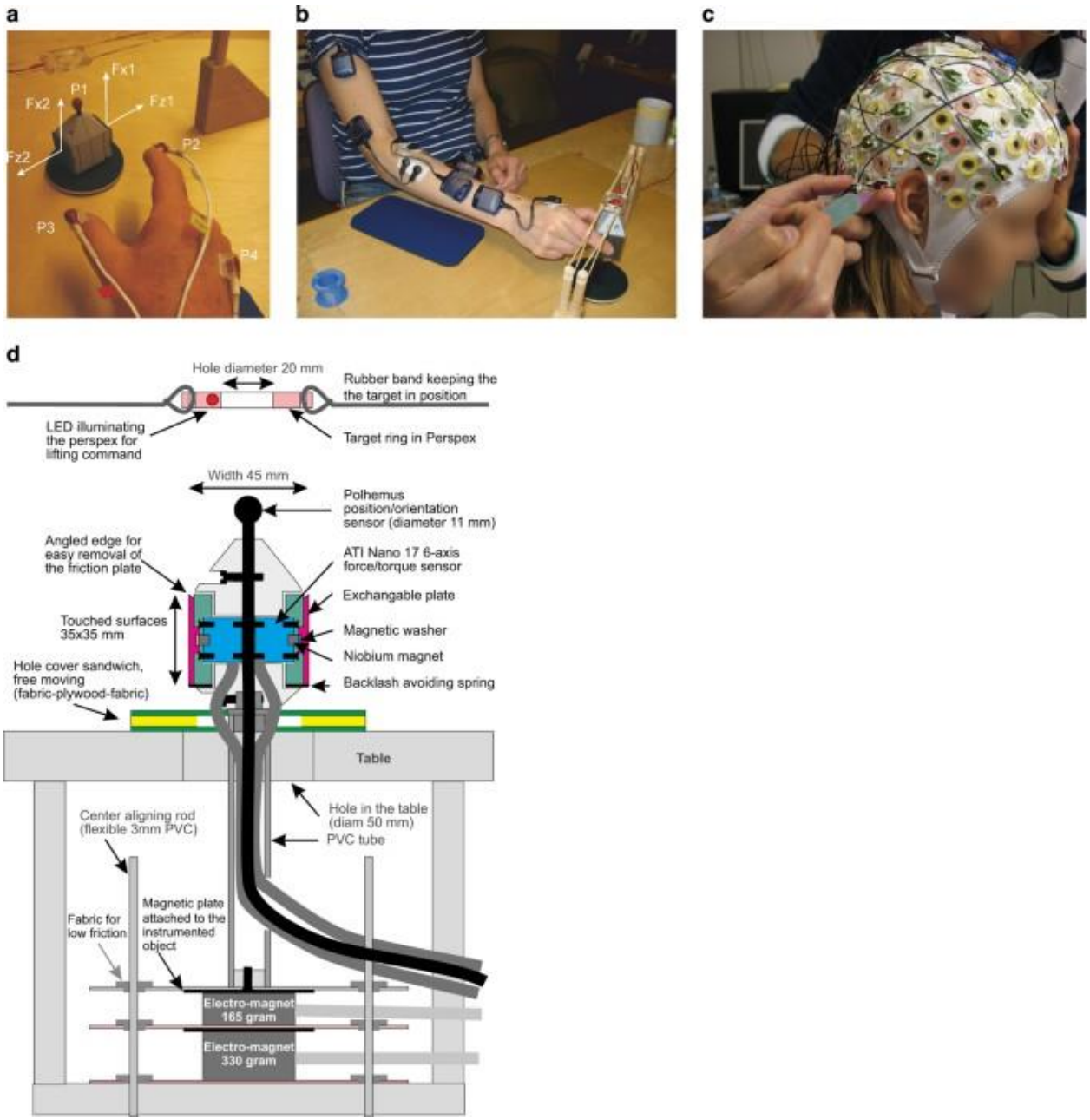


Рис. 2.19. Схема методів експерименту [68]:

Дванадцять учасників виконували серії підйому, в яких вага об’єкта (165, 330 або 660 г), поверхневе тертя (наждачний папір, замша або шовк) або обидва непередбачувано змінювалися між випробуваннями, що викликало зміни в координації сил кінчиків пальців (Рис. 2.19). У кожному із 3936 випробувань учаснику було запропоновано потягнутися до об’єкта, схопити його великим і вказівним пальцями, підняти і утримувати на пару секунд, повернути на опорну поверхню, відпустити, і, нарешті, повернути руку у визначене положення спокою. Були записані ЕЕГ дані, ЕМГ (п’ять м’язів рук і кисті) дані, та

тривимірне положення кисті та об’єкта, а також силу/крутний момент на обох контактних пластинах. За допомогою даних з сенсорів додавалися мітки, які визначають фактичну подію, тобто кожен рух. ЕЕГ дані збираються з частотою 500 ГЦ.

Висновки до розділу 2

Найбільш прогресивною та популярною сферою штучного інтелекту є машинне навчання, метою якого є створення моделей за допомогою зібраних даних для застосування з різною метою у додатках. Спеціально навчені моделі можуть використовувати вхідні дані для швидкого визначення результату. Глибинне навчання на меті якого є створення складних систем, таких як штучні нейронні мережі, для розрахунків та знаходження більш складних залежностей на великих наборах даних є інноваційним підходом для вирішення задач. Такі підходи використовують більше ресурсів, але здатні знаходити більш комплексні кореляції.

Різні підходи та типи шарів для штучних нейронних мереж використовуються для вирішення різних задач:

- для задач аналізу неструктурованих даних використовуються згорткові штучні нейронні мережі (CNN), які використовують просторові залежності даних (наприклад, окремих пікселей зображення) для пошуку об'єктів у просторі та використовують відносне розташування об'єктів для визначення результатів та зменшення кількості ресурсів необхідних для навчання;
- для задач на даних з часовими зв'язками використовуються рекурентні штучні нейронні мережі (RNN), які використовують часові залежності між даними для запам’ятовування та використання зв’язків між окремими даними для визначення більш точного результату. У цих задачах використовується останній стан мережі, який зберігає інформацію про всі об’єкти, для визначення наступних об’єктів;
- для задач аналізу даних із більш складною структурою із наявністю послідовних зв'язків між елементами даних (наприклад, для перекладу) використовуються архітектури трансформери з механізмами уваги, які використовують інформацію про залежність між елементами для більш точного аналізу та зменшення кількості ресурсів. Механізм уваги

допомагає краще зберігати інформацію про зв’язки між об’єктами та уникає проблеми зникаючих градієнтів.

Набори даних, або датасети, мають такі характеристики як розмірність, тип даних та природи цих даних. Також важливі методи їх збору, бо вона може впливати на похибки. Для обрання найбільш ефективних методів обробки даних варто проаналізувати датасет, що буде використовуватись. Датасет варто проаналізувати за розміром даних, типом даних, діапазоном можливих значень, природою даних для обрання необхідних типів шарів штучних нейронних мереж та характеристик цих шарів. Природа даних впливає на тип шарів штучних нейронних мереж. Розмірність даних впливає на характеристики шарів. Попередня обробка даних, використання різних типів шарів штучної нейронної мережі допомагає досягти більш ефективних результатів та зменшити час навчання та кількість ресурсів необхідних для навчання та використання моделей.

РОЗДІЛ 3
РОЗРОБКА МЕТОДІВ АНАЛІЗУ МУЛЬТИМОДАЛЬНИХ ДАНИХ ЕЕГ НА ОСНОВІ ШТУЧНИХ НЕЙРОННИХ МЕРЕЖ

3.1. Інструменти розробки та створення методів AI

Для спрощення процесу створення різних моделей використовуються framework (фреймворки). Фреймворк представляє у вигляді API основні компоненти для створення моделей, полегшує процес їх поєднання та дає змогу використовувати різні налаштування. Фреймворки дозволяють використання власноруч створених модулів у поєднанні з запропонованим. TensorFlow — це безкоштовне програмне забезпечення, яка відкрита та містить бібліотеку інструментів для розробки машинного навчання та штучного інтелекту. Спочатку TensorFlow використовувався лише Google у дослідженнях і при розробці продуктів [70]. Зараз він став відкритим і часто використовується для розробки програмного забезпечення у різних дослідженнях і основним його напрямком є створення моделей глибинних нейронних мереж. Це програмне забезпечення поширюється під ліцензією Apache License 2.0 у 2015 році. TensorFlow отримав нову версію під назвою TensorFlow 2.0 у вересні 2019 року.

TensorFlow сумісний з багатьма мовами програмування, особливо в Python, а також у Javascript, C++ і Java. Це дозволяє обирати потрібну мову програмування під конкретний додаток вільно використовуючи TensorFlow.

Keras — це бібліотека програмного забезпечення з відкритим вихідним кодом, яка надає інтерфейс Python для штучних нейронних мереж [71]. Keras виступає в якості інтерфейсу для бібліотеки TensorFlow.

Аж до версії 2.3 Keras підтримував декілька серверних програм, включаючи TensorFlow, Microsoft Cognitive Toolkit, Theano та PlaidML. Починаючи з версії 2.4, підтримується лише TensorFlow. Створений для швидкого експериментування з глибинними нейронними мережами, тому його перевагами є простота і зручність у використанні, модульність і змога до розширення. Він був розроблений як частина дослідницьких зусиль проекту

ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System), і його основним автором і супроводжуючим є Франсуа Шолле, інженер Google. Шолле також є автором моделі глибинної нейронної мережі XCeption.

Також перевагою Keras є наявність найбільш поширених реалізацій у вигляді сумісних будівельних блоків, з яких легко можна створювати власні архітектури. Для сумісності використанні такі абстракції та конкретні їх реалізації як шари, функції активації, генератори, оптимізатори. Також наявні інструменти для полегшення роботи з даними. Таким чином зменшується кількість часу на вивчення бібліотеки. Код можна переглянути на ресурсі GitHub, а на форумах містяться обговорення різних реалізацій, що полегшує розробки власних модулів. Keras підтримує мережі різних видів, у тому числі містить реалізації деяких згорткових та рекурентних нейронних мереж. Його можна використовувати у поєднанні з іншими популярними Python бібліотеками.

Keras дозволяє користувачам створювати глибинні моделі на різні платформи для різних рішень: на смартфонах (iOS і Android), в Інтернеті або на віртуальній машині Java. Він підтримує використання переваг обладнання та дозволяє використовувати для тренування кластери графічних процесорів (GPU) і тензорних процесорів (TPU), які є більш ефективними та оптимальним рішенням для навчання глибинних нейронних мереж.

3.2. Процес навчання моделі

Основою навчання моделі на базі машинного навчання є градієнтний спуск. Градієнтний спуск - це ітераційний алгоритм оптимізації, який використовується в машинному навчанні для пошуку найкращих результатів (мінімумів кривої) [72].

Алгоритм є ітеративним, що означає, що потрібно обчислити результати кілька разів задля отримання найбільш оптимального результату. Ітераційна якість градієнтного спуску допомагає графіку підходити до найбільш оптимального рішення для даних. Градієнт визначається швидкістю нахилу або

схилу та напрямом спуску. Градієнтний спуск має параметр, який називається швидкість навчання. Ця характеристика може змінюватися під час навчання, щоб підібрати оптимальний результат з найбільшою точністю. Крім того втрати повинні поступово зменшуватися при пошуці оптимального рішення.

Використовуються такі терміни, як епохи, розмір пакету, ітерації, коли набір даних занадто великий, що є частим явищем в машинному навчанні. Вони використовуються через те що неможливо передати всі дані на комп'ютер відразу для обчислень через брак ресурсів. Для подолання цієї проблемі потрібно розділити дані на менші розміри і передати їх на обчислення один за іншим і оновлювати ваги нейронних мереж наприкінці кожного кроку, щоб відповідати обчисленим та проаналізованим даним.

Одна епоха – це процес передачі всього набору даних до нейронної мережі. Оскільки одна епоха занадто велика, щоб відразу передати до моделі на обчислення, дані діляться на кілька менших частин. Передачі всього набору даних до нейронної мережі одноразово недостатньо для отримання результатів. Потрібно передати повний набір даних кілька разів до однієї і тієї ж нейронної мережі, щоб отримати кращі результати. Одна епоха призводить до недостатньо насичених результатів кривої на графіку.

Також варто уникати перенасичення і відслідковувати найкращі результати за допомогою тестування на валідаційному датасеті. Зі збільшенням кількості епох в нейронній мережі змінюється більше разів вага, і крива переходить від кривої недонасиченої до оптимальної, і потім від оптимальної до кривої перенасичення. Кількість епох пов'язана з тим, наскільки різноманітними є дані.

Розмір партії — це загальна кількість навчальних прикладів в одній партії, на яку ділиться кожна епоха.

Ітерації – це кількість партій, необхідних для завершення однієї епохи.

Наприклад, для набору даних з 2000 навчальних прикладів можливо розділити набір даних із 2000 прикладів на пакети по 500, тоді для завершення 1 епохи знадобиться 4 ітерації.

3.3. Підготовка до розробки моделей

3.3.1. Аналіз структури ЕЕГ даних та підготовка даних

Дані ЕЕГ збираються з 32 каналів. Тобто мають розмірність 32 x (кількість вимірювань). Ці дані в датасеті зібрані таким чином, що вимірювання знятті з різних пацієнтів є склеєним та утворюють одну велику матрицю. Дані з мітками мають однакову довжину у 150 значень, що має дорівнювати 0.3 секунді часу.

Для машинного навчання використовуються дані лише числового типу, а не текстового. Тому кожний етап потрібен бути представлений числовим чином. Є декілька технік таких перетворень.

Першою технікою є кодування кожною мітки. Як правило, будь-який структурований набір даних включає в себе декілька стовпців які залежать один від одного та несуть корисну інформацію. Це популярна методика кодування для обробки змінних категорій. У цій техніці кожна мітка визначена унікальним цілим числом на основі алфавітного упорядкування. У цієї техніки є наступні недоліки. Числова репрезентація може створити додаткові зв'язки між категоріями які не існували до кодування. Так категорія закодована числом 1 може вважатись менш релавентною ніж категорія закодована числом 2, тому що 2 більше ніж 1.

Також існує one-hot encoding - це ще одна популярна техніка для вирішення проблеми змінних, які представляють категорію [73]. Цей метод створює додаткову векторну характеристику даних, засновану на кількості унікальних значень у переліку категорій. Кожна унікальна категорія представлена значенням у векторі, а приналежність до цієї категорії визначається самим значенням. Наприклад, якщо значення відповідної категорії у векторі є 1, то ці дані належать до цієї категорії. Якщо 0, то не належать.

Такий тип кодування призводить до декількох можливих проблем. Перша з них це те що техніка призводить до проблеми фіктивної змінної. Це сценарій, в якому змінні дуже співвідносяться один з одним і, отже, результат однієї змінної можна легко передбачити за допомогою залишкових змінних.

Також це призводить до проблеми, відома як мультиколінеарність. Мультиколінеарність виникає там, де існує залежність між незалежними функціями. Мультиколінеарність є серйозною проблемою в моделях машинного навчання, як лінійна регресія та логістична регресія.

Отже, для того, щоб подолати проблему мультиколінеарності, одне зі значень змінних категорій повинна бути скинута.

Була обрана техніка one-hot encoding, тому що було знайдено що даним зібраним в певний проміжок часу має мітки декількох різних етапів. Це можливо по-перше, через помилку при зборі даних чи розмитих та нечітких границях між різними етапами. Це ускладнює визначення на тестових даних тому що відсутня однозначна відповідність даних до мітки. Такий формат даних не підходить для навчання, тому було вирішено не використовувати такі категорії для тренування та навчання. Тому надалі будуть використовуватися 4 категорії з 6.

3.3.2. Нормалізація даних

Теоретично, регресія нечутлива до стандартизації, оскільки будь-якому лінійному перетворенню вхідних даних можна протидіяти шляхом коригування параметрів моделі. Але є дві причини чому цей метод використовують для підготовки даних:

- Стандартизація покращує чисельну стабільність моделі
- Стандартизація може прискорити процес навчання

По-перше, для одновимірних даних X і використання MSE як функції втрат, оновлення градієнта за допомогою градієнтного спуску виглядає так (3.1):

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial Y'} * \frac{\partial Y'}{\partial W} = \frac{2(Y - Y')^T}{N} * X \quad (3.1)$$

Y’ – це передбачення

X знаходиться у формулі градієнтного спуску, що означає, що значення X визначає швидкість оновлення. Тому більший X призведе до більшого стрибка

в градієнтному ландшафті. Тим часом, більший X призводить до меншого W, враховуючи Y (3.2).

$$W = (Y - B) * X^{-1} \quad (3.2)$$

Коли X велике, відстань між початковим W (який вибирається випадковим чином) і глобальним мінімумом, швидше за все, буде малою. Тому алгоритм з більшим X та фіксованою швидкістю навчання, швидше за все, зазнає невдачі, через те що алгоритм робить гігантські стрибки до дуже близької цілі W, тоді як потрібні невеликі кроки. Це перевищення змусить втрати коливатися або вибухати.

По-друге, різні характеристики мають різко різні діапазони, швидкість навчання визначається ознакою з найбільшим діапазоном. Це призводить до ще однієї переваги стандартизації: прискорює процес навчання.

Наприклад, є два діапазони X1 [-1,1] та X2 [-1000,1000]. Швидкість навчання, визначена X2, не є «чудовою» для X1, оскільки X1 є нормальним для набагато більшої швидкості навчання. Використовуються невеликі кроки, в той час як замість них можна було б використовувати дуже великі кроки, що збільшує час тренування. Ми можемо скоротити час навчання, стандартизуючи ці дві функції.

Нейронні мережі можуть використовувати стандартизацію так само, як вона використовується і для регресії. Великі вхідні значення насичують функції активації, такі як сигмовидна або ReLu (негативний вхід). Ці типи функцій активації мають зворотний зв’язок з малим градієнтом або взагалі відсутнім у насиченій області, і тому сповільнюють навчання.

Через це першим кроком для обробки даних є їх нормалізація (або стандартизація).

3.3.3. Аналіз метаданих датасету

Наступним кроком є аналіз метаданих датасету. Датасет складається з 14819347 вимірювань по 32 каналам та з 14819347 мітками по 6 класам для тренування, 1577198 вимірювань по 32 каналам та з 1577198 мітками по 6

класам для валідації та 1589209 вимірювань по 32 каналам та з 1589209 мітками по 6 класам для тесту. У датасеті є 2169 експериментів для тесту, 542 для валідації та 408 для тесту по 150 вимірювань кожен. Отже лише 467850 вимірювань мають хоча б якусь лейблу та можуть використовуватись для корисного навчання. Це 2.6% вимірювань всього датасету. Тому є потреба у ефективному використанні саме розмічених даних під час навчання.

Дані з мітками мають фіксовану довжину 150 елементів. Для навчання та прогнозування вхідні дані повинні мати певний фіксований розмір. Оскільки середня тривалість кожного з етапів складає 0.3 секунд, було вирішено використовувати як вхідні дані серії з 350 послідовних вимірювань. 150 вимірювань до першої мітки, 150 вимірювань з мітками та 50 вимірювань після даних з мітками. Оскільки наша послідовність поєднана у один єдиний датасет потрібно створити ці серії з послідовних вимірювань. Оскільки після закінчення вимірювань минулого пацієнта одразу починаються вимірювання наступного, які ніяким чином не пов'язані з попередніми даними та можуть тільки плутати модель, варто обирати не дуже велику довжину серії. Було вирішено випадковим чином нарізати датасет та створити дочірні датасети даних які стосуються кожної з категорій окремо. Дані були нарізані таким чином, щоб кожний окремий експеримент брався до датасету з запасом з початку та після кінця у 200 вимірювань. Таким чином при навчанні кожна окрема категорія використовує свій датасет та має змогу використовувати випадкове зміщення відносно даних з мітками, щоб отримувати більш випадкові та різноманітні дані для навчання та перевірки точності. Оскільки в цьому випадку кожній серії співвідноситься лише одна мітка, було обрано такий спосіб обрання мітки: вся серія перевіряється чи є у цій серії хоча б одне вимірювання з міткою. Якщо є, ця мітка використовується для всіх даних. У цій роботі використовувались лише дані яким співвідноситься лише 1 категорія для навчання та тесту. Навчання даними без міток не має релевантності. Перевірка на даних без міток також не може визначити категорію. Однією з ідей яка може бути розглянута у майбутньому це створення окремої категорії для даних без

міток. Наприклад, яка означає іншу мозкову активність. За допомогою розбиття датасету на менші датасети по кожній категорії легко контролювати рівномірну по класам подачу даних до моделі під час навчання. Отриманий датасет складається з 6 матриць розмірністю (1491600, 32) з даними по класам (кожному експерименту відповідає 550 вимірювань). Для тренування та валідації було використано 2712 експериментів та для тестів 408 експериментів.

3.3.4. Створення генераторів

Генератори підготовлені таким чином, щоб кожної епохи проходити увесь датасет від початку до кінця. При кожному такому проході повинні бути використані вимірювання кожного з пацієнтів. При кожній епосі генератори беруть рівномірно дані з кожної категорії по черзі з випадково згенерованої послідовності.

Щоб урізноманітнити дані було вирішено обирати початок для послідовності, яка буде використана для тренування, валідації та тестування в певному діапазоні випадковим чином. Діапазон був обраний у 10 значень.

3.4. Огляд структури даних та можливих моделей використання кожної з моделей

3.4.1. Згорткові мережі

Використання згорткових мереж зумовлено тим що дані по кожному з каналів мають просторові зв'язки з іншими каналами. Також оскільки дані мають усі характеристики часового ряду, то зв'язки є не тільки між різними каналами, оскільки мозкові хвилі можуть бути згенеровані у деяких областях та бути зібрані сенсорами які близькі до цієї області, але й залежність між даними у часі. Тобто, дані мають зв'язки як по часу, так і у просторі. Через це можливо використання згорткових нейронних мереж, які використовують просторові зв'язки для покращення результату навчання. Таким же чином як і на картинках, які є основною сферою використання згорткових мереж, на цьому датасеті ЕЕГ можливо отримати більше інформації використовуючи ці зв'язки та вилучаючи особливості більш високого рівня. Тобто, замість того щоб

розрізняти різні об'єкти чи частини об'єктів на картинці, на цьому датасеті модель зі згортковими шарами може розпізнавати збудження та підвищення активності різних областей одна відносно іншої та ознак цієї активності у часі.

3.4.2. Рекурентні архітектури

Використання рекурентних мереж можлива через часовий зв'язок даних. Зазвичай рекурентні мережі використовуються у тому випадку коли попередні дані мають вплив на генерування наступних. Наприклад, як аналіз текстів який є основною сферою використання. Кожне попереднє слово визначає наступне слово. Дані кожного з каналів мають часову характеристику, тому можуть бути одразу передані до рекурентної мережі. Таким чином кожне конкретне вимірювання буде залежати у часі від попередніх вимірювань по кожному з каналів окремо. Замість вектору, який кодує кожне окреме слово у випадку ЕЕГ даних таким вектором є інформація з 32 каналів. Тобто кожен наступний зріз даних по 32 каналам може залежати від попереднього зрізу по 32 каналам і ці залежності у часі та можуть бути використані моделлю. Таким самим чином існують залежності через генерацію сигналів у різних областях та подальшу передачу сигналу в мозку з одного регіону в інший. Отже існують часові зв'язки як і між окремими вимірюваннями, так і між більш високими ознаками. Ці ознаки можуть бути знайдені за допомогою попередньої обробки згорточними мережами та передачу даних до рекурентних моделей, які можуть використовувати часові зв'язки між знайденими ознаками більш високого рівня.

У цій роботі були використані такі РНН архітектури як:

- GRU
- LSTM
- Simple RNN

Ці архітектури, в залежності від конфігурацій, можуть повертати як тільки останній стан так і більше інформації про внутрішній стан. Останній стан зазвичай використовуються в задачах генерації, в яких передбачається кожен новий елемент. API Keras дозволяє отримати доступ до внутрішніх даних

архітектури, які можуть бути корисними або навіть необхідними при розробці складних рекурентних нейронних мереж, таких як модель кодер-декодер.

Кожний елемент LSTM може вивести один прихований стан h для кожного входу. Це можна зробити, встановивши атрибут return_sequences на True під час визначення шару LSTM. return_sequence=True повинен бути використаний під час використання декількох шарів LSTM підряд так, щоб другий шар LSTM мав тривимірну послідовність введення. Також це налаштування може бути корисним, щоб отримати доступ до послідовності вихідних даних прихованого стану під час прогнозування послідовності виводів із щільним вихідним шаром.

3.4.3. Механізм уваги

Також для побудови моделі можуть використовуватись техніки з моделей трансформерів. Оскільки дані з кожного каналу та кожне конкретне вимірювання може мати власну вагу у загальному контексті. Таким чином деякі частини сигналу можуть бути не пов'язані з міткою і мати іншу причину та природу. Ці дані не будуть релевантними для визначення мітки, яка використана для цих даних.

Наприклад, коли узагальнений механізм уваги представлений послідовністю слів, він бере вектор запиту, приписуваний якомусь конкретному слову в послідовності, і оцінює його за кожним ключем у базі даних. При цьому він фіксує, як слово, що розглядається, пов’язане з іншими в послідовності. Потім він масштабує значення відповідно до ваги уваги (обчисленої на основі балів), щоб зберегти увагу на тих словах, які мають відношення до запиту. При цьому він виводить увагу на слово, яке розглядається.

Техніки, що використовуються трансформерами такі як увага, можуть дати гарний результат, тому що вони можуть на відміну від нейронних мереж, визначати контекст та значимість і вплив кожного окремого вимірювання на інші вимірювання, таким самим чином як для машинного перекладу. Також попереднє використання згорткових мереж перед техніками трансформерів

може використовуватися для визначення контексту та впливу не окремих вимірювань, а вищих ознак, тобто вже проаналізованих груп вимірювань.

3.5. Опис розроблених моделей та модифікацій

3.5.1. Рекурентні архітектури

Однією з базових моделей для порівняння було обрано найбільш розповсюджені архітектури RNN (SimpleRNN, GRU, LSTM) (*RNN*_Dense). Найбільш поширені RNN архітектури були використані на даних ЕЕГ. Ця модель дає на виході останній стан і передає його до повнозв’язних шарів, які класифікують послідовність саме на основі останнього стану (Додаток А).

3.5.2. Рекурентні архітектури з прихованими станами

Іншою базовою моделлю для порівняння ефективності інших моделей є розповсюджені архітектури RNN (SimpleRNN, GRU, LSTM) з використанням прихованих станів (*RNN*_Hidden_Dense). Найбільш поширені RNN архітектури були використані, але до повнозв’язних шарів передається не останній стан, а усі приховані стани на кожному кроці обробки RNN. Таким чином повнозв’язні шари отримують більше інформації про обробку сигналу на кожному кроці роботи (Додаток Б).

3.5.3. Рекурентні архітектури з прихованими станами та механізмом уваги

Однією зі створених модифікацій є розповсюджені архітектури RNN (SimpleRNN, GRU, LSTM) з використанням прихованих станів та механізмом уваги (*RNN*_Hidden_Attention_Dense). Найбільш поширені RNN архітектури були використані, але до повнозв’язних шарів передається не останній стан, а усі приховані стани на кожному кроці обробки RNN. Додатковий шар уваги визначає взаємодію між прихованими станами та визначає залежність між якими станами буде корисною для визначення результату. Таким чином повнозв’язні шари отримують більше інформації про обробку сигналу на кожному кроці роботи (Додаток В).

3.5.4. Рекурентні архітектури зі згортковими шарами

Іншою створеною модифікацією є розповсюджені архітектури RNN (SimpleRNN, GRU, LSTM) з використанням згорткових шарів до рекурентних (CNN_*RNN*_Dense). Найбільш поширені RNN архітектури були використані на даних ЕЕГ. До самої обраної архітектури було використано згортковий шар для обчислення інформації більш високого рівня. А саме взаємодію даних з різних каналів та значень з які розташовані поряд. Таким чином до RNN архітектури вже передається інформація про характеристики послідовності. Ця модель дає на виході останній стан і передає його до повнозв'язних шарів, які класифікують послідовність саме на основі останнього стану (Додаток Г).

3.5.5. Рекурентні архітектури зі згортковими шарами та прихованими станами

Також була створена модифікація з розповсюджених архітектур RNN (SimpleRNN, GRU, LSTM) з використанням згорткових шарів до рекурентних та використанням прихованих станів рекурентної мережі (CNN_*RNN*_Hidden_Dense). Найбільш поширені RNN архітектури були використані, перед обраною архітектурою було використано згортковий шар для обчислення інформації більш високого рівня. А саме взаємодію даних з різних каналів та значень які розташовані поряд. Таким чином до RNN архітектури вже передається інформація про характеристики послідовності, а до повнозв'язних шарів передається не останній стан, а усі приховані стани на кожному кроці обробки RNN. Таким чином повнозв'язні шари отримують більше інформації про обробку сигналу на кожному кроці роботи (Додаток Д).

3.5.6. Рекурентні архітектури зі згортковими шарами та прихованими станами з механізмом уваги

Також запропоновані модифікації розповсюджених архітектур RNN (SimpleRNN, GRU, LSTM) з використанням згорткових шарів до рекурентних та використанням прихованих станів рекурентної мережі з використанням

механізму уваги (CNN_*RNN*_Hidden_Attention_Dense). Найбільш поширені RNN архітектури були використані, перед обраною архітектурою було використано згортковий шар для обчислення інформації більш високого рівня. А саме взаємодію даних з різних каналів та значень які розташовані поряд. Таким чином до RNN архітектури вже передається інформація про характеристики послідовності, а до повнозв'язних шарів передається не останній стан, а усі приховані стани на кожному кроці обробки RNN. Додатковий шар уваги визначає взаємодію між прихованими станами та визначає залежність між якими станами буде корисною для визначення результату. Таким чином повнозв'язні шари отримують більше інформації про обробку сигналу на кожному кроці роботи (Додаток Е).

3.5.7. Рекурентні архітектури зі згортоквими шарами до і після

Як альтернатива для механізма уваги були створені модифікації з розповсюджених архітектур RNN (SimpleRNN, GRU, LSTM) з використанням згорткових шарів до рекурентних та використанням згорткових шарів після рекурентної мережі з використанням прихованих станів перед повнозв'язними шарами (CNN_*RNN*_CNN_Dense). Були використані згорткові шари до RNN архітектури і після архітектури. Після RNN архітектури інформація про приховані стани передається до згорткових шарів, які зменшують кількість параметрів повнозв'язної мережі. Також результати згортаються, що допомагає виявляти характеристику прихованих станів і передає їх до повнозв'язних шарів, що може додати інформації на основі якої приймається рішення (Додаток Є).

3.5.8. Використання популярних згорткових архітектур

AlexNet складається з восьми шарів; перші п'ять були згортковими шарами, за деякими з них слідували шари максимального об'єднання, а останні три були повнозв'язними шарами. У ньому використовувалася ненасичена функція активації ReLU.

LeNet-5 складається з семи шарів. На додаток до вхідних даних, кожен інший рівень може тренувати параметри. На малюнку Cx представляє шар згортки, Sx представляє шар підвибірки, Fx представляє повний шар з'єднання, а x представляє індекс шару.

VGG16 також складається зі згорткових шарів та повнозв’язних шарів, але з більшою кількістю параметрів.

3.6. Методи порівняння моделей

Вивчення параметрів функції передбачення та перевірка її на одних і тих же даних є методологічною помилкою: модель, яка б просто повторювала мітки зразків, на яких вона навчалась. Щоб уникнути цього, зазвичай під час виконання експерименту з машинним навчанням зберігається частина доступних даних у вигляді тестового набору.

Під час оцінки різних налаштувань та гіперпараметрів для оцінювачів, все ще існує ризик переобладнання тестового набору, оскільки параметри можна налаштувати, доки оцінювач не працюватиме оптимально. Таким чином, знання про тестовий набір можуть «протікати» в модель, і показники оцінки більше не повідомляють про ефективність узагальнення. Щоб вирішити цю проблему, ще одну частину набору даних можна представити як так званий «перевірочний набір» або “валідаційний набір”: навчання триває на тренувальному наборі, після чого проводиться оцінка на валідаційному наборі, і коли експеримент здається успішним, остаточну оцінку можна зробити на тестовому наборі.

Однак, розділяючи доступні дані на три набори, ми різко зменшуємо кількість вибірок, які можна використовувати для вивчення моделі, і результати можуть залежати від конкретного випадкового вибору для пари наборів (навчання, перевірка).

Рішенням цієї проблеми є процедура, яка називається перехресною перевіркою (CV). Тестовий набір все ще має бути наданий для остаточної оцінки, але набір для перевірки більше не потрібен. У базовому підході, який називається k-кратним CV, навчальний набір розбивається на k менших наборів

(інші підходи описані нижче, але загалом дотримуються тих же принципів). Для кожного з k “вибірок” виконується така процедура:

- Модель тренується з використанням (k-1) складок як навчальних даних;
- отримана модель перевіряється на решті даних (тобто вона використовується як тестовий набір для обчислення показника продуктивності, наприклад точності).

Показник продуктивності, який повідомляється за допомогою k-кратної перехресної перевірки, є середнім значенням, обчисленим у циклі. Цей підхід може бути дорогим з точки зору обчислень, але не витрачає занадто багато даних (як у випадку фіксації довільного набору перевірки), що є основною перевагою в таких проблемах, як зворотний висновок, коли кількість вибірок дуже мала.

Як критерій успішності моделей була обрана методика аналізу характеристику ROC кривої. Крива ROC (крива робочої характеристики приймача) - це графік, що показує продуктивність моделі класифікації на всіх порогах класифікації. Ця крива відображає два параметри:

Істинний позитивний коефіцієнт (TPR) є синонімом відкликання, і тому визначається наступним чином (3.3):

$$TPR = \frac{TP}{TP + FN} \quad (3.3)$$

Коефіцієнт хибних позитивних результатів (FPR) визначається наступним чином (3.4):

$$FPR = \frac{FP}{FP + TN} \quad (3.4)$$

У формулах FP - це кількість хибно позитивні результатів, FN - це кількість хибно негативних результатів, TP - кількість істинно позитивних результатів, TN - кількість істинно негативних результатів.

Крива ROC відображає TPR та FPR при різних порогових значеннях класифікації. Зниження порогу класифікації більше елементів класифікує як позитивні, таким чином збільшуючи як хибнопозитивні, так і справжні

позитивні показники. На основі даних будується крива яка має наступний вигляд:

AUC означає «Площа під кривою ROC». Тобто AUC вимірює всю двовимірну площу під усією кривою ROC (інтегральне обчислення) від (0,0) до (1,1).

AUC надає сукупний показник ефективності за всіма можливими порогами класифікації. Одним із способів інтерпретації AUC є ймовірність того, що модель оцінює випадковий позитивний приклад вище, ніж випадковий негативний приклад.

AUC коливається в межах від 0 до 1. Модель, прогнози якої на 100% помилкові, має AUC 0,0; той, чиї прогнози на 100% вірні, має AUC 1,0.

AUC ефективна з наступних двох причин:

- AUC є масштабно-інваріантним показником. Він визначає, наскільки добре оцінені прогнози, а не їх абсолютні значення.
- AUC є класифікаційно-пороговим інваріантним показником. Він вимірює якість прогнозів моделі незалежно від того, який поріг класифікації вибрано.

Також розрізняють мікро та макро усереднення. Різниця між макро і мікро усередненням полягає в тому, що макро однаково зважує кожен клас, тоді як мікро зважує кожен зразок однаково.

Макро-середнє обчислює метрику незалежно для кожного класу, а потім бере середнє, отже, розглядає всі класи однаково, тоді як мікросереднє агрегує внески всіх класів для обчислення середньої метрики.

Моделі обиралися за найбільшою точністю на валідаційних даних та найменшими втратами на валідаційних даних. При остаточному порівнянні результатів я обрав моделі які мали найбільшу точність на валідаційних даних. Після цього підраховував статистичні характеристики розподілу макро AUC показників, такі як середньоквадратичне відхилення та середнє, та порівнював успішність моделей.

Висновки до розділу 3

Для створення моделі був обраний фреймворк Keras на основі TensorFlow з використанням мови Python. Цей фреймворк був обраний через простоту використання вже існуючих шарів та можливість створення та інтегрування нових власних шарів.

Перед навчанням моделей дані попередньо були оброблені для більш зручного використання та нормалізовані для пришвидшення процесу навчання. Для аналізу результатів були використані методи CV з використанням 15 епох для навчання, які допомагають отримати більш стабільні результати моделі через більш ефективний поділ даних. Як показали моделі використання більше ніж 15 епох не є оптимальним, через відносно невелику кількість даних через що моделі перенасичуються даними та не можуть досягти кращого результату за допомогою додаткових епох. Метадані були проаналізовані для обрання найбільш ефективних методів аналізу. Було визначено, що відсоток даних які несуть корисну інформацію невеликий відносно усього датасету. Щоб зменшити час використання CPU для вибору необхідних для тренування даних, датасет було переформатовано. Деякі дані не були використані через свою неоднозначність та неможливість однозначної класифікації. Також були створені генератори для створення мутацій даних та більш ефективного урізноманітнення корисних даних.

Були використані такі архітектури рекурентних штучних нейронних мереж як RNN, GRU та LSTM. Через просторові зв’язки ЕЕГ даних між різними каналами та часові зв’язки ЕЕГ даних було вирішено модифікувати ці архітектури за допомогою використання:

- згорткових шарів до рекурентної штучної нейронної мережі;
- прихованих станів рекурентної штучної нейронної мережі;
- прихованих станів рекурентної штучної нейронної мережі з механізмами уваги;
- згорткових шарів після рекурентної штучної нейронної мережі;

Характеристики для шарів для кожної модифікації обирались схожим чином, щоб порівнювати доцільність кожної модифікації. Прості рекурентні моделі були створені для порівняння з модифікаціями. Також для порівняння були створені архітектури згорткових нейронних мереж для порівняння таких як AlexNet, LeNet VGG16 для вирішення задачі класифікації даних ЕЕГ. Дані про кількість параметрів, витрачений час, середню точність та відхилення точності записувались та зберігались після тренування кожної з моделей.

РОЗДІЛ 4
АНАЛІЗ ТА ПОРІВНЯННЯ ОТРИМАНИХ РЕЗУЛЬТАТІВ

4.1. Аналіз отриманих результатів за різними модифікаціями між архітектурами

4.1.1. Рекурентні архітектури

Таблиця 4.1

Результати базових рекурентних архітектур

Назва використаної архітектури	AUC ROC макро	Кількість параметрів	Час навчання (сек)
SimpleRNN	0.758	761,604	3284
GRU	0.796	1,299,904	1175
LSTM	0.793	1,565,904	1275

За отриманою точністю можна сказати, що модифікація на основі GRU показала себе найкраще, далі йдуть LSTM та RNN (табл. 4.1) (Рис. 4.1).

У цих нейронних мережах використовується останній стан, який зберігає інформацію стосовно попередніх станів. Тому GRU та LSTM впорались найкраще, а моделі з меншими властивостями запам'ятовування мають гірші результати. Останній стан хоча й несе менше інформації, але значно зменшує кількість параметрів необхідних для отримання результату і робить модель меншою за кількістю параметрів та швидшою під час навчання. Хоч проста RNN мережа має меншу кількість параметрів, вона не використовує покращену версію CUDNN Kernel, і використовує стандартну, яка є повільнішою, але використовує менше ресурсів. Тому хоча і має існувати пряма залежність між кількістю параметрів та часу потрібного на тренування моделі, оптимізації допомагають значно скоротити цей час. LSTM має більшу кількість параметрів ніж GRU та через це є трохи повільнішою.

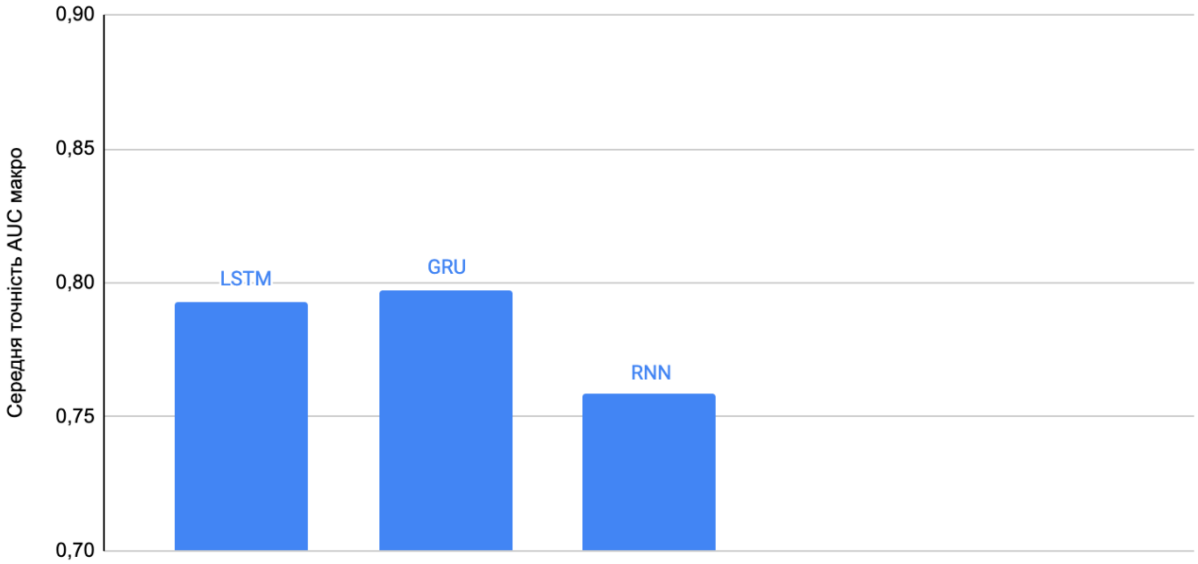


Рис. 4.1. Графік порівняння точності базових рекурентних архітектур

4.1.2. Рекурентні архітектури з прихованими станами

Таблиця 4.2

Результати рекурентних архітектур з прихованими станами

Назва використаної архітектури	AUC ROC макро	Кількість параметрів	Час навчання (сек)
SimpleRNN	0.728	171,772,654	4907
GRU	0.772	172,309,904	3650
LSTM	0.792	172,575,904	3999

За отриманою точністю можна сказати, що модифікація на основі LSTM показала себе найкраще, далі йдуть GRU та RNN (табл. 4.2) (Рис. 4.2).

У цих нейронних мережах використовуються значення усіх проміжних стейтів. LSTM дає перевагу, тому що він має змогу запам'ятовувати довгі зв'язки. Використовується інформація між різними станами, що дає покращення результатів. За допомогою повнозв’язних шарів визначається результат. Результати погіршились у порівнянні зі звичайними RNN архітектурами.

Потрібно використовувати багато ресурсів для тренування та використання моделі. Тому такі моделі не є бажаними для використання.

Хоч проста RNN мережа має меншу кількість параметрів, вона не використовує покращену версію CUDNN Kernel, яка є повільнішою. LSTM має більшу кількість параметрів ніж GRU та через це є трохи повільнішою.

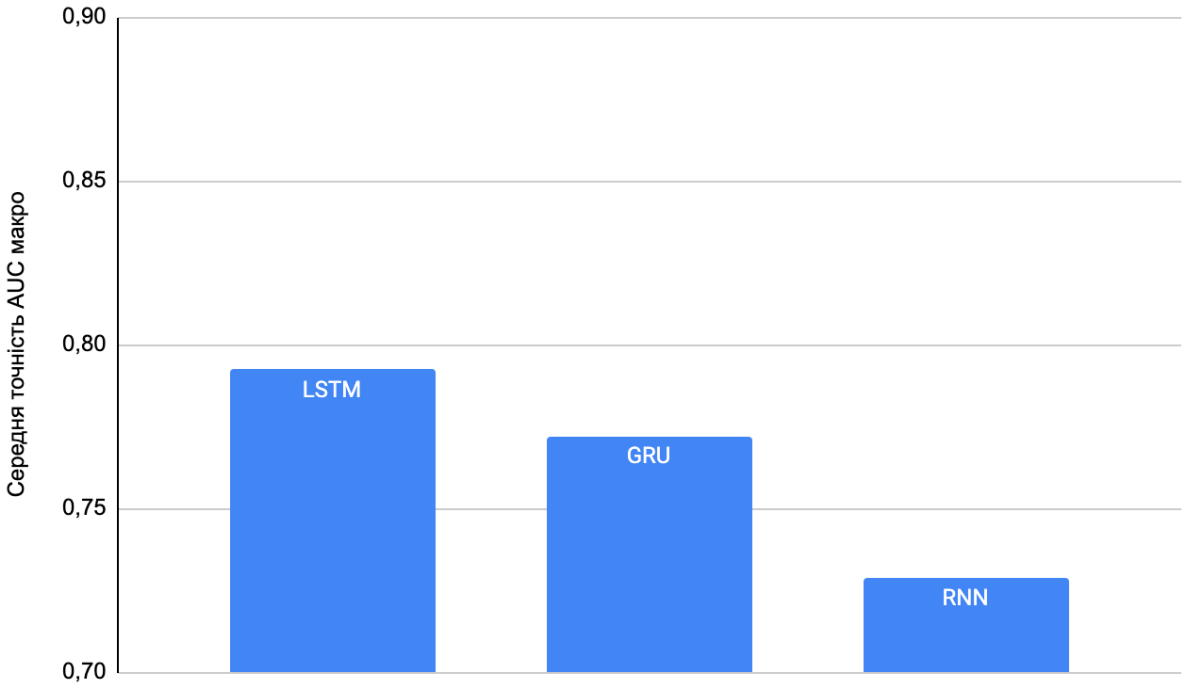


Рис. 4.2. Графік порівняння рекурентних архітектур з прихованими станами

4.1.3. Рекурентні архітектури з прихованими станами та механізмом уваги

Таблиця 4.3

Результати рекурентних архітектур з прихованими станами та механізмом уваги

Назва використаної архітектури	AUC ROC макро	Кількість параметрів	Час навчання (сек)
SimpleRNN	0.854	171,772,654	4594
GRU	0.879	172,310,954	4073
LSTM	0.859	172,576,954	2914

За отриманою точністю можна сказати, що модифікація на основі GRU показала себе найкраще, далі йдуть LSTM та RNN (табл. 4.3) (Рис. 4.3).

Значно покращило результат механізм уваги, який визначає зв'язок між цими прихованими станами додатково. При цьому GRU значно перевершило результат і стало краще за LSTM. LSTM вже має зберігати інформацію про залежність, тому вона не так гарно себе показала. Було досягнуто найкращих результатів точності на основі архітектури GRU. Але модель використовує достатньо багато ресурсів.

LSTM з найбільшою кількістю параметрів зайняв найменше часу для навчання, що потребує детальнішого вивчення.

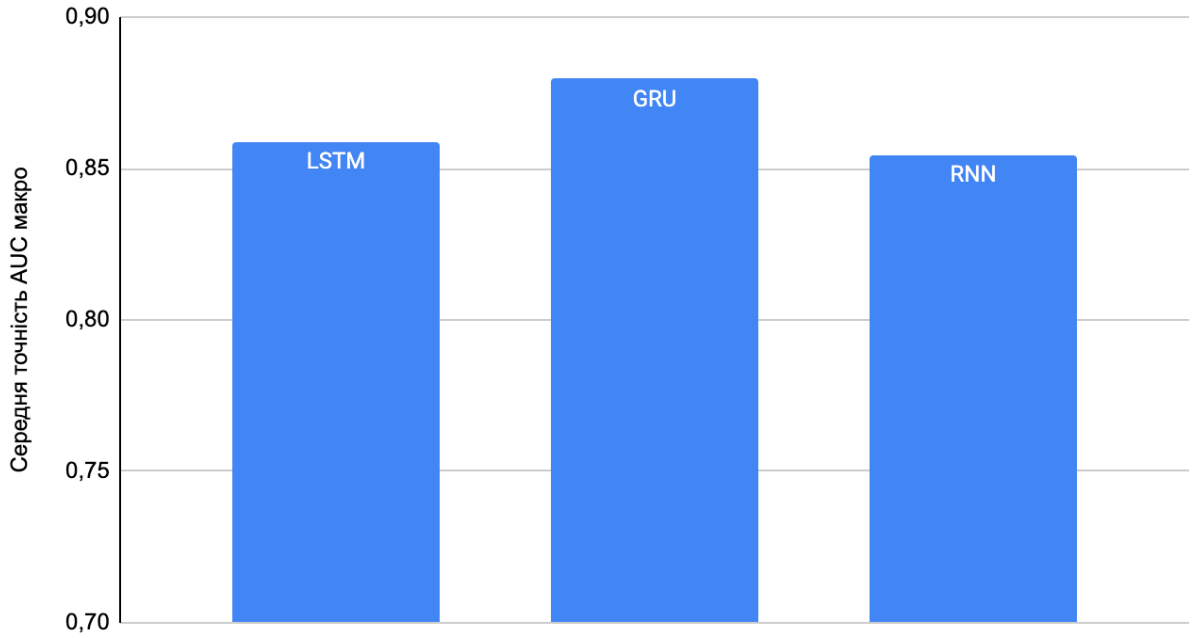


Рис. 4.3. Графік порівняння рекурентних архітектур з прихованими станами та механізмом уваги

4.1.4. Рекурентні архітектури зі згортковими шарами

Таблиця 4.4

Результати рекурентних архітектур зі згортковими шарами

Назва використаної архітектури	AUC ROC макро	Кількість параметрів	Час навчання (сек)
SimpleRNN	0.794	1,359,416	1666
GRU	0.799	3,823,226	899
LSTM	0.807	5,038,616	906

За отриманою точністю можна сказати, що модифікація на основі LSTM показала себе найкраще, далі йдуть GRU та RNN (табл 4.4) (Рис. 4.4).

Використання згорткових шарів перед, допомагає згорнути дані і вивчати ознаки більш високого рівня (такі як характер зміни значень), а не самі значення, що значно покращило результати в порівнянні зі звичайними RNN архітектурами. LSTM має найкращий результат тому що все одно використовується останній стейт, який має збережену інформацію про минулі впливаючи стани найкраще.

Хоч проста RNN мережа має меншу кількість параметрів, вона не використовує покращену версію CUDNN Kernel, яка є повільнішою. LSTM та GRU мають схожі результати по часу навчання.

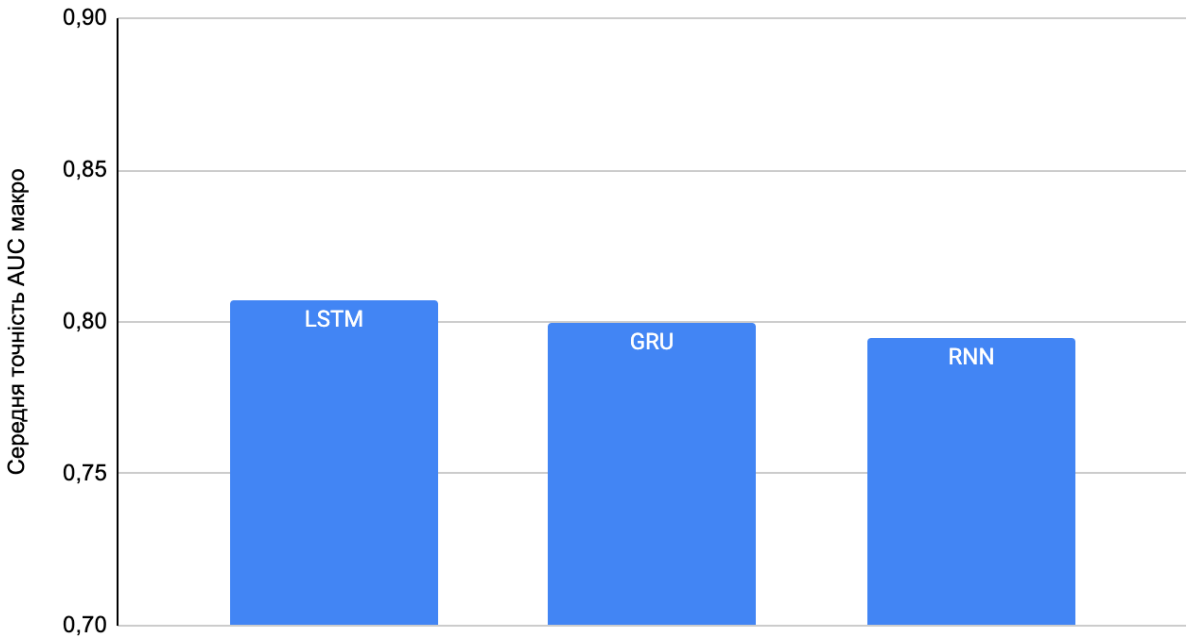


Рис. 4.4. Графік порівняння рекурентних архітектур зі згортковими шарами

4.1.5. Рекурентні архітектури зі згортковими шарами та прихованими станами

Таблиця 4.5

Результати рекурентних архітектур зі згортковими шарами та прихованими станами

Назва використаної архітектури	AUC ROC макро	Кількість параметрів	Час навчання (сек)
SimpleRNN	0.814	11,649,416	1803
GRU	0.873	14,103,266	922
LSTM	0.859	15,328,616	955

За отриманою точністю можна сказати, що модифікація на основі GRU показала себе найкраще, далі йдуть LSTM та RNN (табл. 4.5) (Рис 4.5).

Значно покращились результати ніж у порівняні з HIDDEN states. Згорткові шари аналізують зв'язки між станами на більш високому рівні і отримують

додаткову інформацію. Це може бути використано як заміна атеншен механізмам.

Кількість параметрів значно зменшилась у порівняні з використанням прихованих станів без згортокових шарів, тому що за допомогою згорткових шарів та таких технік як pooling зменшилась кількість даних які передаються до рекурентної мережі. Прихованих станів також меншає і як результат, мережа має змогу бути використаної на машинах з меншою кількістю ресурсів та з меншим часом на навчання.

Хоч проста RNN мережа має меншу кількість параметрів, вона не використовує покращену версію CUDNN Kernel, яка є повільнішою. LSTM та GRU мають схожі результати по часу навчання.

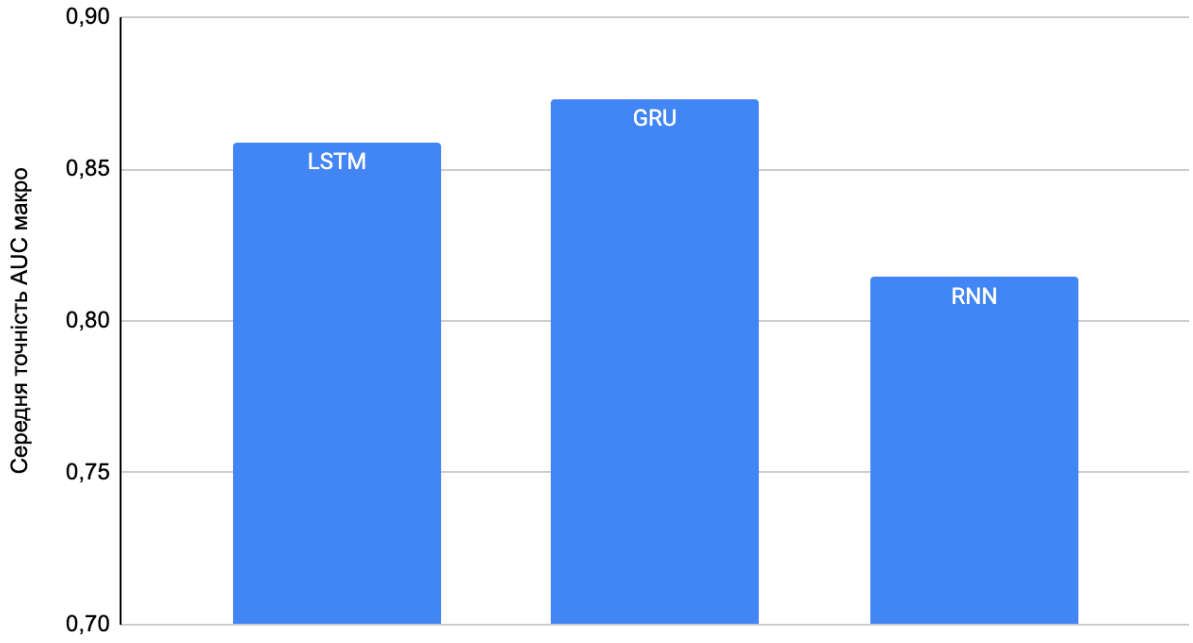


Рис. 4.5. Графік порівняння рекурентних архітектур зі згортковими шарами та прихованими станами

4.1.6. Рекурентні архітектури зі згортковими шарами та прихованими станами з механізмом уваги

Таблиця 4.6

Результати рекурентних архітектур зі згортковими шарами та прихованими станами з механізмом уваги

Назва використаної архітектури	AUC ROC макро	Кількість параметрів	Час навчання (сек)
SimpleRNN	0.865	11,649,851	1766
GRU	0.867	15,329,051	958
LSTM	0.845	15,329,051	946

За отриманою точністю можна сказати, що модифікація на основі GRU та RNN показали себе найкраще, далі йде LSTM (табл. 4.6) (Рис. 4.6).

Механізм уваги допомагає простій RNN визначати зв`язки між значеннями, тому результат покращився у порівнянні з моделлю без механізма уваги. Це вказує на те що саме запам’ятовування інформації про залежність між станами не вистачає RNN. LSTM та GRU, які використовують інформацію про навколишні стейти у собі, не покращили результати тому що вже мають такий механізм у собі. Це вказує на важливість часових зв'язків у ЕЕГ даних і реакції мозку на фізичну активність має деякі часові патерни.

Кількість параметрів значно зменшилась у порівняні з використанням прихованих станів без згорткових шарів, тому що за допомогою згорткових шарів та таких технік як pooling зменшилась кількість даних які передаються до рекурентної мережі.

Хоч проста RNN мережа має меншу кількість параметрів, вона не використовує покращену версію CUDNN Kernel, яка є повільнішою. LSTM та GRU мають схожі результати по часу навчання.

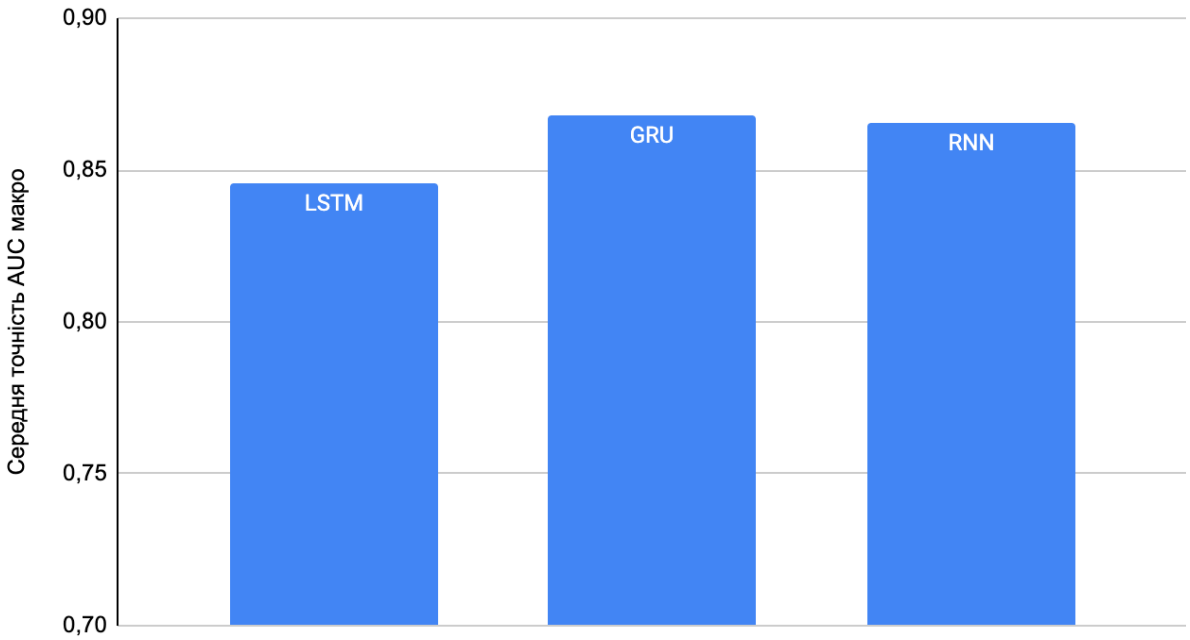


Рис. 4.6. Графік порівняння рекурентних архітектур зі згортковими шарами та прихованими станами з механізмом уваги

4.1.7. Рекурентні архітектури зі згортковими шарами до і після

Таблиця 4.7

Результати рекурентних архітектур зі згортковими шарами та прихованими станами з механізмом уваги

Назва використаної архітектури	AUC ROC макро	Кількість параметрів	Час навчання (сек)
SimpleRNN	0.768	62,148,724	4835
GRU	0.846	62,429,774	1783
LSTM	0.854	62,568,724	1885

За отриманою точністю можна сказати, що модифікація на основі LSTM показала себе найкраще, далі йдуть GRU та RNN (табл. 4.7) (Рис. 4.7).

Значно покращились результати ніж у порівняні з використанням тільки згорткових шарів, окрім простої RNN архітектури. Згорткові шари після архітектури RNN аналізують зв'язки між станами на більш високому рівні і отримують додаткову інформацію. Це може бути використано як заміна атеншен механізмам, бо точність також залишається дуже високою, а ресурсів використовує менше. LSTM та GRU зберігає інформацію про зв'язки між прихованими станами, що дає кращий результат.

Хоч проста RNN мережа має меншу кількість параметрів, вона не використовує покращену версію CUDNN Kernel, яка є повільнішою. LSTM та GRU мають схожі результати по часу навчання.

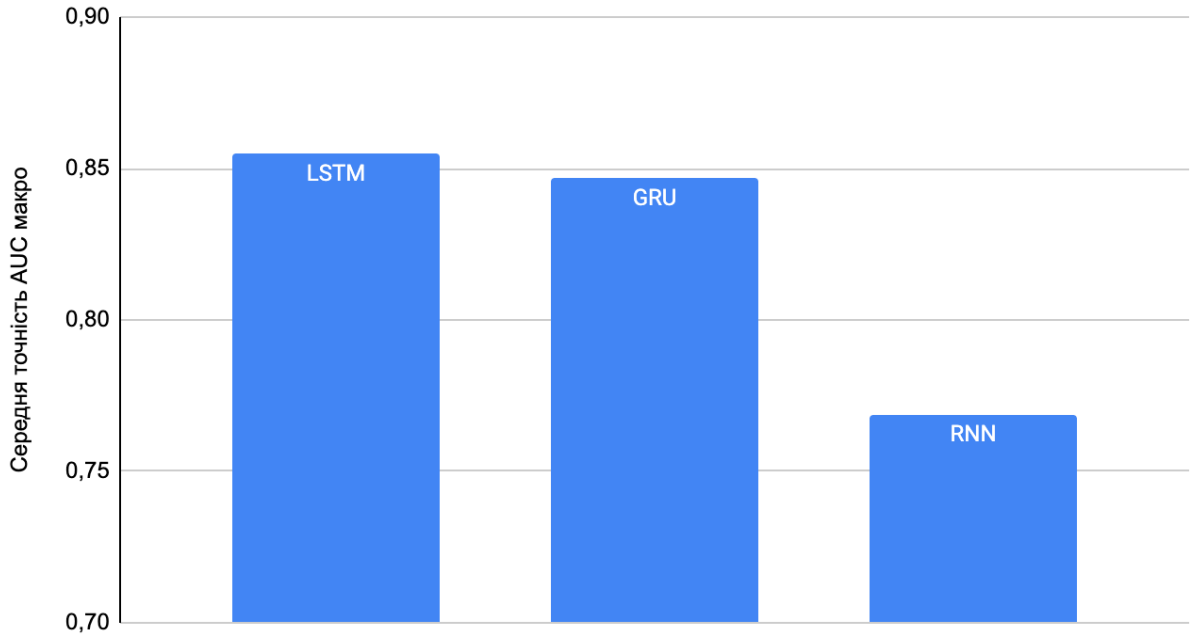


Рис. 4.7. Графік порівняння рекурентних архітектур зі згортковими шарами та прихованими станами з механізмом уваги

4.1.8. Використання популярних згорткових архітектур

Таблиця 4.8

Результати популярних згорткових архітектур

Назва використаної архітектури	AUC ROC макро	Кількість параметрів	Час навчання (сек)
LENET	0.800	25,815,476	895
ALEXNET	0.851	320,446,212	4496
VGG16	-	692,797,764	-

За отриманою точністю можна сказати, що ALEXNET впоралась найкраще (табл. 4.8).

Архітектури, що використовуються для класифікації об’єктів на зображені також можуть використовуватись і для даних ЕЕГ. Це через те що такі архітектури проводять аналіз у просторі у двох вимірах. Для зображень обидва виміри це різні пікселі на картинці. Для ЕЕГ даних це канали та час. Як результат такі мережі здатні розрізняти патерни та досягати досить гарні результати.

У таких мережах значно збільшується кількість параметрів на відміну від використання рекурентних архітектур з різними модифікаціями. Незважаючи на це час навчання не так сильно збільшується через те що такі архітектури можуть більш оптимально використовувати ресурси графічного процесора.

4.2. Аналіз отриманих результатів за різними рекурентними архітектурами між модифікаціями

4.2.1. SimpleRNN

Таблиця 4.9

Результати різних модифікацій на основі SimpleRNN

Назва модифікації	AUC ROC макро	Кількість параметрів	Час навчання (сек)
*RNN*_Dense	0.758	761,604	3284
*RNN*_Hidden_Dense	0.728	171,772,654	4907
*RNN*_Hidden_Attention_Dense	0.854	171,772,654	4594
CNN_*RNN*_Dense	0.794	1,359,416	1666
CNN_*RNN*_Hidden_Dense	0.814	11,649,416	1803
CNN_*RNN*_Hidden_Attention_Dense	0.865	11,649,851	1766
CNN_*RNN*_CNN_Dense	0.768	62,148,724	4835

За результатами використання модифікацій на базову архітектуру RNN (табл. 4.9) було знайдено, що усі модифікації покращили результати. Найбільшої точності досягло використання і згорткових шарів, і механізму уваги з прихованими станами. Варто відзначити, що механізм уваги найбільше покращення у точності на прихованих станах саме архітектурі RNN. Модель, що досягла найкращих результатів у точності також досягла кращих результатів у часі тренування через оптимізації, що використовуються для згоркових шарів та відносно малу кількість параметрів. Кожна з модифікацій покращила результати у точності відносно базової моделі.

4.2.2. LSTM

Таблиця 4.10

Результати різних модифікацій на основі LSTM

Назва модифікації	AUC ROC макро	Кількість параметрів	Час навчання (сек)
*RNN*_Dense	0.793	1,565,904	1275
*RNN*_Hidden_Dense	0.792	172,575,904	3999
*RNN*_Hidden_Attention_Dense	0.859	172,576,954	2914
CNN_*RNN*_Dense	0.807	5,038,616	906
CNN_*RNN*_Hidden_Dense	0.859	15,328,616	955
CNN_*RNN*_Hidden_Attention_Dense	0.845	15,329,051	946
CNN_*RNN*_CNN_Dense	0.854	62,568,724	1885

За результатами використання модифікацій на базову архітектуру LSTM (табл. 4.10) було знайдено, що усі модифікації покращили результати. Найбільшої точності досягло використання механізму уваги з прихованими станами та використання згортокових шарів з прихованими станами. Але згорткові шари допомогли значно зменшити кількість параметрів зменшивши необхідну кількість обчислювальних ресурсів, а отже і час навчання. Варто відзначити, що механізм уваги не завжди покращував результати. Наприклад, він не дав додаткової точності моделі з використанням згорткових шарів та прихованих станів. Це може бути через те що архітектура LSTM вже передбачає знаходження зв’язків у часі, що робить механізм уваги у коротших послідовностях після згорткових шарів не таким ефективним.

4.2.3. GRU

Таблиця 4.11

Результати різних модифікацій на основі GRU

Назва модифікації	AUC ROC макро	Кількість параметрів	Час навчання (сек)
*RNN*_Dense	0.796	1,299,904	1175
*RNN*_Hidden_Dense	0.772	172,309,904	3650
*RNN*_Hidden_Attention_Dense	0.879	172,310,954	4073
CNN_*RNN*_Dense	0.799	3,823,226	899
CNN_*RNN*_Hidden_Dense	0.873	14,103,266	922
CNN_*RNN*_Hidden_Attention_Dense	0.867	15,329,051	958
CNN_*RNN*_CNN_Dense	0.846	62,429,774	1783

За результатами використання модифікацій на базову архітектуру GRU (табл. 4.11) було знайдено, що усі модифікації покращили результати. Найбільшої точності досягло використання механізму уваги з прихованими станами. Варто відзначити, що ця модель має найвищі показники точності. Додаткові згорткові шари допомогли значно зменшити кількість параметрів зменшивши необхідну кількість обчислювальних ресурсів, а отже і час навчання без значної втрати точності. Варто відзначити, що механізм уваги не завжди покращував результати. Наприклад, він не дав додаткової точності моделі з використанням згорткових шарів та прихованих станів. Це може бути через те що архітектура GRU вже передбачає знаходження зв’язків у часі, що робить механізм уваги у коротших послідовностях після згорткових шарів не таким ефективним.

4.4. Порівняння всіх моделей

Найкращий результат досягли декілька моделей і різниця невелика у найкращих результатах показаних моделями, що досягли значення точності більше ніж 85%. Найкращих результатів досягли наступні моделі (рис. 4.8):

- модель на базі найбільш розповсюджених архітектур RNN з використанням прихованих станів та механізмом уваги з використанням GRU, LSTM
- модель на базі найбільш розповсюджених архітектур RNN з використанням згорткових шарів до рекурентних та використанням прихованих станів рекурентної мережі з використанням механізму уваги з використанням SimpleRNN, GRU
- модель на базі найбільш розповсюджених архітектур RNN з використанням згорткових шарів до рекурентних та використанням прихованих станів рекурентної мережі з використанням GRU, LSTM
- модель на базі найбільш розповсюджених архітектур RNN з використанням згорткових шарів до рекурентних та використанням згорткових шарів після рекурентної мережі з використанням прихованих станів перед повнозв'язними шарами з використанням LSTM

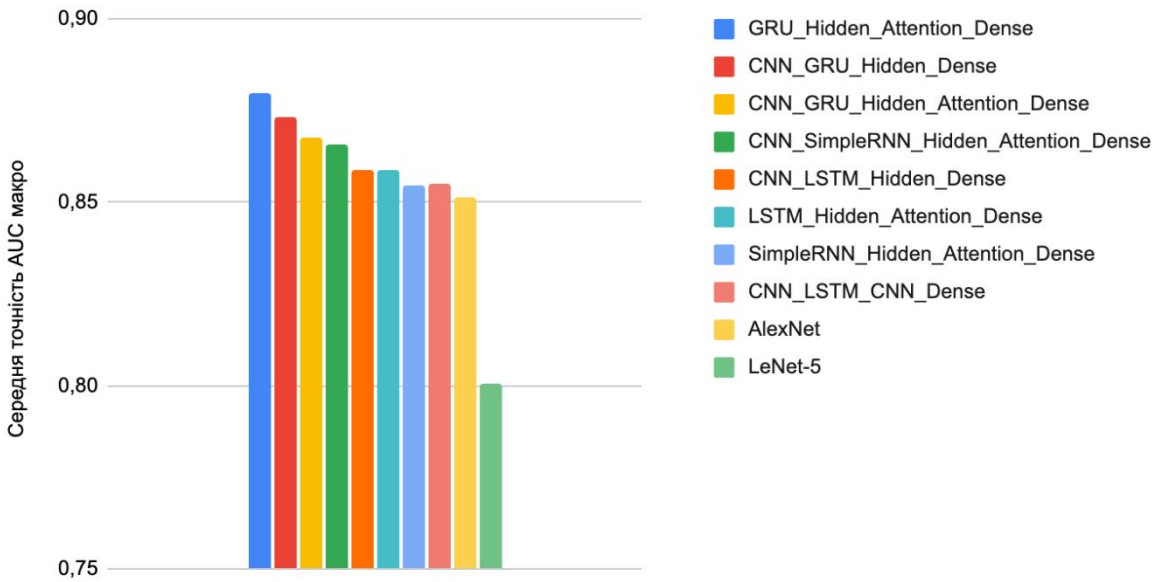


Рис. 4.8. Графік результатів точності модифікацій рекурентних моделей, що досягли >85%, та згорткових архітектур

Усі найкращі результати були досягнуті модифікаціями архітектури GRU. Після цього йдуть деякі модифікації LSTM та SimpleRNN. Успіх саме LSTM зі згортковими шарами до та після пояснюються тим, що LSTM вже запам'ятовує інформацію про зв'язки між даними та не потребує додаткового механізму уваги.

GRU зберігає більше інформації ніж просто RNN, що робить його більш привабливим для модифікації.

Хоча найкраща точність була досягнута без використання згорткових шарів, згорткові шари допомогли значно зменшити кількість параметрів (рис. 4.9). Отже моделі зі згортковими шарами можуть бути більш привабливими, якщо ресурси обмежені. Наприклад, більш легкі моделі з використанням лише згорткових та повнозв’язних шарів для аналізу вже зібраних даних, використання RNN архітектур для аналізу в реальному часі і передбачені в реальному часі, а поєднання CNN та механізму уваги можна спробувати використовувати для класифікації у реальному часі на пристроях з обмеженими ресурсами.

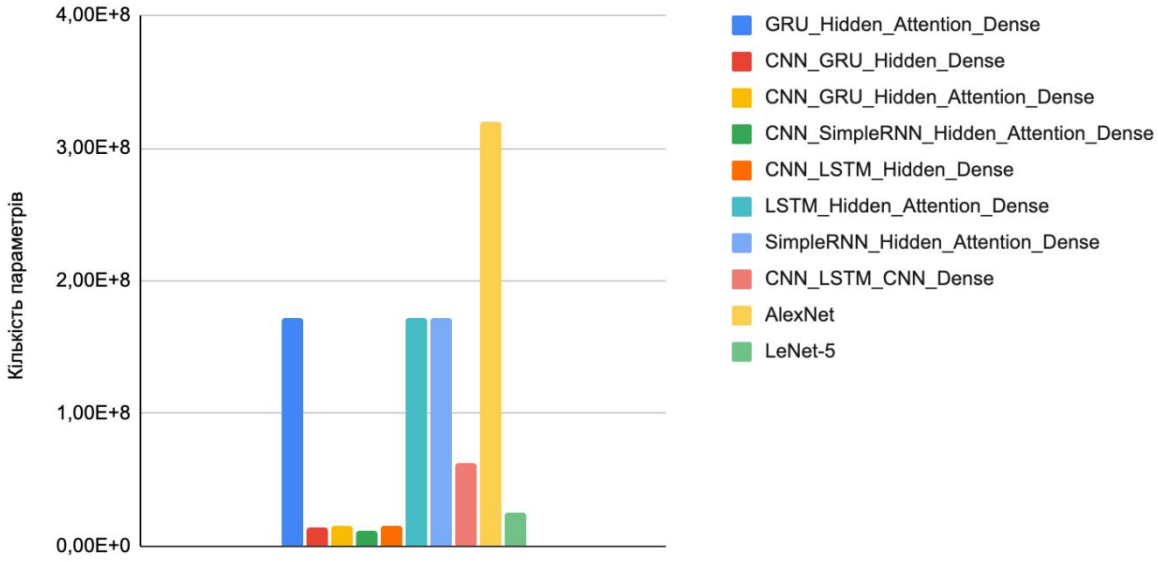


Рис. 4.9. Графік кількості параметрів модифікацій рекурентних моделей, що досягли >85%, та згорткових архітектур

Хоча згорткові шари допомогли значно зменшити кількість параметрів у моделях, час навчання змінився схожим чином, але меншою мірою (Рис. 4.10). Це через те що велику кількість часу займає виконання генераторів, які використовують менш оптимізований та більш повільний CPU для цієї роботи. GPU використовуються по різному на різних моделях в залежності від кількості обчислень та можливостей використання оптимізованих операцій. Так GPU є більш ефективним на згорткових шарах та може використовувати усі переваги кількості ресурсів на GPU тільки коли GPU завантажено більше. При меншому навантажені GPU може бути менш ефективним через втрачену можливість розпаралелювання обчислень. Але менша кількість параметрів зменшує вимоги до обчислювальних ресурсів необхідних для використання моделей.

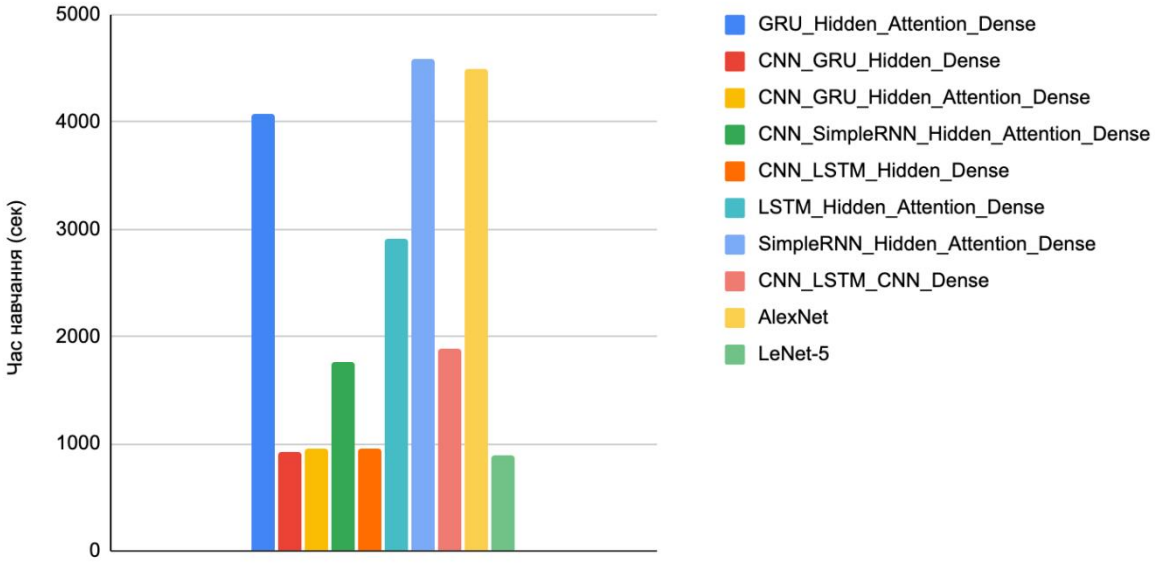


Рис. 4.10. Графік часу навчання у секундах модифікацій рекурентних моделей, що досягли >85%, та згорткових архітектур

Використання рекурентних архітектур краще використовує залежність даних від часу на відміну від згорткових архітектур, що використовуються для аналізу зображень. Через це вони досягають такої ж або кращої точності, але використовують меншу кількість параметрів, отже використовують меншу кількість ресурсів.

Створені моделі можуть бути використані для наступного:

- зменшення кількості значень у векторі, який використовується для навчання та класифікації. Це може дати більш точну відповідь, який саме патерн допомагає визначити наступний рух. В який саме момент він генерується до цієї активності та якою частиною мозку.
- дослідження активності мозку під час інших видів активностей
- дослідження залежностей каналів під час різних видів рухів
- дослідження більш досконалих архітектур, які краще будуть використовувати зв'язки між різними каналами

Висновки до розділу 4

Результати були порівнянні між модифікаціями і між різними архітектурами за наступними характеристиками:

- макро точність за допомогою ROC AUC;
- кількість параметрів кожної моделі;
- час навчання кожної моделі.

Порівнюючи модифікації моделей з базовими рекурентними архітектурами, було виявлено що модифікації є більш ефективними ніж використання простих рекурентних архітектур. Використання згорткових шарів, механізмів уваги у поєднанні з використанням прихованих станів та варіанти їх поєднання показали більшу точність класифікації для всіх базових рекурентних архітектур. Таким чином можна стверджувати, що механізми уваги та згорткові шари допомагають виявляти більше інформації через просторові та часові зв’язки ЕЕГ даних

Порівнюючи всі моделі, було знайдено що деякі моделі досягли схожої високої точності, але мають різні показник по кількості параметрів та часу для навчання. Було виявлено, що приховані шари з механізмами уваги значно покращують точність GRU архітектури, але потребують багато ресурсів. Додаткові згорткові шари допомагають зменшити кількість параметрів, а отже і часу для тренування зі незначною втратою точності.

Порівнюючи модифікації та їх результати їх застосування на різних базових рекурентних архітектурах, було виявлено що різні модифікації по різному впливають на різні базові архітектури. Використання механізму уваги на прихованих станах найбільш ефективно покращила результати RNN, але найкращої точності досягла на GRU.

Архітектури згорткових штучних нейронних мереж, що використовуються для задач зі зображеннями, показали високий результат на даних ЕЕГ тому що вони також мають змогу використовувати зв’язки у двох вимірах. Для даних ЕЕГ це час та простір.

Моделі вони використовують різні характеристики датасету і як результат можуть використовуватись у додатках з різною кількістю обчислювальних ресурсів та у додатках різних типів (використання у реальному часі, використання на вже зібраних даних).

ВИСНОВКИ

У роботі були досліджені основні методи аналізу та класифікацій даних різного виду та на основі них розроблені методи для аналізу даних ЕЕГ. За допомогою розроблених методів була отримана можливість розрізняти деякі люди на основі набору даних мозкової активності альфа-хвиль людей.

Були визначені наступні характеристики даних ЕЕГ, які роблять використання методів штучного інтелекту ефективним інструментом:

- дані є мультимодальними, оскільки одночасно з ЕЕГ виконуються заміри часу кожного з вимірювань, створюється опис дій людини під час вимірів, дані збираються з декількох каналів одночасно.
- дані збираються у великій кількості, у роботі дані для використання були зібрані з 16 каналів одночасно з частотою 500 Гц.
- діяльність мозку має нелінійний характер взаємодії даних різних частин мозку (просторові зв’язки) та можуть мати певні залежності у часі (часові зв’язки), тому є складним предметом для аналізу. Ці залежності можуть бути використані для оптимізації та покращення методів аналізу.

У цій роботі для вирішення проблеми класифікації рухів на основі даних про активність альфа-хвиль утворених мозком, були запропоновані методи машинного навчання, такі як глибинне навчання. Різні інструменти глибинного навчання дають змогу знаходити залежності різного виду у великих наборах мультимодальних даних.

Датасет було проаналізовано за розміром даних, типом даних, діапазоном можливих значень, природою даних для обрання необхідних типів шарів штучних нейронних мереж та обрання характеристик цих шарів. На етапі підготовки даних, зібрані дані було відформатовано та нормалізовано, були створені генератори для рівномірного розподілу корисних даних під час навчання та збільшення їх різноманітності. Через просторові зв’язки між різними каналами та часові залежності у ЕЕГ даних, у роботі були використані різні типи штучних нейронних мереж та їх поєднання:

- згорткові штучні нейронні мережі (CNN), які використовують просторові залежності даних для пошуку об'єктів у просторі та використовують відносне розташування об'єктів для визначення результатів та зменшення кількості ресурсів необхідних для навчання;
- рекурентні штучні нейронні мережі (RNN), які використовують часові залежності між даними для запам’ятовування та використання зв’язків між окремими даними для визначення більш точного результату.
- архітектури трансформери з механізмами уваги, які використовують інформацію про залежність між окремими елементами.

Результати аналізувались за показниками точності після CV. Були використані такі архітектури рекурентних штучних нейронних мереж як RNN, GRU та LSTM. Були розроблені наступні модифікації цих архітектур:

- використання згорткових шарів до рекурентної штучної нейронної мережі;
- використання прихованих станів рекурентної штучної нейронної мережі перед повнозв’язними шарами;
- використання прихованих станів рекурентної штучної нейронної мережі з механізмами уваги перед повнозв’язними шарами;
- використання згорткових шарів після рекурентної штучної нейронної мережі на прихованих станах рекурентної штучної нейронної мережі.

Розроблені модифікації були порівнянні з простим використанням рекурентних нейронних мереж та сучасними архітектурами згорткових глибинних штучних нейронних мереж таких як VGG16, AlexNet, LeNet. Результати були порівнянні за наступними характеристиками:

- макро точність за допомогою ROC AUC;
- кількість параметрів кожної моделі;
- час навчання кожної моделі.

Деяка максимальна точність була досягнута декількома моделями. Моделі використовують різні особливості датасету і мають різні переваги. Використання прихованих станів у поєднанні з GRU та механізмами уваги

досягло максимальної точності, але потребує значних ресурсів для використання та більше часу для навчання. Таку модель можна використовувати для точних прогнозів на більш потужних комп’ютерах.

Додаткові згорткові шари допомагають скоротити необхідну кількість ресурсів і часу на навчання з незначною втратою точності. Як результат вони можуть використовуватись у додатках з незначною кількістю обчислювальних ресурсів та у додатках для опрацювання в реальному часі. У деяких модифікаціях додаткові згорткові шари підвищили точність окремих моделей.

Механізм уваги значно покращив результати простої RNN та покращив її можливості до знаходження зв’язків у даних. Він не дав таких значних покращень архітектурам GRU та LSTM, які мають більш комплексні механізми запам’ятовування зв’язків ніж проста RNN. Саме механізм уваги дозволив досягти максимальної точності.

Розроблені моделі можуть бути використані у різних сферах та для різних досліджень, а саме:

- дослідження патернів та часу активності мозку, за якою можна визначити наступну дію пацієнта;
- дослідження активності мозку під час інших видів активностей;
- дослідження залежностей каналів під час різних видів рухів;
- дослідження більш досконалих архітектур, які краще будуть використовувати зв'язки між різними каналами.

Запропоновані методи аналізу можуть бути використані при розробці нових ЕЕГ пристроїв та додатків для різного призначення.

ПЕРЕЛІК ПОСИЛАНЬ

1. Niedermeyer E. Electroencephalography: basic principles, clinical applications, and related fields / Niedermeyer E., da Silva F. L. (Ред.) // Lippincott Williams & Wilkins, 2005.
2. Tatum W. O. Handbook of EEG Interpretation / Husain, A. M., Benbadis, S. R. // Demos Medical Publishing, 2008.
3. EEG (Electroencephalogram) [Електронний ресурс] - Mayo Clinic - Режим доступу: https://www.mayoclinic.org/tests-procedures/eeg/about/pac-20393875.
4. Saccadic spike potentials in gamma-band EEG: Characterization, detection and suppression / Keren, Alon S.; Yuval-Greenberg, Shlomit; Deouell, Leon Y. // NeuroImage, 2010.
5. Nunez PL Srinivasan R. Electric fields of the brain: The neurophysics of EEG // Oxford University Press, 1981.
6. Principles of brain functioning / Haken H // Springer, 1996.
7. Neurophysiological and computational principles of cortical rhythms in cognition / Wang XJ // Physiological Reviews, 2010.
8. Dynamical systems in neuroscience / Izhikevich EM // Massachusetts: The MIT Press, 2007.
9. Driving fast-spiking cells induces gamma rhythm and controls sensory responses / Cardin JA, Carlén M, Meletis K, Knoblich U, Zhang F, Deisseroth K, et al. // Nature. 459, 2009.
10. The neuronal basis for consciousness / Llinás R, Ribary U, Contreras D, Pedroarena C // Biological Sciences, 1998.
11. Exploring mechanisms of spontaneous functional connectivity in MEG: how delayed network interactions lead to structured amplitude envelopes of band-pass filtered oscillations / Cabral J, Luckhoo H, Woolrich M, Joensson M, Mohseni H, Baker A, et al. // NeuroImage, 2014.
12. Evaluation of the NeuroSky MindFlex EEG headset brain waves data / J. Katona, I. Farkas, T. Ujbanyi, P. Dukan and A. Kovari // IEEE 12th International Symposium on Applied Machine Intelligence and Informatics, 2014.

13. Mindflex Duel [Електронний ресурс]. - NeuroSky, 2020. – Режим доступу: https://store.neurosky.com/products/mindflex-duel.
14. Estermann B. Evaluation of Low-Cost EEG Hardware in a Motor Imagery Based Brain Computer Interface // 2017.
15. Ultracortex Mark IV [Електронний ресурс]. - OpenBCI, 2020. – Режим доступу: https://docs.openbci.com/docs/04AddOns/01-Headwear/MarkIV.
16. Hinrichs H. Comparison between a wireless dry electrode EEG system with a conventional wired wet electrode EEG system for clinical applications / Hinrichs, H., Scholz, M., Baum, A. K., Kam, J. W., Knight, R. T., & Heinze, H. J. // Scientific Reports, 2020.
17. actiCHamp Plus [Електронний ресурс]. - BrainVision, 2022. – Режим доступу: https://brainvision.com/products/actichamp-plus/
18. Sensor Fusion in Time-Triggered Systems, PhD Thesis / Elmenreich W. // Vienna University of Technology, 2002.
19. Applied Fourier Analysis and Elements of Modern Signal Processing Lecture 3 [Електронний ресурс]. - Emmanuel Candes, 2016. – Режим доступу: https://candes.su.domains/teaching/math262/
20. Time-frequency feature representation using energy concentration: An overview of recent advances / Sejdić E.; Djurović I.; Jiang J. // Digital Signal Processing, 2009.
21. Buck J. R. Discrete-time signal processing / Buck J. R., Oppenheim A. V., Schafer R. W. // вид. Prentice Hall, 1999.
22. Капшій О. В. Вейвлет-перетворення у компресії та попередній обробці зображень / Капшій О. В., Коваль О. І., Русин Б. П. // Львів : Сполом, 2008. — 206 с.
23. Lotte F.. A review of classification algorithms for EEG-based brain–computer interfaces: a 10 year update / Lotte F., Bougrain L., Cichocki A., Cler, M., Congedo M., Rakotomamonjy A., Yger F. // Journal of neural engineering, 2018.
24. LeCun Y. Deep learning / LeCun Y., Bengio Y., Hinton G. // nature, 2015. - С. 436-444.

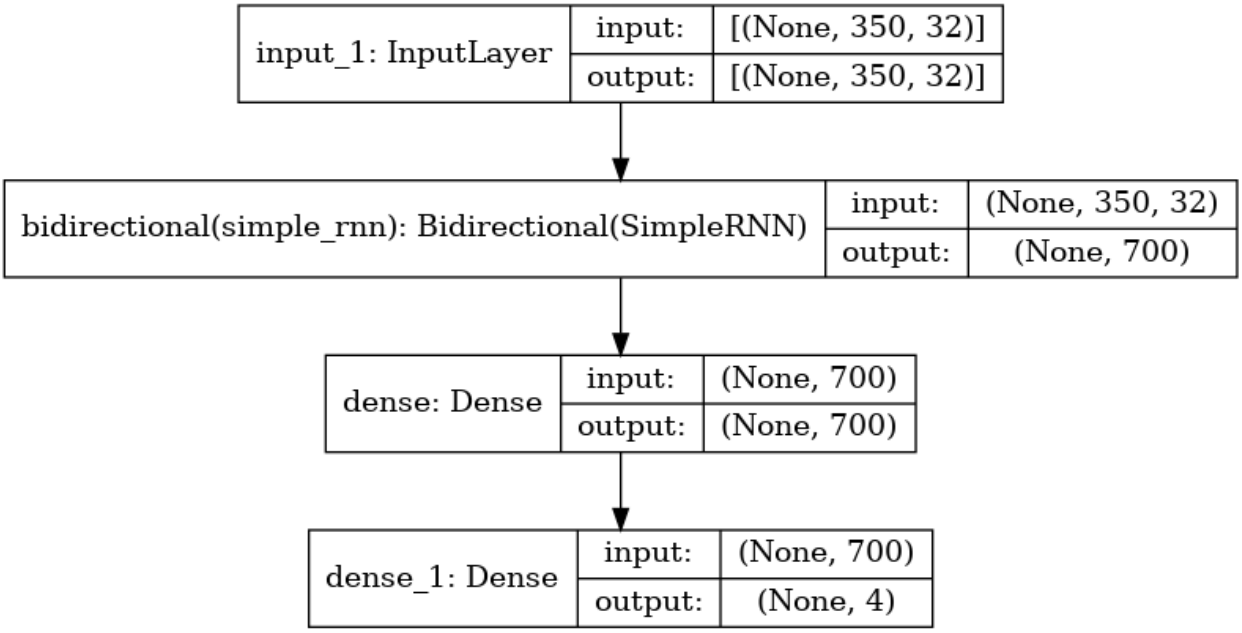
25. Schirrmeister R. T. Deep learning with convolutional neural networks for EEG decoding and visualization / Schirrmeister R. T., Springenberg J. T., Fiederer L. D. J., Glasstetter M., Eggensperger K., Tangermann M., Ball T. // Human brain mapping, 38(11), 2017. - С. 5391-5420.
26. EEG applications for sport and performance / Trevor Thompson, Tony Steffert, Tomas Ros, Joseph Leach, John Gruzelier // ISSN 1046-2023, 2008.
27. Real-Time EEG-Based Emotion Recognition and Its Applications / Liu, Y., Sourina, O., Nguyen, M.K // Springer, 2011.
28. Mohanchandra K. Using brain waves as new biometric feature for authenticating a computer user in real-time / Mohanchandra K., Lingaraju G., Kambli P., Krishnamurthy V. // International Journal of Biometrics and Bioinformatics (IJBB), 2013.
29. Hundia, R. Brain computer interface-controlling devices utilizing the alpha brain waves // International Journal of Scientific & Technology Research, (2015). - С. 281-285.
30. Глибовець М. М., Олецький О.В. Штучний інтелект. — Київ : «Києво-Могилянська академія», 2002. — 364 с. — ISBN 966518153X. (укр.)
31. Machine Learning / Mitchell, Tom // New York: McGraw Hill, 1997.
32. Introduction to Machine Learning / Alpaydin, Ethem // MIT Press, 2010.
33. Artificial Intelligence: A Modern Approach (Third ed.) / Russell, Stuart J. // Prentice Hall, 2010.
34. Neural Networks / Jordan, Michael I.; Bishop, Christopher M. // Computer Science Handbook, Second Edition (Section VII: Intelligent Systems), 2004.
35. Weak Supervision: The New Programming Paradigm for Machine Learning / Alex Ratner; Stephen Bach; Paroma Varma; Chris [Електронний ресурс], - 2019 - Режим доступу: http://ai.stanford.edu/blog/weak-supervision/.
36. Reinforcement learning and markov decision processes / van Otterlo, M.; Wiering, M. // Reinforcement Learning. Adaptation, Learning, and Optimization, 2012.
37. Deep Learning / Bengio, Yoshua; LeCun, Yann; Hinton, Geoffrey // Nature, 2015.

38. Top 10 algorithms in data mining / Wu, Xindong; Kumar, Vipin; Ross Quinlan, J.; Ghosh, Joydeep; Yang, Qiang; Motoda, Hiroshi; McLachlan, Geoffrey J.; Ng, Angus; Liu, Bing; Yu, Philip S.; Zhou, Zhi-Hua // Knowledge and Information Systems, 2008.
39. Sanford Weisberg Criticism and Influence Analysis in Regression / R. Dennis Cook // Sociological Methodology, Vol. 13, 1982.
40. Genetic algorithms and machine learning / Goldberg, David E.; Holland, John H. Machine Learning, 1988.
41. Artificial neural networks, back propagation, and the Kelley-Bryson gradient procedure / Dreyfus, Stuart E.// Journal of Guidance, Control, and Dynamics, 1990.
42. Application of the residue number system to reduce hardware costs of the convolutional neural network implementation / Valueva, M.V.; Nagornov, N.N.; Lyakhov, P.A.; Valuev, G.V.; Chervyakov, N.I. // Mathematics and Computers in Simulation, 2020
43. Receptive fields and functional architecture of monkey striate cortex / Hubel, D. H.; Wiesel, T. N. // The Journal of Physiology, 1968.
44. Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow / Géron, Aurélien // Sebastopol, CA: O'Reilly Media, 2019.
45. Convolutional Neural Networks (LeNet) DeepLearning 0.1 documentation [Електронний ресурс] - DeepLearning, 2013. - Режим доступу: https://www.scribd.com/document/333051443/Convolutional-Neural-Networks-LeNet-DeepLearning-0-1-Documentation
46. Guide to convolutional neural networks : a practical application to traffic-sign detection and classification / Habibi, Aghdam, Hamed // Heravi, Elnaz Jahani. Cham, Switzerland, 2017.
47. 6.5 Back-Propagation and Other Differentiation Algorithms / Goodfellow, Ian; Bengio, Yoshua; Courville, Aaron // Deep Learning. MIT Press, 2016.
48. CS231n Convolutional Neural Networks for Visual Recognition [Електронний ресурс]. - Stanford CS, 2022. - Режим доступу: https://cs231n.github.io/

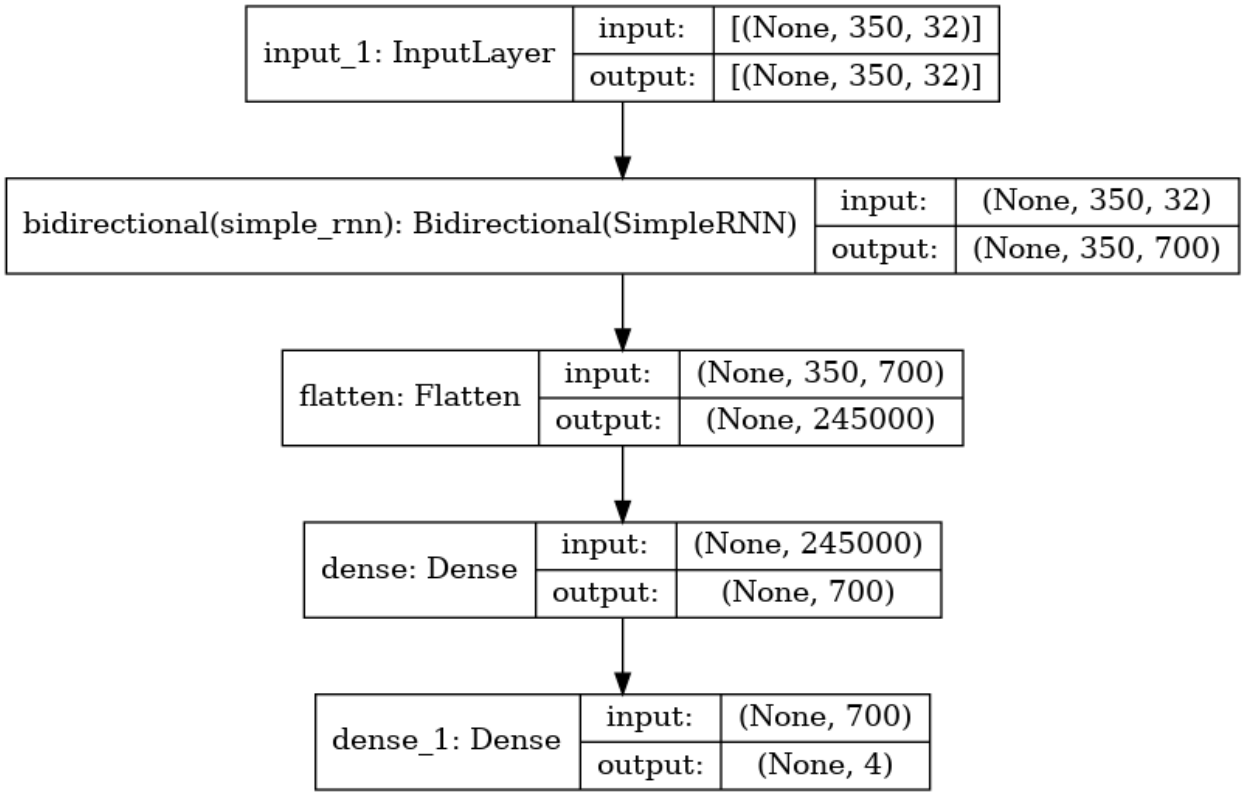
49. Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow / Géron, Aurélien // Sebastopol, CA: O'Reilly Media, 2019.
50. Why do deep convolutional networks generalize so poorly to small image transformations? / Azulay, Aharon; Weiss, Yair // Journal of Machine Learning Research, 2019.
51. Appropriate number and allocation of ReLUs in convolutional neural networks / Romanuke, Vadim // Research Bulletin of NTUU "Kyiv Polytechnic Institute", 2017.
52. Imagenet classification with deep convolutional neural networks / Krizhevsky, A.; Sutskever, I.; Hinton, G. E. // Advances in Neural Information Processing Systems, 2012.
53. Principal Component Analysis. Springer Series in Statistics / Jolliffe, I. T. // New York: Springer-Verlag, 2002.
54. Fully connected layer [Електронний ресурс]. - MATLAB, 2021. - Режим доступу: https://www.mathworks.com/help/deeplearning/ref/nnet.cnn.layer.fullyconnectedlayer.html;jsessionid=91f7abc5034e40e030ac3381d504
55. ImageNet classification with deep convolutional neural networks / Krizhevsky, Alex; Sutskever, Ilya; Hinton, Geoffrey E. // Communications of the ACM, 2017.
56. Gradient-based learning applied to document recognition / Lecun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. // Proceedings of the IEEE, 1998.
57. VGG16 – Convolutional Network for Classification and Detection [Електронний ресурс]. - Neurohive, 2018. - Режим доступу: https://neurohive.io/en/popular-networks/vgg16/.
58. A thorough review on the current advance of neural network structures / Dupond, Samuel // Annual Reviews in Control, 2019.
59. Bidirectional recurrent neural networks / Schuster, Mike, and Kuldip K. Paliwal // Signal Processing, IEEE Transactions on 45.11, 1997.
60. Long short-term memory / Sepp Hochreiter; Jürgen Schmidhuber // Neural Computation, 1997.

61. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches / Cho, Kyunghyun; van Merrienboer, Bart; Bahdanau, DZmitry; Bengio, Yoshua // arXiv:1409.1259, 2014.
62. Attention Is All You Need / Vaswani, Ashish; Shazeer, Noam; Parmar, Niki; Uszkoreit, Jakob; Jones, Llion; Gomez, Aidan N.; Kaiser, Lukasz; Polosukhin, Illia // arXiv:1706.03762, 2017.
63. Open Sourcing BERT: State-of-the-Art Pre-training for Natural Language Processing [Електронний ресурс]. - Google AI Blog, 2022. - Режим доступу: https://ai.googleblog.com/2018/11/open-sourcing-bert-state-of-art-pre.html.
64. Neural Machine Translation by Jointly Learning to Align and Translate / Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio // arXiv:1409.0473.
65. Big Data': Big gaps of knowledge in the field of Internet / Snijders, C.; Matzat, U.; Reips, U.-D. // International Journal of Internet Science, 2012.
66. Handwritten digit database [Електронний ресурс] - Gangaputra, Sachin - Режим доступу: https://www.cis.jhu.edu/~sachin/digit/digit.html
67. ImageNet Overview [Електронний ресурс] - ImageNet, 2016 - Режим доступу: https://image-net.org/about.php.
68. Multi-channel EEG recordings during 3,936 grasp and lift trials with varying weight and friction / Matthew D Luciw, Ewa Jarocka & Benoni B Edin // Nature, 2014.
69. The ten-twenty electrode system of the International Federation. / H.H. Jasper. (1958) // Electroencephalogr Clin Neurophysiol, 1958.
70. TensorFlow: Open source machine learning. [Електронний ресурс] - Google. 2015 – Режим доступу: https://www.tensorflow.org/
71. Keras [Електронний ресурс] - Keras, 2022 - Режим доступу: https://keras.io/
72. The Method of Steepest Descent for Non-linear Minimization Problems/ Curry, Haskell B. // Quart. Appl. Math, 1944
73. Digital design and computer architecture (2nd ed.) / Harris, David and Harris, Sarah // San Francisco, Calif.: Morgan Kaufmann, 2012

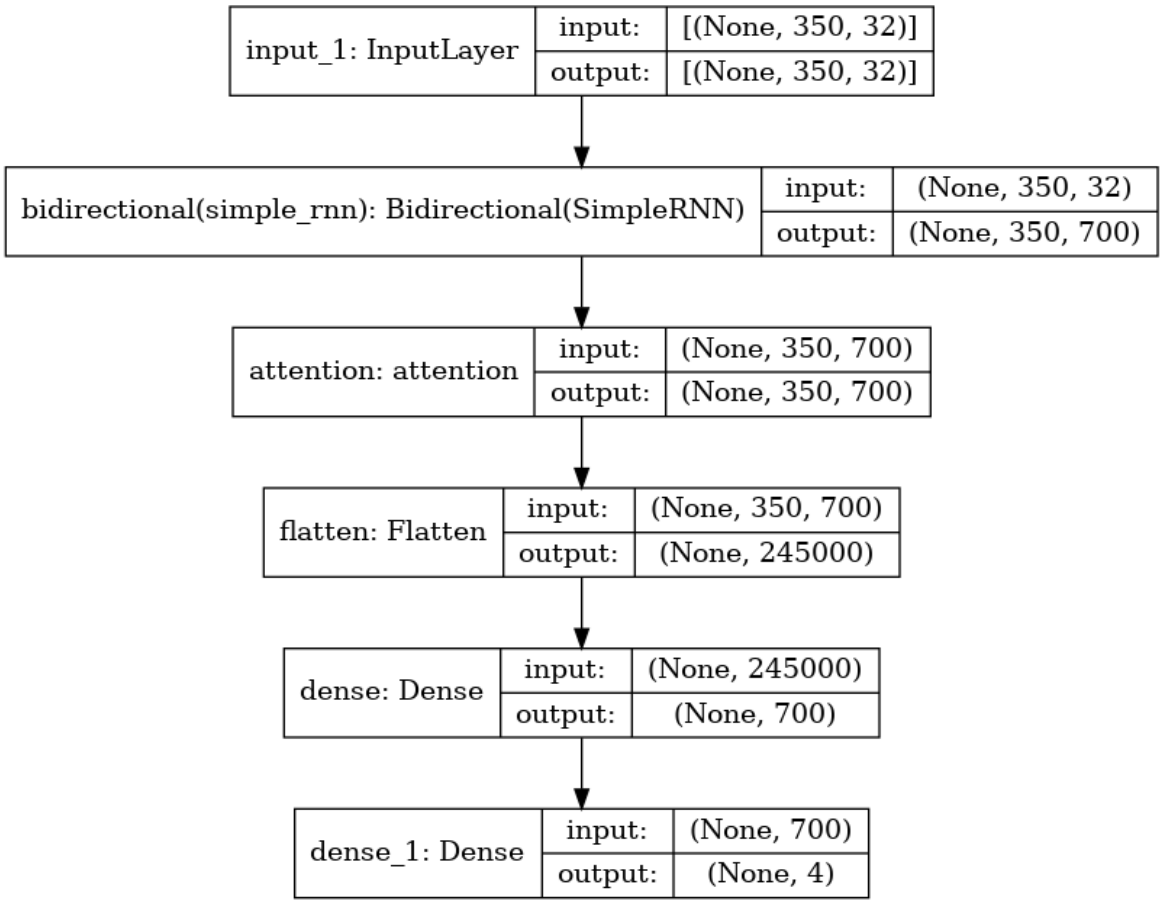
ДОДАТОК А. Мережа на основі найбільш розповсюджені архітектури RNN.



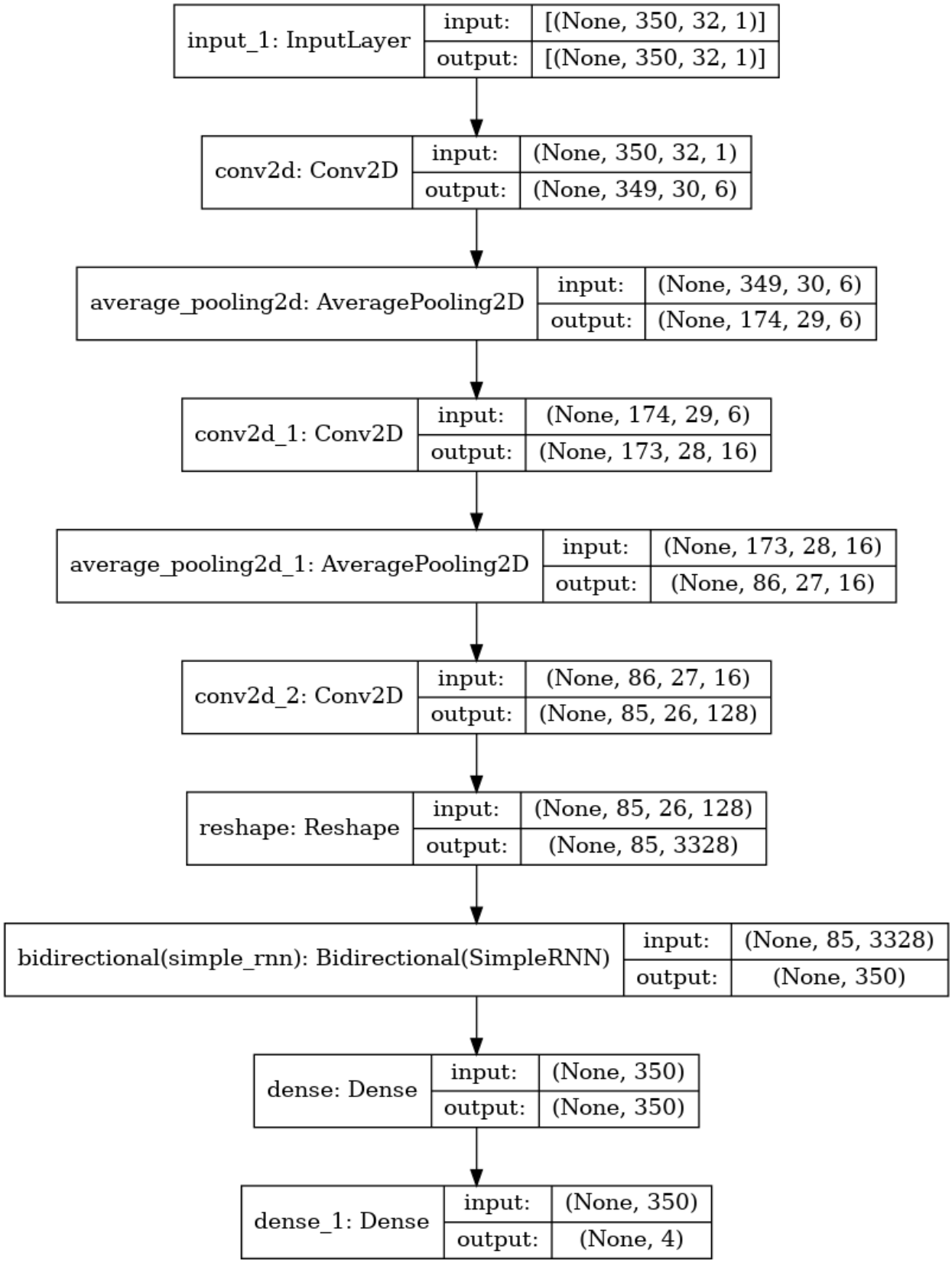
ДОДАТОК Б. Мережа на основі найбільш розповсюджені архітектури RNN з використанням прихованих станів.



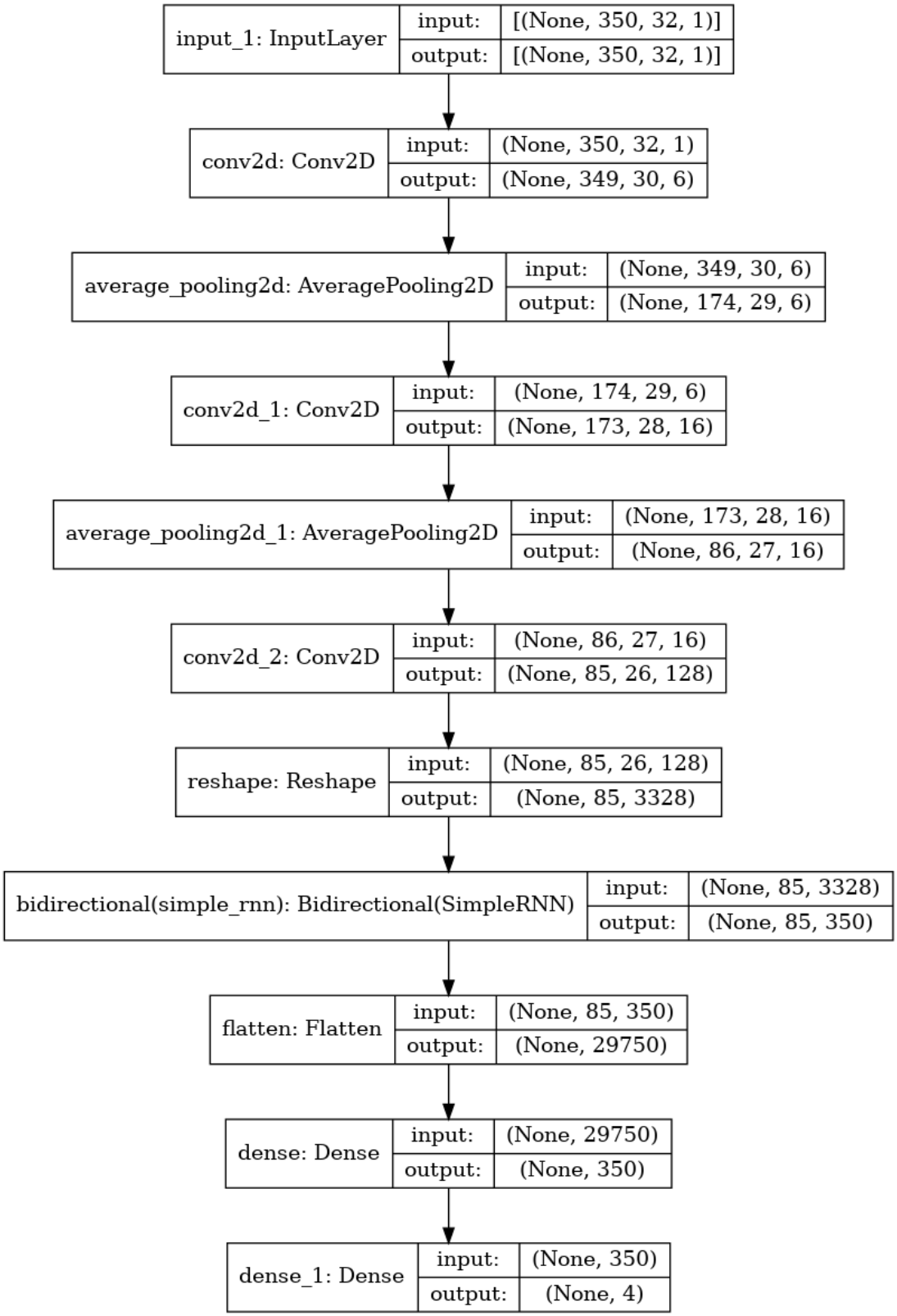
ДОДАТОК В. Мережа на основі найбільш розповсюджені архітектури RNN з використанням прихованих станів та механізмом уваги.



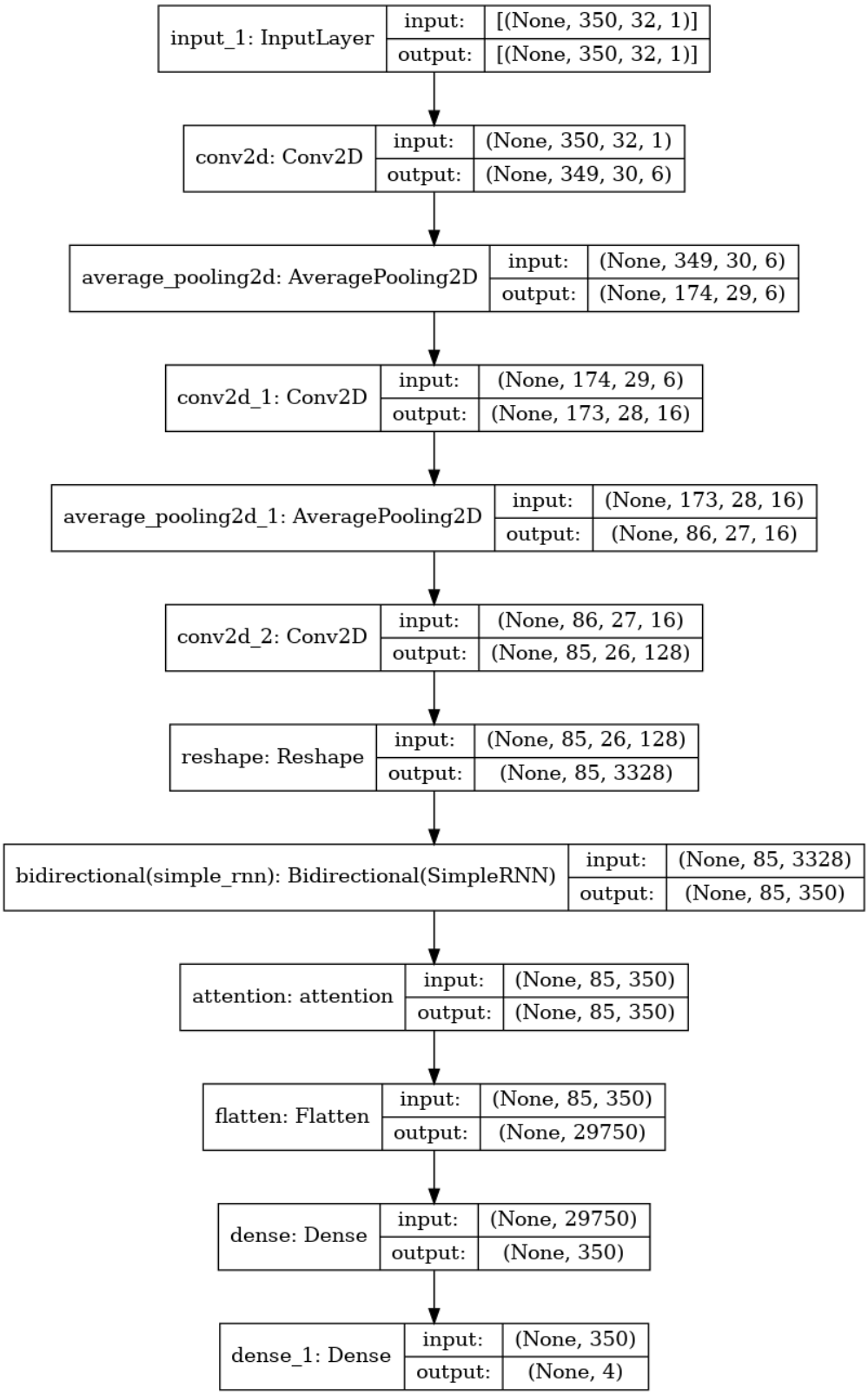
ДОДАТОК Г. Мережа на основі розповсюджених архітектур RNN з використанням згорткових шарів до рекурентних .



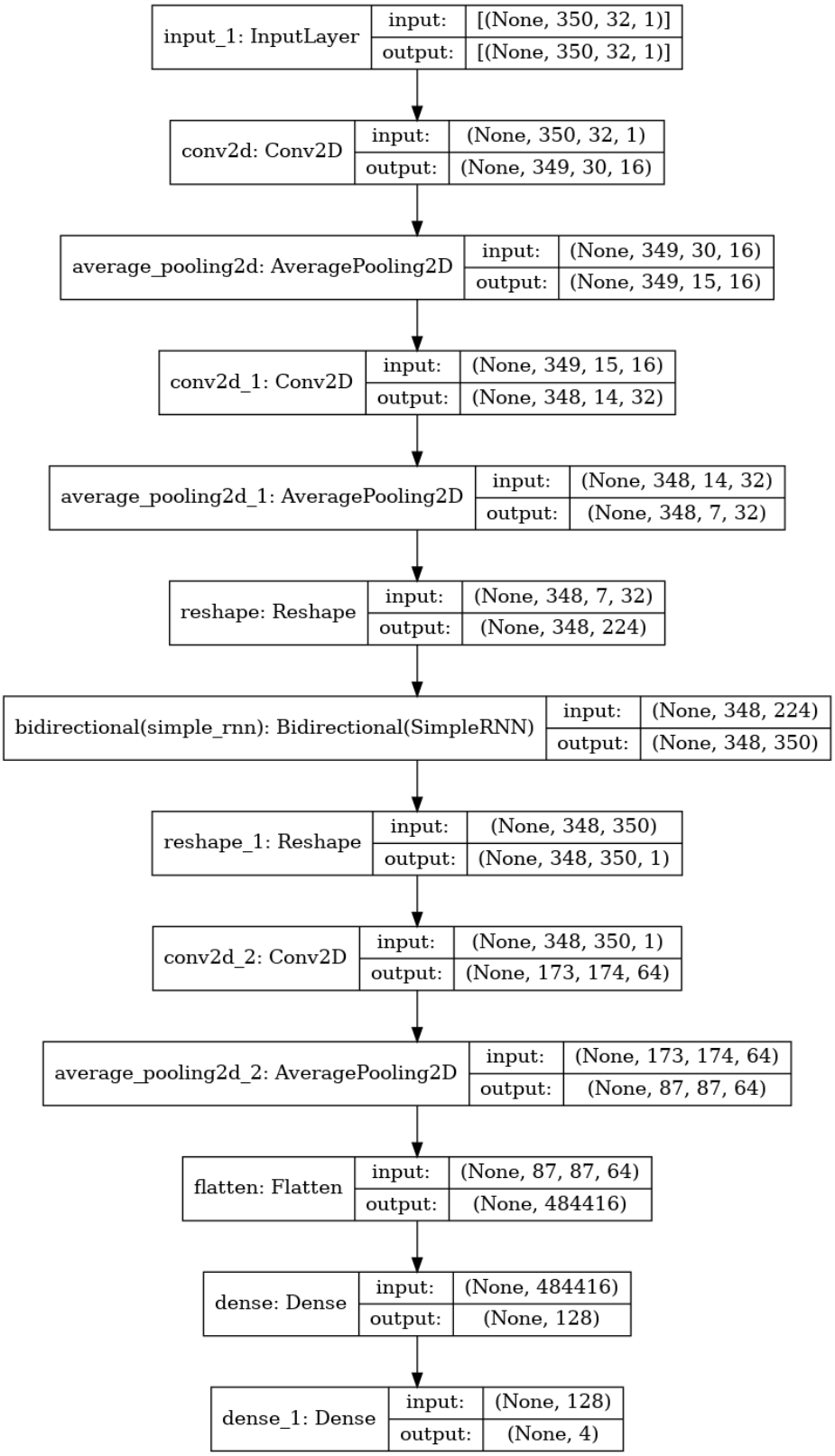
ДОДАТОК Д. Мережа на основі розповсюджені архітектур RNN з використанням згорткових шарів до рекурентних та використанням прихованих станів рекурентної мережі .



ДОДАТОК Е. Мережа на основі розповсюджених архітектур RNN з використанням згорткових шарів до рекурентних та використанням прихованих станів рекурентної мережі з використанням механізму уваги .



ДОДАТОК Є. Мережа на основі розповсюджени архітектур RNN з використанням згорткових шарів до рекурентних та використанням згорткових шарів після рекурентної мережі з використанням прихованих станів перед повнозв'язними шарами



ДОДАТОК Ж. Результати моделей та їх модифікацій

Назва моделі	**Точність AUC макро**	**Кількість параметрів**	**Час навчання (сек)**
CNN_LSTM_CNN_Dense	0,854	62568724	1885
CNN_GRU_CNN_Dense	0,846	62429774	1783
CNN_RNN_CNN_Dense	0,768	62148724	4835
CNN_LSTM_Hidden_Attention_Dense	0,845	15329051	946
CNN_GRU_Hidden_Attention_Dense	0,867	15329051	958
CNN_SimpleRNN_Hidden_Attention_Dense	0,865	11649851	1766
CNN_LSTM_Hidden_Dense	0,859	15328616	955
CNN_GRU_Hidden_Dense	0,873	14103266	922
CNN_SimpleRNN_Hidden_Dense	0,814	11649416	1803
CNN_LSTM_Dense	0,807	5038616	906
CNN_GRU_Dense	0,799	3823226	899
CNN_SimpleRNN_Dense	0,794	1359416	1666
LSTM_Hidden_Attention_Dense	0,859	172576954	2914
GRU_Hidden_Attention_Dense	0,879	172310954	4073
SimpleRNN_Hidden_Attention_Dense	0,854	171772654	4594
LSTM_Hidden_Dense	0,792	172575904	3999
GRU_Hidden_Dense	0,772	172309904	3650
SimpleRNN_Hidden_Dense	0,728	171772654	4907
LSTM_Dense	0,793	1565904	1275
GRU_Dense	0,796	1299904	1175
SimpleRNN_Dense	0,758	761604	3284
LENET	0,800	25815476	895
ALEX	0,851	320446212	4496
VGG16		692797764	

ДОДАТОК З. Код програми

```
import tensorflow.keras as keras

from tensorflow.keras.layers import Reshape, MaxPooling2D, AveragePooling2D
from tensorflow.keras import backend as K
from keras.layers import merge
#import Position_Embedding

nb_classes = 4

def slicee(x, h2, h3):
    return x[:,h2:h3,:]
class attention(keras.layers.Layer):
    def __init__(self, name="Attention", dtype=None, return_sequences=False, trainable=True):
        self.return_sequences = return_sequences
        super(attention,self).__init__()

    def build(self, input_shape):
        self.W=self.add_weight(name="att_weight", shape=(input_shape[-1],1), initializer="random_normal", trainable=True)
        self.b=self.add_weight(name="att_bias", shape=(input_shape[1],1),initializer="zeros", trainable=True)

        super(attention,self).build(input_shape)



    def call(self, x):
        e = K.tanh(K.dot(x,self.W)+self.b)
        a = K.softmax(e, axis=1)
        output = x*a
        if self.return_sequences:
            return output
        return K.sum(output, axis=1)
    def get_config(self):

        config = super().get_config().copy()
        config.update({
            'return_sequences': self.return_sequences,
        })
        return config

def create_model_SELF_ATTENTION_new_ALEX(time_steps, subsample):

    pool_kernel = (2,1)

    input_shape = (time_steps, 32, 1)
    input_layer = Input(shape=input_shape)

    layer = keras.layers.Conv2D(filters=96, kernel_size=(11,1), strides=(4,1), activation='relu')(input_layer)
    layer = keras.layers.BatchNormalization()(layer)
    layer = keras.layers.MaxPool2D(pool_size=(3,1), strides=(2,1))(layer)
    layer = keras.layers.Conv2D(filters=256, kernel_size=(5,5), strides=(1,1), activation='relu', padding="same")(layer)
    layer = keras.layers.BatchNormalization()(layer)
    layer = keras.layers.MaxPool2D(pool_size=(3,1), strides=(2,1))(layer)
    layer = keras.layers.Conv2D(filters=384, kernel_size=(3,1), strides=(1,1), activation='relu', padding="same")(layer)
    layer = keras.layers.BatchNormalization()(layer)
    layer = keras.layers.Conv2D(filters=384, kernel_size=(3,1), strides=(1,1), activation='relu', padding="same")(layer)
    layer = keras.layers.BatchNormalization()(layer)
    layer = keras.layers.Conv2D(filters=256, kernel_size=(3,1), strides=(1,1), activation='relu', padding="same")(layer)
    layer = keras.layers.BatchNormalization()(layer)
    layer = keras.layers.MaxPool2D(pool_size=(3,1), strides=(2,1))(layer)
    layer = keras.layers.Flatten()(layer)
    layer = keras.layers.Dense(4096, activation='relu')(layer)
    layer = keras.layers.Dropout(0.5)(layer)
    layer = keras.layers.Dense(4096, activation='relu')(layer)
    layer = keras.layers.Dropout(0.5)(layer)
    output_layer = keras.layers.Dense(4, activation='sigmoid')(layer)
```

```
    model = keras.models.Model(inputs=input_layer, outputs=output_layer)

    adam = Adam(learning_rate = 0.0001)

    model.compile(optimizer = adam, loss = "categorical_crossentropy", metrics = ["accuracy"])

    #model.summary()

    return model

def create_model_SELF_ATTENTION_new_LENET(time_steps, subsample):

    pool_kernel = (2,1)

    input_shape = (time_steps, 32, 1)
    input_layer = Input(shape=input_shape)

    layer = keras.layers.Conv2D(filters=6, kernel_size=(5,1), strides=(1,1), activation='tanh')(input_layer)
    layer = keras.layers.AveragePooling2D(pool_size = pool_kernel, strides=(2, 1), padding='valid')(layer)

    layer = keras.layers.Conv2D(16, kernel_size=(5, 1), strides=(1, 1), activation='tanh', padding='valid')(layer)

    layer = keras.layers.AveragePooling2D(pool_size = pool_kernel, strides=(2, 1), padding='valid')(layer)

    layer = keras.layers.Conv2D(120, kernel_size=(5, 1), strides=(1, 1), activation='tanh', padding='valid')(layer)

    layer = keras.layers.Flatten()(layer)

    layer = keras.layers.Dense(84, activation='tanh')(layer)

    output_layer = keras.layers.Dense(4, activation = "sigmoid")(layer)
    adam = Adam(learning_rate = 0.0001)

    model = keras.models.Model(inputs=input_layer, outputs=output_layer)

    model.compile(optimizer = adam, loss = "categorical_crossentropy", metrics = ["accuracy"])

    #model.summary()

    return model



def create_model_SELF_ATTENTION_new_VGG16(time_steps, subsample):
    pool_kernel = (2,1)

    input_shape = (time_steps, 32, 1)
    input_layer = Input(shape=input_shape)


    layer = keras.layers.Conv2D(filters=64,kernel_size=(3,1),padding="same", activation="relu")(input_layer)
    layer = keras.layers.Conv2D(filters=64,kernel_size=(3,1),padding="same", activation="relu")(layer)
    layer = keras.layers.MaxPooling2D(pool_size=(2,1),strides=(2,1))(layer)

    layer = keras.layers.Conv2D(filters=128, kernel_size=(3,1), padding="same", activation="relu")(layer)
    layer = keras.layers.Conv2D(filters=128, kernel_size=(3,1), padding="same", activation="relu")(layer)
    layer = keras.layers.MaxPooling2D(pool_size=(2,1),strides=(2,1))(layer)

    layer = keras.layers.Conv2D(filters=256, kernel_size=(3,1), padding="same", activation="relu")(layer)
    layer = keras.layers.Conv2D(filters=256, kernel_size=(3,1), padding="same", activation="relu")(layer)
    layer = keras.layers.Conv2D(filters=256, kernel_size=(3,1), padding="same", activation="relu")(layer)
    layer = keras.layers.MaxPooling2D(pool_size=(2,1),strides=(2,1))(layer)

    layer = keras.layers.Conv2D(filters=512, kernel_size=(3,1), padding="same", activation="relu")(layer)
    layer = keras.layers.Conv2D(filters=512, kernel_size=(3,1), padding="same", activation="relu")(layer)
```

```
    layer = keras.layers.Conv2D(filters=512, kernel_size=(3,1), padding="same", activation="relu")(layer)
    layer = keras.layers.MaxPooling2D(pool_size=(2,1),strides=(2,1))(layer)

    layer = keras.layers.Conv2D(filters=512, kernel_size=(3,1), padding="same", activation="relu")(layer)
    layer = keras.layers.Conv2D(filters=512, kernel_size=(3,1), padding="same", activation="relu")(layer)
    layer = keras.layers.Conv2D(filters=512, kernel_size=(3,1), padding="same", activation="relu")(layer)
    layer = keras.layers.MaxPooling2D(pool_size=(2,1),strides=(2,1))(layer)


    layer = keras.layers.Flatten()(layer)
    layer = keras.layers.Dense(units=4096,activation="relu")(layer)
    layer = keras.layers.Dense(units=4096,activation="relu")(layer)
    output_layer = keras.layers.Dense(units=4, activation="sigmoid")(layer)
    adam = Adam(learning_rate = 0.0001)

    model = keras.models.Model(inputs=input_layer, outputs=output_layer)

    model.compile(optimizer = adam, loss = "categorical_crossentropy", metrics = ["accuracy"])

    #model.summary()

    return model


def create_model_SELF_ATTENTION_new_RNN_SIMPLE(time_steps, subsample):

    pool_kernel = (2,1)

    units = int(time_steps/2)
    input_shape = (time_steps, 32)
    input_layer = Input(shape=input_shape)

    layer = keras.layers.Bidirectional(keras.layers.SimpleRNN(units*2))(input_layer)
    layer = keras.layers.Dense(units*4, activation='tanh')(layer)

    output_layer = keras.layers.Dense(4, activation = "sigmoid")(layer)
    adam = Adam(learning_rate = 0.0001)

    model = keras.models.Model(inputs=input_layer, outputs=output_layer)

    model.compile(optimizer = adam, loss = "categorical_crossentropy", metrics = ["accuracy"])

    #model.summary()

    return model


def create_model_SELF_ATTENTION_new_LSTM(time_steps, subsample):

    pool_kernel = (2,1)

    units = int(time_steps/2)
    input_shape = (time_steps, 32)
    input_layer = Input(shape=input_shape)

    layer = keras.layers.Bidirectional(keras.layers.LSTM(units*2))(input_layer)
    layer = keras.layers.Dense(units*2*2, activation='tanh')(layer)

    output_layer = keras.layers.Dense(4, activation = "sigmoid")(layer)
    adam = Adam(learning_rate = 0.0001)

    model = keras.models.Model(inputs=input_layer, outputs=output_layer)

    model.compile(optimizer = adam, loss = "categorical_crossentropy", metrics = ["accuracy"])

    #model.summary()
```

```
    return model


def create_model_SELF_ATTENTION_new_GRU(time_steps, subsample):

    pool_kernel = (2,1)

    units = int(time_steps/2)
    input_shape = (time_steps, 32)
    input_layer = Input(shape=input_shape)

    layer = keras.layers.Bidirectional(keras.layers.GRU(units*2))(input_layer)
    layer = keras.layers.Dense(units*2*2, activation='tanh')(layer)

    output_layer = keras.layers.Dense(4, activation = "sigmoid")(layer)
    adam = Adam(learning_rate = 0.0001)

    model = keras.models.Model(inputs=input_layer, outputs=output_layer)

    model.compile(optimizer = adam, loss = "categorical_crossentropy", metrics = ["accuracy"])

    #model.summary()

    return model

def create_model_SELF_ATTENTION_new_GRU_HIDDEN_STATES(time_steps, subsample):

    pool_kernel = (2,1)

    units = int(time_steps/2)
    input_shape = (time_steps, 32)
    input_layer = Input(shape=input_shape)

    layer = keras.layers.Bidirectional(keras.layers.GRU(units*2, return_sequences=True))(input_layer)
    layer = keras.layers.Flatten()(layer)
    layer = keras.layers.Dense(units*2*2, activation='tanh')(layer)

    output_layer = keras.layers.Dense(4, activation = "sigmoid")(layer)
    adam = Adam(learning_rate = 0.0001)

    model = keras.models.Model(inputs=input_layer, outputs=output_layer)

    model.compile(optimizer = adam, loss = "categorical_crossentropy", metrics = ["accuracy"])

    #model.summary()

    return model


def create_model_SELF_ATTENTION_new_LSTM_HIDDEN_STATES(time_steps, subsample):

    pool_kernel = (2,1)

    units = int(time_steps/2)
    input_shape = (time_steps, 32)
    input_layer = Input(shape=input_shape)

    layer = keras.layers.Bidirectional(keras.layers.LSTM(units*2, return_sequences=True))(input_layer)

    layer = keras.layers.Flatten()(layer)
    layer = keras.layers.Dense(units*2*2, activation='tanh')(layer)

    output_layer = keras.layers.Dense(4, activation = "sigmoid")(layer)
    adam = Adam(learning_rate = 0.0001)

    model = keras.models.Model(inputs=input_layer, outputs=output_layer)
```

```
    model.compile(optimizer = adam, loss = "categorical_crossentropy", metrics = ["accuracy"])

    #model.summary()

    return model

def create_model_SELF_ATTENTION_new_LSTM_HIDDEN_ATTENTION(time_steps, subsample):

    pool_kernel = (2,1)

    units = int(time_steps/2)
    input_shape = (time_steps, 32)
    input_layer = Input(shape=input_shape)

    layer = keras.layers.Bidirectional(keras.layers.LSTM(units*2, return_sequences=True))(input_layer)

    layer = attention(return_sequences=True)(layer)
    layer = keras.layers.Flatten()(layer)
    layer = keras.layers.Dense(units*2*2, activation='tanh')(layer)

    output_layer = keras.layers.Dense(4, activation = "sigmoid")(layer)
    adam = Adam(learning_rate = 0.0001)

    model = keras.models.Model(inputs=input_layer, outputs=output_layer)

    model.compile(optimizer = adam, loss = "categorical_crossentropy", metrics = ["accuracy"])

    #model.summary()

    return model

def create_model_SELF_ATTENTION_new_GRU_HIDDEN_ATTENTION(time_steps, subsample):

    pool_kernel = (2,1)

    units = int(time_steps/2)
    input_shape = (time_steps, 32)
    input_layer = Input(shape=input_shape)

    layer = keras.layers.Bidirectional(keras.layers.GRU(units*2, return_sequences=True))(input_layer)

    layer = attention(return_sequences=True)(layer)
    layer = keras.layers.Flatten()(layer)
    layer = keras.layers.Dense(units*2*2, activation='tanh')(layer)

    output_layer = keras.layers.Dense(4, activation = "sigmoid")(layer)
    adam = Adam(learning_rate = 0.0001)

    model = keras.models.Model(inputs=input_layer, outputs=output_layer)

    model.compile(optimizer = adam, loss = "categorical_crossentropy", metrics = ["accuracy"])

    #model.summary()

    return model


def create_model_SELF_ATTENTION_new_RNN_HIDDEN_ATTENTION(time_steps, subsample):

    pool_kernel = (2,1)

    units = int(time_steps/2)
    input_shape = (time_steps, 32)
    input_layer = Input(shape=input_shape)

    layer = keras.layers.Bidirectional(keras.layers.SimpleRNN(units*2, return_sequences=True))(input_layer)
```

```
    layer = attention(return_sequences=True)(layer)
    layer = keras.layers.Flatten()(layer)
    layer = keras.layers.Dense(units*2*2, activation='tanh')(layer)

    output_layer = keras.layers.Dense(4, activation = "sigmoid")(layer)
    adam = Adam(learning_rate = 0.0001)

    model = keras.models.Model(inputs=input_layer, outputs=output_layer)

    model.compile(optimizer = adam, loss = "categorical_crossentropy", metrics = ["accuracy"])

    #model.summary()

    return model

def create_model_SELF_ATTENTION_new_RNN_HIDDEN_STATES(time_steps, subsample):

    pool_kernel = (1,2)

    units = int(time_steps/2)
    input_shape = (time_steps, 32)
    input_layer = Input(shape=input_shape)

    layer = keras.layers.Bidirectional(keras.layers.SimpleRNN(units*2, return_sequences=True))(input_layer)
    layer = keras.layers.Flatten()(layer)
    layer = keras.layers.Dense(units*2*2, activation='tanh')(layer)

    output_layer = keras.layers.Dense(4, activation = "sigmoid")(layer)
    adam = Adam(learning_rate = 0.0001)

    model = keras.models.Model(inputs=input_layer, outputs=output_layer)

    model.compile(optimizer = adam, loss = "categorical_crossentropy", metrics = ["accuracy"])

    #model.summary()

    return model


def create_model_SELF_ATTENTION_new_CNN_PLUS_RNN(time_steps, subsample):

    pool_kernel = (2,2)

    units = int(time_steps/2)

    input_shape = (time_steps, 32, 1)
    input_layer = Input(shape=input_shape)


    layer = keras.layers.Conv2D(filters=6, kernel_size=(2,3), strides=(1,1), activation='tanh')(input_layer)
    layer = keras.layers.AveragePooling2D(pool_size = pool_kernel, strides=(2, 1), padding='valid')(layer)

    layer = keras.layers.Conv2D(16, kernel_size=(2, 2), strides=(1, 1), activation='tanh', padding='valid')(layer)

    layer = keras.layers.AveragePooling2D(pool_size = pool_kernel, strides=(2, 1), padding='valid')(layer)

    layer = keras.layers.Conv2D(128, kernel_size=(2, 2), strides=(1, 1), activation='tanh', padding='valid')(layer)

    CNN_out = Reshape((int(layer.shape[1]), int(layer.shape[2] * layer.shape[3])))(layer)
    layer = keras.layers.Bidirectional(keras.layers.SimpleRNN(units))(CNN_out)
    layer = keras.layers.Dense(units*2, activation='tanh')(layer)

    output_layer = keras.layers.Dense(4, activation = "sigmoid")(layer)
    adam = Adam(learning_rate = 0.0001)

    model = keras.models.Model(inputs=input_layer, outputs=output_layer)
```

```
    model.compile(optimizer = adam, loss = "categorical_crossentropy", metrics = ["accuracy"])

    #model.summary()

    return model

def create_model_SELF_ATTENTION_new_CNN_PLUS_LSTM(time_steps, subsample):

    pool_kernel = (2,2)

    units = int(time_steps/2)
    input_shape = (time_steps, 32, 1)
    input_layer = Input(shape=input_shape)


    layer = keras.layers.Conv2D(filters=6, kernel_size=(2,3), strides=(1,1), activation='tanh')(input_layer)
    layer = keras.layers.AveragePooling2D(pool_size = pool_kernel, strides=(2, 1), padding='valid')(layer)

    layer = keras.layers.Conv2D(16, kernel_size=(2, 2), strides=(1, 1), activation='tanh', padding='valid')(layer)

    layer = keras.layers.AveragePooling2D(pool_size = pool_kernel, strides=(2, 1), padding='valid')(layer)

    layer = keras.layers.Conv2D(128, kernel_size=(2, 2), strides=(1, 1), activation='tanh', padding='valid')(layer)


    CNN_out = Reshape((int(layer.shape[1]), int(layer.shape[2] * layer.shape[3])))(layer)
    layer = keras.layers.Bidirectional(keras.layers.LSTM(units))(CNN_out)
    layer = keras.layers.Dense(units*2, activation='tanh')(layer)

    output_layer = keras.layers.Dense(4, activation = "sigmoid")(layer)
    adam = Adam(learning_rate = 0.0001)

    model = keras.models.Model(inputs=input_layer, outputs=output_layer)

    model.compile(optimizer = adam, loss = "categorical_crossentropy", metrics = ["accuracy"])

    #model.summary()

    return model

def create_model_SELF_ATTENTION_new_CNN_PLUS_GRU(time_steps, subsample):
    pool_kernel = (2,2)

    units = int(time_steps/2)
    input_shape = (time_steps, 32, 1)
    input_layer = Input(shape=input_shape)


    layer = keras.layers.Conv2D(filters=6, kernel_size=(2,3), strides=(1,1), activation='tanh')(input_layer)
    layer = keras.layers.AveragePooling2D(pool_size = pool_kernel, strides=(2, 1), padding='valid')(layer)

    layer = keras.layers.Conv2D(16, kernel_size=(2, 2), strides=(1, 1), activation='tanh', padding='valid')(layer)

    layer = keras.layers.AveragePooling2D(pool_size = pool_kernel, strides=(2, 1), padding='valid')(layer)

    layer = keras.layers.Conv2D(128, kernel_size=(2, 2), strides=(1, 1), activation='tanh', padding='valid')(layer)


    CNN_out = Reshape((int(layer.shape[1]), int(layer.shape[2] * layer.shape[3])))(layer)
    layer = keras.layers.Bidirectional(keras.layers.GRU(units))(CNN_out)
    layer = keras.layers.Dense(units*2, activation='tanh')(layer)
    layer = keras.layers.Dense(32, activation='tanh')(layer)
    output_layer = keras.layers.Dense(4, activation = "sigmoid")(layer)
    adam = Adam(learning_rate = 0.0001)

    model = keras.models.Model(inputs=input_layer, outputs=output_layer)
```

```
    model.compile(optimizer = adam, loss = "categorical_crossentropy", metrics = ["accuracy"])

    #model.summary()

    return model


def create_model_SELF_ATTENTION_new_CNN_PLUS_LSTM_PLUS_HIDDEN_STATES(time_steps, subsample):
    pool_kernel = (2,2)

    input_shape = (time_steps, 32, 1)
    input_layer = Input(shape=input_shape)

    units = int(time_steps/2)

    layer = keras.layers.Conv2D(filters=6, kernel_size=(2,3), strides=(1,1), activation='tanh')(input_layer)
    layer = keras.layers.AveragePooling2D(pool_size = pool_kernel, strides=(2, 1), padding='valid')(layer)

    layer = keras.layers.Conv2D(16, kernel_size=(2, 2), strides=(1, 1), activation='tanh', padding='valid')(layer)

    layer = keras.layers.AveragePooling2D(pool_size = pool_kernel, strides=(2, 1), padding='valid')(layer)

    layer = keras.layers.Conv2D(128, kernel_size=(2, 2), strides=(1, 1), activation='tanh', padding='valid')(layer)


    CNN_out = Reshape((int(layer.shape[1]), int(layer.shape[2] * layer.shape[3])))(layer)
    layer = keras.layers.Bidirectional(keras.layers.LSTM(units, return_sequences=True))(CNN_out)
    layer = keras.layers.Flatten()(layer)
    layer = keras.layers.Dense(units*2, activation='tanh')(layer)

    output_layer = keras.layers.Dense(4, activation = "sigmoid")(layer)
    adam = Adam(learning_rate = 0.0001)

    model = keras.models.Model(inputs=input_layer, outputs=output_layer)

    model.compile(optimizer = adam, loss = "categorical_crossentropy", metrics = ["accuracy"])

    #model.summary()

    return model



def create_model_SELF_ATTENTION_new_CNN_PLUS_GRU_PLUS_HIDDEN_STATES(time_steps, subsample):
    pool_kernel = (2,2)

    input_shape = (time_steps, 32, 1)
    input_layer = Input(shape=input_shape)

    units = int(time_steps/2)

    layer = keras.layers.Conv2D(filters=6, kernel_size=(2,3), strides=(1,1), activation='tanh')(input_layer)
    layer = keras.layers.AveragePooling2D(pool_size = pool_kernel, strides=(2, 1), padding='valid')(layer)

    layer = keras.layers.Conv2D(16, kernel_size=(2, 2), strides=(1, 1), activation='tanh', padding='valid')(layer)

    layer = keras.layers.AveragePooling2D(pool_size = pool_kernel, strides=(2, 1), padding='valid')(layer)

    layer = keras.layers.Conv2D(128, kernel_size=(2, 2), strides=(1, 1), activation='tanh', padding='valid')(layer)


    CNN_out = Reshape((int(layer.shape[1]), int(layer.shape[2] * layer.shape[3])))(layer)
    layer = keras.layers.Bidirectional(keras.layers.GRU(units, return_sequences=True))(CNN_out)
    layer = keras.layers.Flatten()(layer)
    layer = keras.layers.Dense(units*2, activation='tanh')(layer)

    output_layer = keras.layers.Dense(4, activation = "sigmoid")(layer)
```

```
        adam = Adam(learning_rate = 0.0001)

        model = keras.models.Model(inputs=input_layer, outputs=output_layer)

        model.compile(optimizer = adam, loss = "categorical_crossentropy", metrics = ["accuracy"])

        #model.summary()

        return model

    def create_model_SELF_ATTENTION_new_CNN_PLUS_RNN_PLUS_HIDDEN_STATES_ATTENTION(time_steps,
subsample):
        pool_kernel = (2,2)

        units = int(time_steps/2)
        input_shape = (time_steps, 32, 1)
        input_layer = Input(shape=input_shape)


        layer = keras.layers.Conv2D(filters=6, kernel_size=(2,3), strides=(1,1), activation='tanh')(input_layer)
        layer = keras.layers.AveragePooling2D(pool_size = pool_kernel, strides=(2, 1), padding='valid')(layer)

        layer = keras.layers.Conv2D(16, kernel_size=(2, 2), strides=(1, 1), activation='tanh', padding='valid')(layer)

        layer = keras.layers.AveragePooling2D(pool_size = pool_kernel, strides=(2, 1), padding='valid')(layer)

        layer = keras.layers.Conv2D(128, kernel_size=(2, 2), strides=(1, 1), activation='tanh', padding='valid')(layer)


        CNN_out = Reshape((int(layer.shape[1]), int(layer.shape[2] * layer.shape[3])))(layer)
        layer = keras.layers.Bidirectional(keras.layers.SimpleRNN(units, return_sequences=True))(CNN_out)

        layer = attention(return_sequences=True)(layer)
        layer = keras.layers.Flatten()(layer)
        layer = keras.layers.Dense(units*2, activation='tanh')(layer)

        output_layer = keras.layers.Dense(4, activation = "sigmoid")(layer)
        adam = Adam(learning_rate = 0.0001)

        model = keras.models.Model(inputs=input_layer, outputs=output_layer)

        model.compile(optimizer = adam, loss = "categorical_crossentropy", metrics = ["accuracy"])

        #model.summary()

        return model


    def create_model_SELF_ATTENTION_new_CNN_PLUS_LSTM_PLUS_HIDDEN_STATES_ATTENTION(time_steps,
subsample):
        pool_kernel = (2,2)

        units = int(time_steps/2)
        input_shape = (time_steps, 32, 1)
        input_layer = Input(shape=input_shape)


        layer = keras.layers.Conv2D(filters=6, kernel_size=(2,3), strides=(1,1), activation='tanh')(input_layer)
        layer = keras.layers.AveragePooling2D(pool_size = pool_kernel, strides=(2, 1), padding='valid')(layer)

        layer = keras.layers.Conv2D(16, kernel_size=(2, 2), strides=(1, 1), activation='tanh', padding='valid')(layer)

        layer = keras.layers.AveragePooling2D(pool_size = pool_kernel, strides=(2, 1), padding='valid')(layer)

        layer = keras.layers.Conv2D(128, kernel_size=(2, 2), strides=(1, 1), activation='tanh', padding='valid')(layer)


        CNN_out = Reshape((int(layer.shape[1]), int(layer.shape[2] * layer.shape[3])))(layer)
```

```
        layer = keras.layers.Bidirectional(keras.layers.LSTM(units, return_sequences=True))(CNN_out)

        layer = attention(return_sequences=True)(layer)
        layer = keras.layers.Flatten()(layer)
        layer = keras.layers.Dense(units*2, activation='tanh')(layer)

        output_layer = keras.layers.Dense(4, activation = "sigmoid")(layer)
        adam = Adam(learning_rate = 0.0001)

        model = keras.models.Model(inputs=input_layer, outputs=output_layer)

        model.compile(optimizer = adam, loss = "categorical_crossentropy", metrics = ["accuracy"])

        #model.summary()

        return model

    def create_model_SELF_ATTENTION_new_CNN_PLUS_GRU_PLUS_HIDDEN_STATES_ATTENTION(time_steps, subsample):
        pool_kernel = (2,2)

        units = int(time_steps/2)
        input_shape = (time_steps, 32, 1)
        input_layer = Input(shape=input_shape)


        layer = keras.layers.Conv2D(filters=6, kernel_size=(2,3), strides=(1,1), activation='tanh')(input_layer)
        layer = keras.layers.AveragePooling2D(pool_size = pool_kernel, strides=(2, 1), padding='valid')(layer)

        layer = keras.layers.Conv2D(16, kernel_size=(2, 2), strides=(1, 1), activation='tanh', padding='valid')(layer)

        layer = keras.layers.AveragePooling2D(pool_size = pool_kernel, strides=(2, 1), padding='valid')(layer)

        layer = keras.layers.Conv2D(128, kernel_size=(2, 2), strides=(1, 1), activation='tanh', padding='valid')(layer)


        CNN_out = Reshape((int(layer.shape[1]), int(layer.shape[2] * layer.shape[3])))(layer)
        layer = keras.layers.Bidirectional(keras.layers.GRU(units, return_sequences=True))(CNN_out)

        layer = attention(return_sequences=True)(layer)
        layer = keras.layers.Flatten()(layer)
        layer = keras.layers.Dense(units*2, activation='tanh')(layer)

        output_layer = keras.layers.Dense(4, activation = "sigmoid")(layer)
        adam = Adam(learning_rate = 0.0001)

        model = keras.models.Model(inputs=input_layer, outputs=output_layer)

        model.compile(optimizer = adam, loss = "categorical_crossentropy", metrics = ["accuracy"])

        #model.summary()

        return model


    def create_model_SELF_ATTENTION_new_CNN_PLUS_RNN_PLUS_HIDDEN_STATES(time_steps, subsample):

        pool_kernel = (2,2)

        units = int(time_steps/2)
        input_shape = (time_steps, 32, 1)
        input_layer = Input(shape=input_shape)


        layer = keras.layers.Conv2D(filters=6, kernel_size=(2,3), strides=(1,1), activation='tanh')(input_layer)
        layer = keras.layers.AveragePooling2D(pool_size = pool_kernel, strides=(2, 1), padding='valid')(layer)
```

```
    layer = keras.layers.Conv2D(16, kernel_size=(2, 2), strides=(1, 1), activation='tanh', padding='valid')(layer)

    layer = keras.layers.AveragePooling2D(pool_size = pool_kernel, strides=(2, 1), padding='valid')(layer)

    layer = keras.layers.Conv2D(128, kernel_size=(2, 2), strides=(1, 1), activation='tanh', padding='valid')(layer)


    CNN_out = Reshape((int(layer.shape[1]), int(layer.shape[2] * layer.shape[3])))(layer)
    layer = keras.layers.Bidirectional(keras.layers.SimpleRNN(units, return_sequences=True))(CNN_out)
    layer = keras.layers.Flatten()(layer)
    layer = keras.layers.Dense(units*2, activation='tanh')(layer)

    output_layer = keras.layers.Dense(4, activation = "sigmoid")(layer)
    adam = Adam(learning_rate = 0.0001)

    model = keras.models.Model(inputs=input_layer, outputs=output_layer)

    model.compile(optimizer = adam, loss = "categorical_crossentropy", metrics = ["accuracy"])

    #model.summary()

    return model


def create_model_SELF_ATTENTION_new_CNN_PLUS_LSTM_HIDDEN_STATES(time_steps, subsample):

    pool_kernel = (2,2)

    units = int(time_steps/2)
    input_shape = (time_steps, 32, 1)
    input_layer = Input(shape=input_shape)


    layer = keras.layers.Conv2D(filters=6, kernel_size=(2,3), strides=(1,1), activation='tanh')(input_layer)
    layer = keras.layers.AveragePooling2D(pool_size = pool_kernel, strides=(2, 1), padding='valid')(layer)

    layer = keras.layers.Conv2D(16, kernel_size=(2, 2), strides=(1, 1), activation='tanh', padding='valid')(layer)

    layer = keras.layers.AveragePooling2D(pool_size = pool_kernel, strides=(2, 1), padding='valid')(layer)

    layer = keras.layers.Conv2D(128, kernel_size=(2, 2), strides=(1, 1), activation='tanh', padding='valid')(layer)


    CNN_out = Reshape((int(layer.shape[1]), int(layer.shape[2] * layer.shape[3])))(layer)
    layer = keras.layers.Bidirectional(keras.layers.LSTM(units, return_sequences=True))(CNN_out)
    layer = keras.layers.Flatten()(layer)
    layer = keras.layers.Dense(units*2, activation='tanh')(layer)

    output_layer = keras.layers.Dense(4, activation = "sigmoid")(layer)
    adam = Adam(learning_rate = 0.0001)

    model = keras.models.Model(inputs=input_layer, outputs=output_layer)

    model.compile(optimizer = adam, loss = "categorical_crossentropy", metrics = ["accuracy"])

    #model.summary()

    return model



def create_model_SELF_ATTENTION_new__(time_steps, subsample):

    units = int(time_steps)
    n_feature_maps = 100
    input_shape = (time_steps, 32, 1)
```

```
    input_layer = Input(shape=input_shape)
    print(input_layer.shape)


    cnn = keras.layers.TimeDistributed(keras.layers.Conv1D(filters=32, kernel_size=3, activation='relu'))
    conv1 = cnn(input_layer)
    print(conv1.shape)

    m = keras.layers.TimeDistributed(keras.layers.AveragePooling1D(pool_size=2))(conv1)
    conv2 = keras.layers.TimeDistributed(keras.layers.Conv1D(filters=64, kernel_size=3, activation='relu'))(m)
    m = keras.layers.TimeDistributed(keras.layers.AveragePooling1D(pool_size=2))(conv2)

    CNN_out = Reshape((int(m.shape[1]), int(m.shape[2] * m.shape[3])))(m)
    lstm = keras.layers.Bidirectional(keras.layers.LSTM(units))(CNN_out)

    layer = keras.layers.Dense(64, activation='tanh')(lstm)
    layer = keras.layers.Dense(32, activation='tanh')(layer)
    output_layer = keras.layers.Dense(4, activation = "sigmoid")(layer)

    model = keras.models.Model(inputs=input_layer, outputs=output_layer)
    adam = Adam(learning_rate = 0.0001)
    model.compile(optimizer = adam, loss = "categorical_crossentropy", metrics = ["accuracy"])

    return model



def create_model_SELF_ATTENTION_new_CNN_LSTM_CNN(time_steps, subsample):

    pool_kernel = (1,2)

    units = int(time_steps/2)
    input_shape = (time_steps, 32, 1)
    input_layer = Input(shape=input_shape)


    layer = keras.layers.Conv2D(filters=16, kernel_size=(2,3), strides=(1,1), activation='tanh')(input_layer)
    layer = keras.layers.AveragePooling2D(pool_size = pool_kernel, strides=(1,2), padding='valid')(layer)

    layer = keras.layers.Conv2D(32, kernel_size=(2, 2), strides=(1,1), activation='tanh', padding='valid')(layer)

    layer = keras.layers.AveragePooling2D(pool_size = pool_kernel, strides=(1,2), padding='valid')(layer)

    CNN_out = Reshape((int(layer.shape[1]), int(layer.shape[2] * layer.shape[3])))(layer)
    layer = keras.layers.Bidirectional(keras.layers.LSTM(units, return_sequences=True))(CNN_out)
    print(int(layer.shape[1]), int(layer.shape[2]))
    layer = Reshape((int(layer.shape[1]), int(layer.shape[2]), 1))(layer)

    layer = keras.layers.Conv2D(64, kernel_size=(3, 3), strides=(2,2), activation='tanh', padding='valid')(layer)

    layer = keras.layers.AveragePooling2D(pool_size = pool_kernel, strides=(2,2), padding='valid')(layer)

    layer = keras.layers.Flatten()(layer)

    layer = keras.layers.Dense(128, activation='tanh')(layer)
    output_layer = keras.layers.Dense(4, activation = "sigmoid")(layer)
    adam = Adam(learning_rate = 0.0001)

    model = keras.models.Model(inputs=input_layer, outputs=output_layer)

    model.compile(optimizer = adam, loss = "categorical_crossentropy", metrics = ["accuracy"])

    #model.summary()

    return model
```

```
def create_model_SELF_ATTENTION_new_CNN_GRU_CNN(time_steps, subsample):

    pool_kernel = (1,2)

    units = int(time_steps/2)
    input_shape = (time_steps, 32, 1)
    input_layer = Input(shape=input_shape)


    layer = keras.layers.Conv2D(filters=16, kernel_size=(2,3), strides=(1,1), activation='tanh')(input_layer)
    layer = keras.layers.AveragePooling2D(pool_size = pool_kernel, strides=(1,2), padding='valid')(layer)

    layer = keras.layers.Conv2D(32, kernel_size=(2, 2), strides=(1,1), activation='tanh', padding='valid')(layer)

    layer = keras.layers.AveragePooling2D(pool_size = pool_kernel, strides=(1,2), padding='valid')(layer)

    CNN_out = Reshape((int(layer.shape[1]), int(layer.shape[2] * layer.shape[3])))(layer)
    layer = keras.layers.Bidirectional(keras.layers.GRU(units, return_sequences=True))(CNN_out)
    print(int(layer.shape[1]), int(layer.shape[2]))
    layer = Reshape((int(layer.shape[1]), int(layer.shape[2]), 1))(layer)

    layer = keras.layers.Conv2D(64, kernel_size=(3, 3), strides=(2,2), activation='tanh', padding='valid')(layer)

    layer = keras.layers.AveragePooling2D(pool_size = pool_kernel, strides=(2,2), padding='valid')(layer)

    layer = keras.layers.Flatten()(layer)

    layer = keras.layers.Dense(128, activation='tanh')(layer)
    output_layer = keras.layers.Dense(4, activation = "sigmoid")(layer)
    adam = Adam(learning_rate = 0.0001)

    model = keras.models.Model(inputs=input_layer, outputs=output_layer)

    model.compile(optimizer = adam, loss = "categorical_crossentropy", metrics = ["accuracy"])

    #model.summary()

    return model


def create_model_SELF_ATTENTION_new_CNN_RNN_CNN(time_steps, subsample):

    pool_kernel = (1,2)

    units = int(time_steps/2)
    input_shape = (time_steps, 32, 1)
    input_layer = Input(shape=input_shape)


    layer = keras.layers.Conv2D(filters=16, kernel_size=(2,3), strides=(1,1), activation='tanh')(input_layer)
    layer = keras.layers.AveragePooling2D(pool_size = pool_kernel, strides=(1,2), padding='valid')(layer)

    layer = keras.layers.Conv2D(32, kernel_size=(2, 2), strides=(1,1), activation='tanh', padding='valid')(layer)

    layer = keras.layers.AveragePooling2D(pool_size = pool_kernel, strides=(1,2), padding='valid')(layer)

    CNN_out = Reshape((int(layer.shape[1]), int(layer.shape[2] * layer.shape[3])))(layer)
    layer = keras.layers.Bidirectional(keras.layers.SimpleRNN(units, return_sequences=True))(CNN_out)
    print(int(layer.shape[1]), int(layer.shape[2]))
    layer = Reshape((int(layer.shape[1]), int(layer.shape[2]), 1))(layer)

    layer = keras.layers.Conv2D(64, kernel_size=(3, 3), strides=(2,2), activation='tanh', padding='valid')(layer)

    layer = keras.layers.AveragePooling2D(pool_size = pool_kernel, strides=(2,2), padding='valid')(layer)

    layer = keras.layers.Flatten()(layer)
```

```
    layer = keras.layers.Dense(128, activation='tanh')(layer)
    output_layer = keras.layers.Dense(4, activation = "sigmoid")(layer)
    adam = Adam(learning_rate = 0.0001)

    model = keras.models.Model(inputs=input_layer, outputs=output_layer)

    model.compile(optimizer = adam, loss = "categorical_crossentropy", metrics = ["accuracy"])

    #model.summary()

    return model

all_y = []
global_index_train = [0] * 6
global_index_val = [0] * 6
global_index_test = [0] * 6
train_index = 0
def get_normal(minv, maxv):
    v = int(np.random.randint(minv, maxv))
    return v

def get_normal1(minv, maxv):
    val_min = minv
    val_max = maxv
    variation = (val_max - val_min)/2
    std_dev = variation/3
    mean = (val_max + val_min)/2
    v = int(np.random.normal(mean, std_dev))
    return v
### WITH NOISE !!!
#mu, sigma = 0, 0.01 ## for data augmentation by noise sd=0.05
def find_first_last_index(begin_index, x_d, y_d, time_steps, ds_index, offset):
    first_found = False
    last_found = False
    index_start = (0.5 - offset) * time_steps
    first = 0
    last = 0
    for index in range(begin_index, len(y_d)):
        y_index = int(index_start + index)
        if (y_index < 0 or y_index >= len(y_d)):
            continue

        yy = y_d[y_index][ds_index]
        if not first_found:
            if yy != 0:
                #print("ff ", index, y_d[index], yy)
                first = index
                first_found = True
                continue
        elif not last_found:
            if yy == 0:
                last = index - 1
                #print("lf ", last)
                last_found = True
                break

    if first == 0 or last == 0:
        print("Begin ", begin_index, " len ", len(y_d))
        print(first, last)
        assert(0)
    first = first - time_steps
    last = last
    return first, last, last


def generator(batch_size, offset, time_steps, subsample):
    global global_index_train, num_of_samples_train, train_index, train_test
    all_y = []
```

```
        indx = 0
        indx2 = 0
        j = 0
        while 1:
            x_time_data = np.zeros((batch_size, time_steps, 32))
            yy = []
            for i in range(batch_size):

                while j%6 == 1 or j%6 == 2:
                    j = j +1
                x_d = x_data_new_train_val[j%6]
                y_d = y_data_new_train_val[j%6]

                train_set = train_test[train_index][0]
                #print(j%6, train_index, global_index_train[j%6], train_set[global_index_train[j%6]%num_of_samples_train],
train_set[global_index_train[j%6]%num_of_samples_train]* 550)
                first_index, last_index, y_index = find_first_last_index(train_set[global_index_train[j%6]%num_of_samples_train]*
550, x_d, y_d, time_steps, j%6, offset)

                random_index = get_normal(first_index + 250, last_index - 240)

                #print("gen ", random_index)
                #print("gen ", first_index, ":",random_index+time_steps, "(",first_index + 185, ")")
                x_time_data[i] = x_d[random_index:random_index+time_steps]

                global_index_train[j%6] = global_index_train[j%6] + 1

                x_time_data[i] = x_d[random_index:random_index+time_steps]
                y_res = [0] * 4
                if j%6 == 0:
                    y_res[0] = 1
                elif j%6 == 3:
                    y_res[1] = 1
                elif j%6 == 4:
                    y_res[2] = 1
                elif j%6 == 5:
                    y_res[3] = 1
                yy.append(y_res)

                j = j + 1
            yy = np.asarray(yy)

            yield x_time_data.reshape((x_time_data.shape[0],x_time_data.shape[1], x_time_data.shape[2])), yy

    def val_generator(batch_size, offset, time_steps, subsample):
        global global_index_val, num_of_samples_val, train_test, train_index
        indx = 0
        indx2 = 0
        j = 0
        while 1:
            x_time_data = np.zeros((batch_size, time_steps, 32))
            yy = []
            for i in range(batch_size):

                while j%6 == 1 or j%6 == 2:
                    j = j +1
                x_d = x_data_new_train_val[j%6]
                y_d = y_data_new_train_val[j%6]

                test_set = train_test[train_index][1]
                #print(j%6, global_index_val[j%6], test_set[global_index_val[j%6]%num_of_samples_val],
test_set[global_index_val[j%6]%num_of_samples_val]* 550)
                first_index, last_index, y_index = find_first_last_index(test_set[global_index_val[j%6]%num_of_samples_val]* 550,
x_d, y_d, time_steps, j%6, offset)


                random_index = get_normal(first_index + 250, last_index - 240)

                #print("vgen ", random_index)
```

```
            x_time_data[i] = x_d[random_index:random_index+time_steps]

            global_index_val[j%6] = global_index_val[j%6] + 1

            x_time_data[i] = x_d[random_index:random_index+time_steps]
            y_res = [0] * 4
            if j%6 == 0:
                y_res[0] = 1
            elif j%6 == 3:
                y_res[1] = 1
            elif j%6 == 4:
                y_res[2] = 1
            elif j%6 == 5:
                y_res[3] = 1
            yy.append(y_res)

            j = j + 1
        yy = np.asarray(yy)


        yield x_time_data.reshape((x_time_data.shape[0],x_time_data.shape[1], x_time_data.shape[2])), yy

def test_generator(offset, time_steps, subsample):
    global global_index_test, num_of_samples_test
    indx = 0
    indx2 = 0
    j = 0

    batch_size = 1
    while 1:
        x_time_data = np.zeros((batch_size, time_steps, 32))
        yy = []
        for i in range(batch_size):


            while j%6 == 1 or j%6 == 2:
                j = j +1
            x_d = x_data_new_test[j%6]
            y_d = y_data_new_test[j%6]
            if global_index_test[j%6] > len(x_d) - 400:
                global_index_test[j%6] = 0

            first_index, last_index, y_index = find_first_last_index(global_index_test[j%6], x_d, y_d, time_steps, j%6, offset)

            random_index = get_normal(first_index + 250, last_index - 240)

            #print("tgen ", first_index)
            x_time_data[i] = x_d[random_index:random_index+time_steps]

            global_index_test[j%6] = last_index + 1

            x_time_data[i] = x_d[random_index:random_index+time_steps]
            y_res = [0] * 4
            if j%6 == 0:
                y_res[0] = 1
            elif j%6 == 3:
                y_res[1] = 1
            elif j%6 == 4:
                y_res[2] = 1
            elif j%6 == 5:
                y_res[3] = 1
            yy.append(y_res)

            j = j + 1
        yy = np.asarray(yy)


        yield x_time_data.reshape((x_time_data.shape[0],x_time_data.shape[1], x_time_data.shape[2])), yy


def print_all_y():
    stats = [ 0, 0, 0, 0, 0, 0]
```

```
print(len(all_y))
for y in all_y:
    for i in range(6):
        stats[i] = stats[i] + y[i]
print(stats)
```