Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

# КВАЛІФІКАЦІЙНА РОБОТА
# Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження застосування spiking нейронних мереж у задачах розпізнавання просторо-часових патернів в онлайн-навчанні
(тема)

Виконав:
студент 2 курсу, групи СШІм-21-1
Савенков Д.В.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системи штучного інтелекту
(повна назва спеціалізації)

Керівник д. т. н., проф. Семенець В.В.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ В.О. Філатов
(підпис) (прізвище, ініціали)

2023 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системи штучного інтелекту (СШІ)
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____» _____ 20 ___ р.

# ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Савенкову Денису Вадимовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження застосування spiking нейронних мереж у задачах розпізнавання просторо-часових патернів в онлайн-навчанні

затверджена наказом університету від 31 березня 20 23 р. № 306Ст

2. Термін подання студентом роботи до екзаменаційної комісії 15 травня 20 23 р.

3. Вихідні дані до роботи Науково-технічні публікації, дані Інтернет-джерел, відомих наукових проектів, документацій щодо розробки спайкових нейромереж у завданнях аналізу просторо-часових патернів з потоку даних в сценаріях онлайн-навчання, скрипт мовою програмування Python

4. Перелік питань, що потрібно опрацювати в роботі

1) Сучасні проблеми у онлайн-навчанні та як спайкові нейромережі допомагають їх вирішити

2) Спайкові нейронні мережі у завданнях вивчення просторо-часових патернів

3) Адаптація спайкових машин до дрейфу концепції

4) Використання спайкових мереж для ідентифікації дрейфу концепції

5) Експеримент з використанням спайкової машини у ідентифікації різкого дрейфу концепції

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Рисунок 1.1 – Процес навчання пакетами; Рисунок 1.2 – Процес онлайн-навчання; Рисунок 1.3 – Типи дрейфів за швидкістю; Рисунок 1.4 – Візуалізація роботи дрейф-детектора; Рисунок 1.5 – Анатомія біологічного нейрону; Рисунок 1.6 – Схема нейрону Ходжкіна-Хакслі; Рисунок 1.7 – Зміна частоти імпульсів від параметрів моделі; Рисунок 1.8 – Схема моделі LIF; Рисунок 2.1 – Архітектура LMU; Рисунок 2.2 – Зміна чутливості при змінах параметрів моделі; Рисунок 3.1 – Архітектура eSNN; Рисунок 3.2 – Схема роботи OeSNN; Рисунок 3.3 – Заповнення нейронного сховища без методик зменшення розмірності; Рисунок 3.4 – Заповнення нейронного сховища з методиками зменшення розмірності; Рисунок 4.1 – Процес детекції дрейфу у eSNN; Рисунок 5.1 – Процес навчання моделі SNN; Рисунок 5.2 – Процес навчання дерева Хофдінга; Рисунок 5.3 – Процес навчання моделі SNN із меншим τ.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1 )

| Найменування розділу | Консультант (посада, прізвище, ім'я, по батькові) | Позначка консультанта про виконання розділу | |
|---|---|---|---|
| | | підпис | дата |
| | | | |
| | | | |

**КАЛЕНДАРНИЙ ПЛАН**

| № | Назва етапів роботи | Терміни виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Аналіз предметної галузі | 04.04.2023-10.04.2023 | Виконано |
| 2 | Постановка завдання | 11.04.2023 | Виконано |
| 3 | Дослідження сучасного стану та проблем онлайн-навчання | 12.04.2023-17.04.2023 | Виконано |
| 4 | Дослідження літератури по спайковим нейромережам | 18.04.2023-20.04.2023 | Виконано |
| 5 | Постановка експерименту | 20.04.2023 | Виконано |
| 6 | Дослідження та вибір засобів реалізації експерименту | 21.04.2023 | Виконано |
| 7 | Розробка моделі та алгоритму навчання | 22.04.2023 | Виконано |
| 8 | Оформлення пояснювальної записки | 22.04.2023-29.04.2023 | Виконано |
| 9 | Оформлення презентаційних матеріалів | 30.04.2023-03.05.2023 | Виконано |
| 10 | Попередній захист | 08.05.2023 | Виконано |
| 11 | Захист перед екзаменаційною комісією | 15.05.2023 | |
| | | | |

Дата видачі завдання 3 квітня 2023 р.

Студент _____ _____
(підпис)

Керівник роботи _____ проф. Семенець В. В.
(підпис) (посада, прізвище, ініціали)

# РЕФЕРАТ

Пояснювальна записка: 75 с., 18 рис., 3 табл., 2 дод., 45 джерел.

АНАЛІЗ ПОТОКУ ДАНИХ, ДРЕЙФ-ДЕТЕКТОРИ, ДРЕЙФ КОНЦЕПЦІЇ, МАШИНЕ НАВЧАННЯ, ОНЛАЙН НАВЧАННЯ, СПАЙКОВА НЕЙРОНА МЕРЕЖА, ШТУЧНА НЕЙРОНА МЕРЕЖА.

Об’єктом дослідження є аналіз просторо-часових патернів у задачах онлайн-навчання за допомоги спайкових нейромереж.

Предметом дослідження є спайкові нейромережі, їх спроможності та ефективність у вивчення просторо-часових патернів для вирішення задач онлайн-навчання.

Мета кваліфікаційної роботи – дослідження сучасного стану онлайн-навчання, наявних проблем та як спайкові нейромережі допомагають їх вирішити; розробити алгоритм навчання спайкового дрейф-детектора, провести та проаналізувати емпіричний досвід.

Методи дослідження – аналіз та систематизація теоретичного матеріалу, проведення та аналіз експериментів.

Було систематизовано існуючі знання щодо використання спайкових мереж у різних задачах онлайн-навчання; було виявлено їх переваги перед та недоліки; був проведений аналіз вбудованих механізмів короткочасової пам’яті, як вони можуть бути використані у аналізі просторо-часових патернів у потоці даних. Було побудовано новаторський алгоритм навчання дрейф-детектору використовуючи тільки вхідні дані та отримано конкурентно-спроможні результати із мінімальною кількість False Positive помилок. Було зазначено наявні проблеми та перспективи.

Результати роботи можуть бути використано для подальших досліджень, практичного застосування та для ознайомлення із предметною галузі у закладах вищої освіти.

## ABSTRACT

Explanatory note: 75 p., 18 fig., 3 tabs, 2 ann., 45 sources.

ARTIFICIAL NEURAL NETWORK, CONCEPT DRIFT, DATA STREAM ANALYSIS, DRIFT DETECTORS, MACHINE LEARNING, ONLINE LEARNING, SPIKING NEURAL NETWORK.

The object of the study is the analysis of spatio-temporal patterns in online learning tasks using spiking neural networks.

The subject of the study is spiking neural networks, their capabilities and effectiveness in learning spatio-temporal patterns to solve online learning tasks.

The purpose of the work is to study the current state of the online learning, existing problems and how spiking neural networks can help solve them; to develop an algorithm for training a spiking drift detector, to conduct and analyse empirical experience.

Research methods: analysis and systematisation of theoretical material, conducting and analysing experiments.

The existing knowledge about the use of spiking networks in various online learning tasks was systematised; their advantages and disadvantages were identified; the built-in short-term memory mechanisms were analysed, and how they can be used in the analysis of spatio-temporal patterns in the data stream. An innovative algorithm for training a drift detector using only input data was built and competitive results were obtained with a minimum number of False Positive errors. The existing problems and prospects were noted.

The results of the work can be used for further research, practical application and for familiarisation with the subject area in higher education institutions.

**ЗМІСТ**

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ ТА ТЕРМІНІВ

AI – Artificial Intelligence – штучний інтелект;

ANN – Artificial Neural Network – штучна нейронна мережа;

BL – Batch Learning – навчання пакетами;

eSNN – Evolving Spiking Neural Network – еволюціонуюча спайкова нейрона нейронна мережа;

eSNN-DD – Evolving Spiking Neural Network for Drift Detection – еволюціонуюча спайкова нейронна мережа для детекції дрейфу;

GRU – Gradient Recurrent Unit – градієнтна періодична одиниця;

LIF – Leaky Integrate-and-Fire – дірявий нейрон механізму інтеграції та стрільби;

LMU – Legendre Memory Unit – одиниця пам’яті Лежандра

ML – Machine Learning – машинне навчання;

OeSNN – Online Evolving Spiking Neural Network – онлайн еволюціонуюча спайкова нейронна мережа;

OL – Online Learning – онлайн-навчання;

PLIF – Parametric Leaky Integrate-and-Fire – параметричний дірявий нейрон механізму інтеграції та стрільби;

RNN – Recurrent Neural Network – імпульсна рекурентна нейрона мережа;

RSNN – Recurrent Spiking Neural Network – рекурентна спайкова нейронна мережа

SNN – Spiking Neural Network – спайкова (імпульсна) нейронна мережа.

# ВСТУП

Більшість сучасних систем інтелектуального аналізу даних будуються на основі статичних навчальних середовищ, де вибірка даних є заздалегідь відома та повністю охоплює усі можливі значення. У таких системах, зазвичай, модель машинного навчання (Machine Learning, ML) проектується та навчається одноразово. Але велика частка прикладних завдань у реальному світі не мають таких умов: часто вибірки є обмеженими й потребують постійного оновлення, навчальні середовища мають динамічну природу й історичні тренди не відповідають сучасним. Інтелектуальні моделі у таких ситуаціях мають тенденції втрачати ефективність та потребують періодичного перенавчання або перемодулювання, що у свою чергу потребує додаткових затрат на побудову систем моніторингу та підтримки від спеціалістів.

Саме тому останнього часу дослідники та інженери в області штучного інтелекту (Artificial Intelligence, AI) почали звертати більше уваги до парадигми онлайн-навчання (Online-Learning, OL), зокрема у завданнях аналізу потоку даних. Порівняно із «традиційним» навчанням пакетами (Batch Learning, BL), алгоритми OL оновлюють модель кожного кроку при отриманні нових даних. Також, при такій парадигмі моделі будують адаптивними та спроможними до роботи в режимі реального часу.

OL має свої недоліки та існуючі проблеми, як катастрофічне забування інформації, знаходження компромісу між використанням наявних та створенням нових знань, необхідність будувати масштабовані системи та їх продукціоналізація, тощо. Але найкритичніше є те, що більшість моделей ML, особливо штучні нейромережі (Artificial Neural Network, ANN), неспроможні до швидкої та адекватної адаптації до дрейфу концепцій [1]. Також, це додатково ускладнюється необхідністю одночасно враховувати просторово-часову природу певних потоків даних, та робити моделі енергоефективними та швидкими для роботи в режимі реального часу. Це

ускладнюється тим, що звичайні ANN не вміють працювати із часовим виміром, що потребує використання рекурентних нейромереж (Recurrent Neural Network, RNN), які у свою чергу потребують великих обчислювальних ресурсів й, відповідно, погано інтегруються у системи реального часу.

Через зазначені недоліки ANN, додаткову увагу почали приділяти спайковим (імпульсним) нейромережам (Spiking Neural Network, SNN) [2]. Ці мережі вважаються третім поколінням й вони відрізняються тим, що обмін інформацією між нейронами проходить короткими імпульсами однакової амплітуди, працюючи при цьому із аналоговими сигналами та маючи накопичувальну природу, що несе в собі врахування часового вимірювання. З точки зору фізіології, цей підхід є найбільш реалістичною моделлю біологічного нейрону. Також, ця різниця у обчисленні є більш енергоефективною [3], дає можливість аналізувати просторово-часові патерни даних обчислювано-нескладними моделями [4] та можливість працювати у режимі реального часу, що робить ці моделі привабливими у завданнях онлайн-навчання [5].

Мета цієї кваліфікаційної роботи полягає у дослідженні природи та особливостей застосування SNN у завданнях онлайн-навчання та як вони вирішують перелічені раніше основні проблеми побудови інтелектуальних систем для аналізу потоку даних просторово-часової природи в умовах динамічних навчальних середовищ.

# 1 СУЧАСНІ ПРОБЛЕМИ У ОНЛАЙН НАВЧАННІ ТА ЯК СПАЙКОВІ НЕЙРОМЕРЕЖІ ДОПОМАГАЮТЬ ЇХ ВИРІШИТИ

## 1.1 Сучасний стан онлайн-навчання

### 1.1.1 Різниці між онлайн-навчанням та навчанням пакетами

Як було зазначено у вступі даної роботи, OL є принциповою альтернативою BL у задачах інтелектуального аналізу потоку даних. Парадигма BL застосовується у ситуаціях повної відомості навчальної вибірки або коли потоком даних приходять підвибірки великого розміру. У таких умовах моделі ML часто навчаються одноразово або необхідність у їх перенавчанні стає відносно рідко й може бути частково автоматизовано. Також, це дозволяє будувати моделі великої складності, витрачати більше часу і ресурсів на їх тренування, надає можливості додаткової обробки навчальної вибірки із створенням штучних даних, та навчати моделі кількома епохами. Це дозволяє розробникам систем AI будувати потужні та ефективні моделі ML.

У завданнях навчання з потоку даних, щоб модель дізналась про нові дані, її потрібно навчити повністю з нуля, з усіма даними. Після цього, стару модель видаляють, а сам процес, навіть з автоматизацією, може займати кілька годин або навіть днів. Процес навчання часто потребує великих обчислювальних ресурсів та пам’яті.

Загально, процес BL у завданнях навчання з потоку даних зображено на рисунку 1.1:

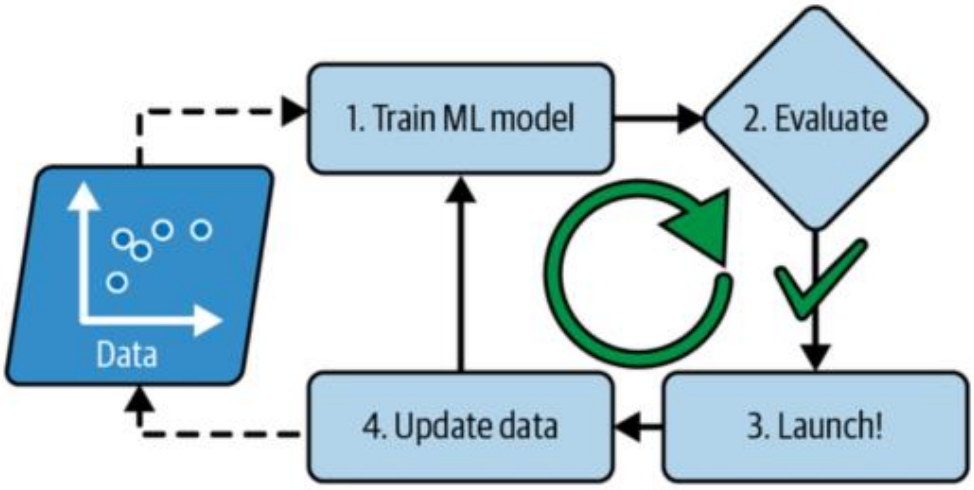


Рисунок 1.1 – Процес навчання пакетами

Але певні сценарії не дозволяють адекватно використати BL для вирішення поставленої задачі: навчальна вибірка може бути недоступною, системи реального часу (як онлайн-реклама, аналіз фінансових транзакцій, системи персоналізації або рекомендаційні системи) не можуть довго чекати оновлення моделей, старі патерни в даних можуть не відповідати новим тенденціям.

Саме у таких випадках використовують онлайн-навчання. На відміну від BL, алгоритми на основі OL будуються із очікуванням подачі лише одного зразку даних у кожен момент часу й миттєвим інкрементальним оновленням моделі ML. Тобто, алгоритми OL будують ітеративно. Кожен крок є швидким і обчислювано дешевим, системи мають можливість добувати знання з нових даних, в міру їх надходження.

Ця парадигма чудово підходить для систем ML, які працюють із безперервним потоком даних (тобто у режимі реального часу), де статистичний розподіл вхідних даних є динамічним та нові знання мають більшу вагу ніж старі. Моделі оновлюються швидко без необхідності повністю перенавчатись на повній вибірці даних; краще аналізують просторово-часові патерни, що характерні для потоку даних [6]. Також

системи потребують значно менше пам’яті та обчислювальних ресурсів. Наведена у рисунку 1.2 діаграма зображує процес OL:

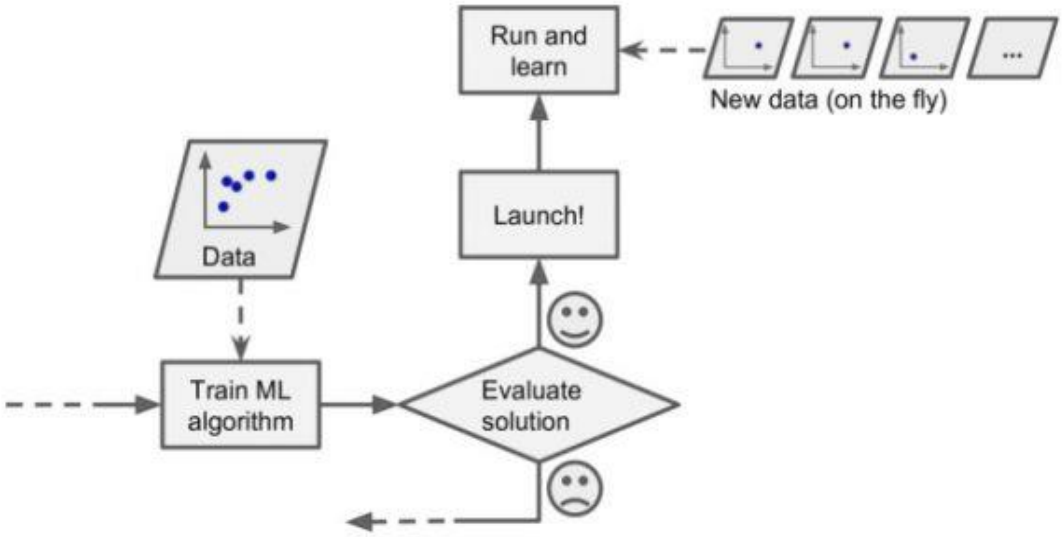


Рисунок 1.2 – Процес онлайн навчання

Окрім роботи із потоком даних, алгоритми OL також знаходять своє застосування у завданнях із великим набором даних, що не вміщуються у оперативну пам’ять комп'ютера. Ця проблема ще відома як позаядерне навчання (out-of-core learning) [7]. В таких випадках модель завантажує тільки частину даних, виконує крок навчання і повторює цей процес.

### 1.1.2 Існуючі проблеми та виклики онлайн-навчання

Хоча парадигма OL вирішує частку наявних проблем BL, вона має ряд власних викликів. По-перше, OL накладає інші обчислювальні обмеження на навчальне середовище, порівняно із більш традиційним BL. До цих обмежень входять:

– моделі повинні вміти обробляти зразки послідовно, знаходячи баланс між новими та старими знаннями;

– зразки даних повинні оброблятись стабільно швидко, час обробки не повинен перевищувати інтервал надходження наступного зразку;

– модель та алгоритм не повинні перевищувати заздалегідь виділений і обмежений обсяг пам’яті та інші обчислювальні ресурси;

– створена модель повинна бути, як мінімум, еквівалентна моделі навченою парадигмою BL.

Ці обмеження викликають ряд інших проблем. Багато сучасних моделей ML, особливо ANN [1], погано працюють в таких умовах. Наприклад, хоча RNN вміють аналізувати послідовності даних та знаходити просторово-часові закономірності, вони, при цьому, використовують багато обчислювальних ресурсів. Деякі моделі занадто покладаються на методи BL. Мало моделей вміють швидко перетренуватись до нових трендів, потребують великих об’ємів даних й потребують необхідності епохального навчання. І серед спроможних до OL моделей існують власні проблеми, як необхідність обережної оптимізації гіпер-параметрів: занадто висока швидкість навчання (learning rate) має тенденцію забувати вивчені знання, коли занадто низька швидкість буде заважати отримання нових, через що алгоритм наближається до звичайного BL. До проблем аналітичних «пайплайнів» ще відносять його паралелізацію, як ефективно провести методики обробки даних (наприклад, feature selection та feature transformation) або як опрацювати дисбаланс класів [8].

Також якість нових вхідних даних, та адаптації алгоритму до них, дуже впливає на якість інтелектуальної системи – користувач миттєво побачить наслідки поганого алгоритму при неспроможності підлаштуватись до нових тенденцій. Зокрема, це викликає необхідність створення додаткових систем стеження за продуктивністю ML, щоб, наприклад, ідентифікувати дрейф концепції [9], яку окремо буде розглянуто у пункті 1.1.3.

### 1.1.3 Дрейф концепції

OL в умовах наявності дрейфу концепцій є дуже актуальною темою протягом останніх років [10]. Дрейф концепції – загальний термін, що описує явище у OL та аналізі потоків, коли статистичні властивості даних кардинально непередбачувано змінюються, що призводить до погіршення якості наявної моделі ML. Як було зазначено у пункті 1.1.2, проблема ідентифікації дрейфу та адаптації систем до нього є однією з найважливіших та першорядних.

Дрейф може з’явитись як в області ознак або в області класів. Вони можуть виражатись у з’явлені нових ознак або класів, їх зникненням або іншою зміною їх значень.

Ці явище можна класифікувати за різними характеристиками. Якщо дивитись на те, що змінюється, виділяють [11]:

– справжній або реальний дрейф – характеризується зміною апостеріорних ймовірностей класів незалежно від розподілу ознак;

– віртуальний дрейф – характеризується зміною розподілу ознак без впливу на апостеріорні ймовірності класів.

Або, якщо характеризувати за швидкістю та мірою складності (серйозності) [12], виділяють:

– раптовий дрейф – зміна між «контекстами» відбувається неочікувано й миттєво;

– поступовий дрейф – зміна є плавною;

– інкрементальний дрейф – схожий із поступовим, з відмінністю у наявності проміжних значень;

– рекурентний дрейф – зміна контекстів повторюються з певним інтервалом;

– випадковий дрейф – новий контекст з’являється тимчасово в якості аномалії, зазвичай не має впливу на майбутні дані й тому адаптація алгоритму не є потрібною.

Візуальну інтерпретацію цих дрейфів можна побачити на рисунку 1.3 [5]:

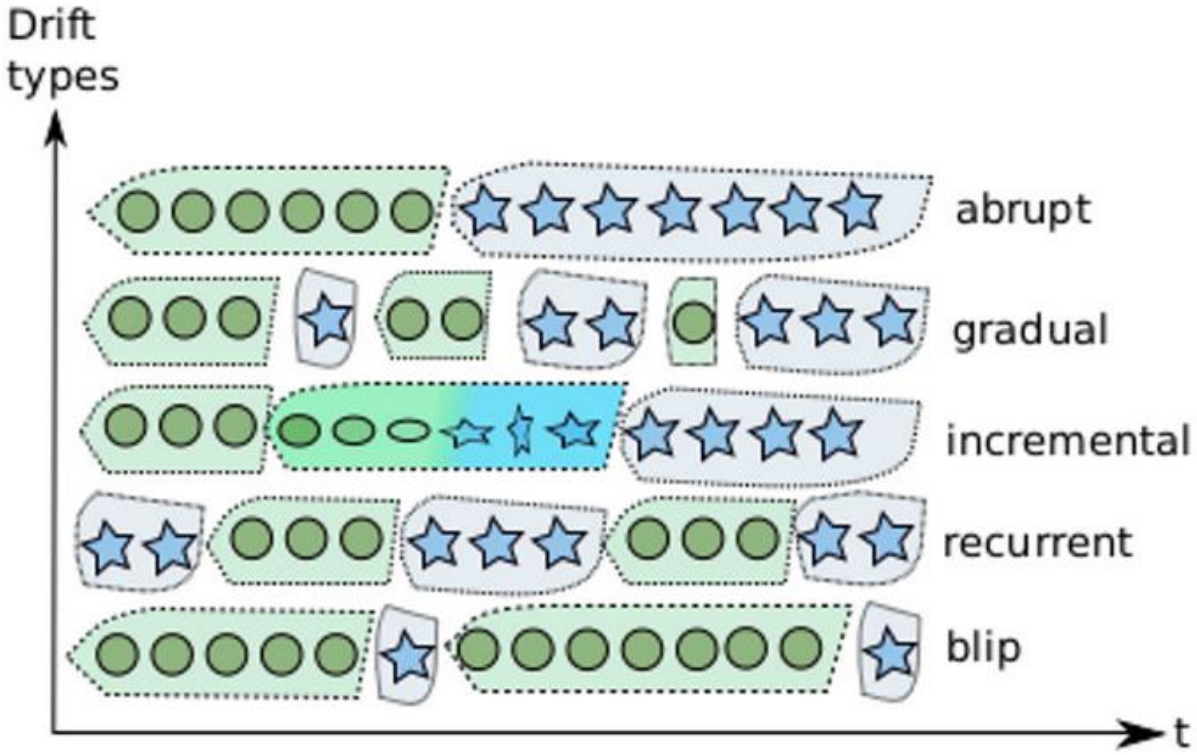


Рисунок 1.3 – Типи дрейфів за швидкістю

Алгоритми OL можуть справлятись із дрейфом концепцій двома різними стратегіями: активною стратегією, що вимагає спочатку ідентифікації дрейфу й після цього оновлюється, та пасивною стратегією, де алгоритм оновлюється безперервно при кожному надходженні нових даних. Ці стратегії також ще називають інформованими та сліпими відповідно. Пасивний підхід більш ефективний при поступових, інкрементальних та рекурентних дрейфах. Також вони краще підходять для BL. Активні підходи краще працюють із різким дрейфом й у завданнях OL.

Але незважаючи на підхід, важливим компонентом є так звані дрейф-детектори – методи та моделі, що можуть ідентифікувати зміни у розподілі вхідних даних опираючись або на інформацію основної моделі або

аналізуючи самі дані, зокрема їх просторово-часові патерни. Останнє виявляється доволі складною задачею, враховуючи інші виклики, описані у пункті 1.1.2.

Дрейф-детектори, зазвичай, повертають бінарний сигнал про наявність дрейфу, але також можуть й повертати попереджуючий сигнал (див. рисунок 1.4 [5]). Важливою метрикою ефективності детекторів є часова затримка виявлення дрейфу (особливо різкого) – чим швидше дрейф буде ідентифіковано, тим швидше з'явиться нова ефективна модель і, відповідно, тим менше проблем з'явиться у користувачів інтелектуальної системи. Звичайні ML-метрики, як точність або F1-Score, також є важливими.

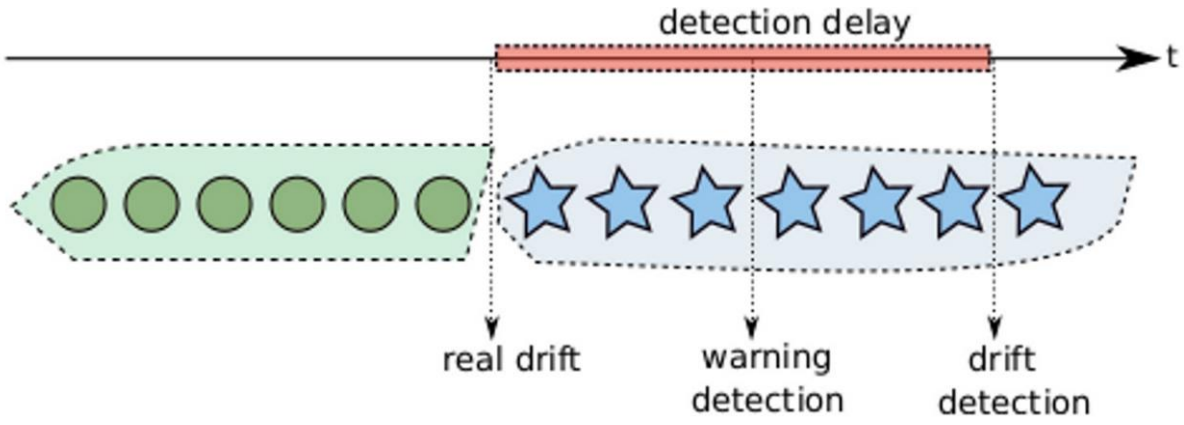


Рисунок 1.4 – Візуалізація роботи дрейф-детектора

Дрейф-детектори є важливим та проблематичним компонентом OL та аналізу потоку даних, але існує безліч проблем із побудови моделей ML, що будуть вчасно та ефективно ідентифікувати дрейфи. Моделі повинні одночасно вміти аналізувати послідовні дані із просторово-часовою природою, роботи це швидко та не виходячи за обчислювальні обмеження.

Традиційні моделі ML погано працюють із часовим виміром, як й звичайні ANN. З іншого боку, різні архітектури RNN мають спроможність до вивчення просторово-часових патернів із вхідних даних, але вони

страждають від проблем катастрофічного забування інформації. Також вони доволі ресурсоємні, довго вчаться і можуть бути доволі повільними при певних умов поставленої задачі.

Саме тому у OL (й не тільки в цій парадигмі) почали звертати більше уваги до SNN. Ці нейромережі мають певні особливості й переваги, порівняно із «класичними» ANN, які можуть дозволити ефективно використовувати їх у завданнях OL – як у адаптації до змін у потоці даних, так й у якості дрейф-детекторів.

Про ці особливості, – та як вони можуть допомогти у OL, – буде йти мова у наступному підрозділі.

## 1.2 Спайкові нейронні мережі

### 1.2.1 Особливості та відмінності спайкових нейронних мереж від штучних

Спайкові або імпульсні нейромережі – це третє покоління ANN, що мають за мету повністю імітувати механізми біологічних нейронів. Біологічний нейрон є свого роду акумулятором з вимикачем. Активація, або біологічно кажучи «закриття», нейрону трапляється, коли у його сприятливе поле (receptive field) накопичує певний рівень електронного заряду (заряд поступає при надходженні певних подразників до органів, тканин та, відповідно, нервову систему) і досягає певного порогу. Під час активації нейрон вистрілює потенціал дії – сплеск (spike) високою напруги. Після потенціалу дії на кінцях (терміналах) аксонів проходить вибух нейромедіаторів, передається до синапсу і прикріплюється до дендриту іншого нейрона, передаючи йому частину заряду, яка залежить від міцності зв’язку між ними. Після активації, нейрон стабілізується й входить у стан рефрактерного періоду (refractory period), під час якого дуже важко або в

деяких випадках неможливо активувати нейрон знов. Цей рефрактерний період виступає свого роду регуляризацією або нормалізацією, що не дозволяє нейронам переходити у хаотичну систему, що постійно передає інформацію (що буде дуже сильно споживати енергетичні ресурси).

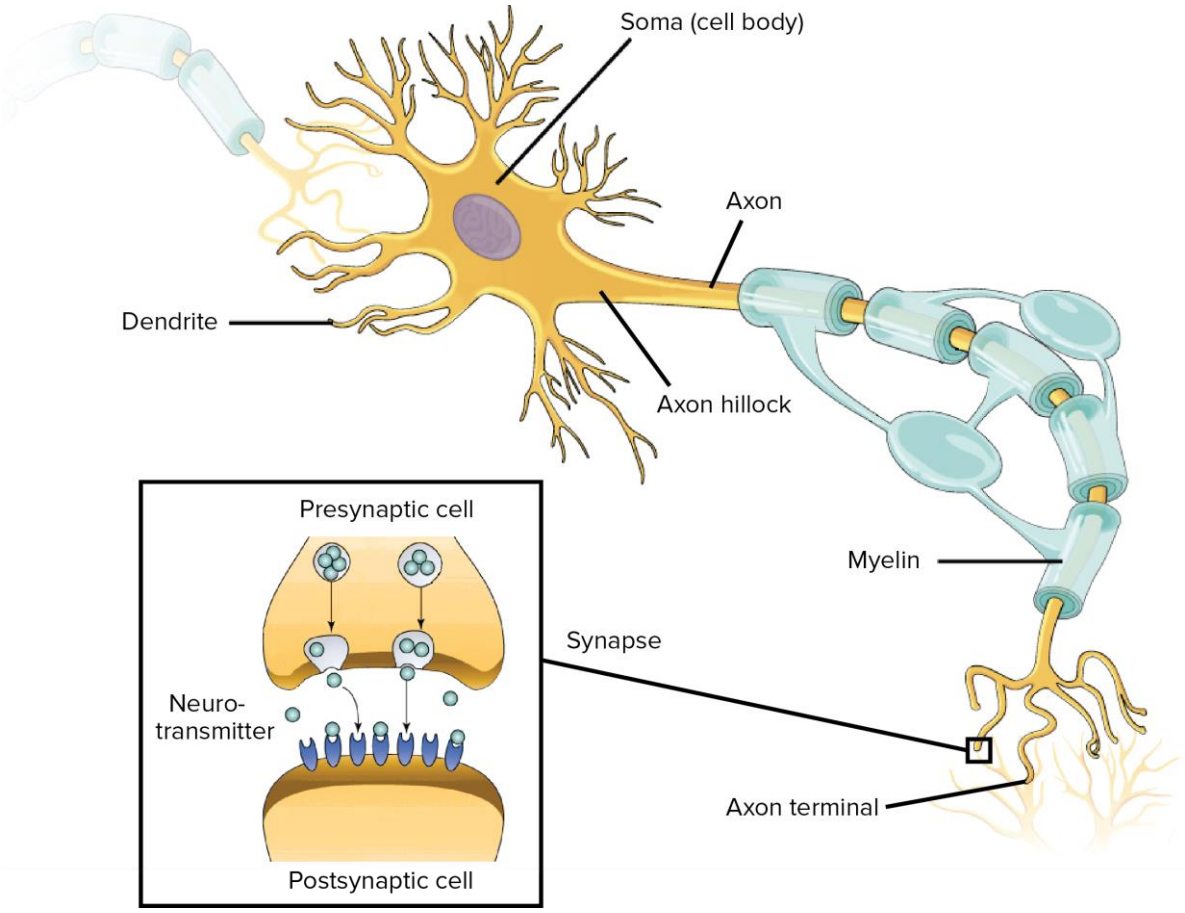


Рисунок 1.5 – Анатомія біологічного нейрону

Звичайні ANN першого та другого покоління (які ще відомі як бінарні та статистичні відповідно) частково імітують певні процеси біологічних нейронів: нейрони поєднані між собою й мають міцність зв’язку у формі матриці ваг, акумулюють (сумують) вхідні значення, активуються певними функціями, тощо. Але ця імітація є доволі абстрактною й, окрім структури, майже ніяк не відповідають природному оригіналу. Яскравий приклад – ANN не мають рефрактерного періоду та постійно передають інформацію

між собою. Вони не мають «подійно-часову» природу біологічних нейронів, де для активація нейрону потрібна серія поступаючих сигналів (імпульсів з інших нейронів) у певному інтервалі.

SNN у свою чергу намагаються повністю імплементувати всі біологічні процеси [2]. Різні математичні моделі (про них піде мова у пункті 1.2.2) імплементують процес накопичення потенціалу, рефрактерний період, розпад заряду, імітація вибуху нейромедіаторів (у вигляді певних активаційних порогових функцій), робота із аналоговим сигналом, подійний характер, передача інформація тільки при активації, тощо.

Й ще одна головна відмінність від ANN попереднього покоління – інтеграція часового простору у обчислення. Детально про це піде мова у розділі 2.

### 1.2.2 Основні математичні моделі

Як й серед звичайних ANN, існує кілька підходів до імплементації біологічно-подібних нейронів. Моделі нейронів відрізняються ступенем емуляції фізико-хімічних процесів мозку – деякі моделі намагаються повністю спроектувати усі реакції хімічних каналів, інші зупиняються тільки на більш абстрактних фізичних (електричних) закономірностях. Кожен з цих підходів має власні переваги, наприклад, більш детальні імплементації використовуються у нейробіологічних симуляціях, коли фізично-орієнтовані знаходять своє місце у сенсорах в робототехніці та IoT. Цей напрямок активно розвивається, активно з’являються принципово нові моделі або модифікації старих; і зараз не існує універсального рішення. Хоча реалізацій доволі немало, більшість з них базується на одному з трьох основних імплементацій. Саме про них і піде мова у цьому пункті.

1.2.2.1 Модель нейрону Ходжкіна-Хакслі

Перша й найдавніша математична модель була запропонована нейрофізіологами Аланом Ллойдом Ходжкіном та Ендрю Хакслі, які стали Нобелівськими лауреатами з фізіології та медицині у 1963 році. Їх модель нейрону [13] базується на найпоширеніших іонах, що беруть участь у виробленні потенціалу дії: канал натрію (Na+) та канал калію (K+). Також, окрім цих каналів, вони включили у свою модель явище витоку струму (leak current), що ще відомий у інших моделях як розпад заряду (про який вже була згадка).

На основі закону Ома вони вивели наступні рівняння:

$$I_{Na} = g_{Na}(E - E_{Na}), \tag{1.1}$$

$$I_K = g_K(E - E_K), \tag{1.2}$$

$$I_L = g_L(E - E_L). \tag{1.3}$$

Далі вони побудували електрону схему, де зазначені струми протікали у паралелі між собою та із конденсатор (див. рисунок 1.6). Відповідно до законів електроструму, вихідний струм схеми дорівнює:

$$I_i = I_{Na}(I_K - I_L), \tag{1.4}$$

$$I = C_m \frac{dV}{dt} + I_i, \tag{1.5}$$

де V – переміщення від потенціалу рівноваги через конденсатор.

Відповідно з цього, виводяться наступні формули:

$$I_{Na} = g_{Na}(V - V_{Na}) = g_{Na}(E_{Na} - E_R), \quad (1.6)$$

$$I_K = g_K(V - V_K) = g_K(E_K - E_R), \quad (1.7)$$

$$I_L = g_L(V - V_L) = g_L(E_L - E_R), \quad (1.8)$$

$$V = E - E_R, \quad (1.9)$$

де $E_k$ – потенціал рівноваги;

E – потенціал струму всередині мембрани.

$g_{Na}$, $g_K$ і $g_L$ – це провідність резисторів.

gL є фіксованою постійною величиною, тоді як $g_{Na}$ і $g_K$ залежать від кількості відкритих іонних каналів:

$$g_k = \overline{g_k} n^4, \quad (1.10)$$

$$g_{Na} = m^3 h \overline{g_{Na}}. \quad (1.11)$$

Параметри n, m та h вказують на частку відкритих каналів і змінюються за наступним законом:

$$\frac{d_n}{d_t} = \alpha_n(1 - n) - \beta_n n, \quad (1.12)$$

$$\frac{d_m}{d_t} = \alpha_m(1 - m) - \beta_m m, \quad (1.13)$$

$$\frac{d_h}{d_t} = \alpha_h(1 - h) - \beta_h h, \quad (1.14)$$

де $\alpha$ – константа швидкості відкриття;

$\beta$ – константа швидкості закриття іонних каналів.

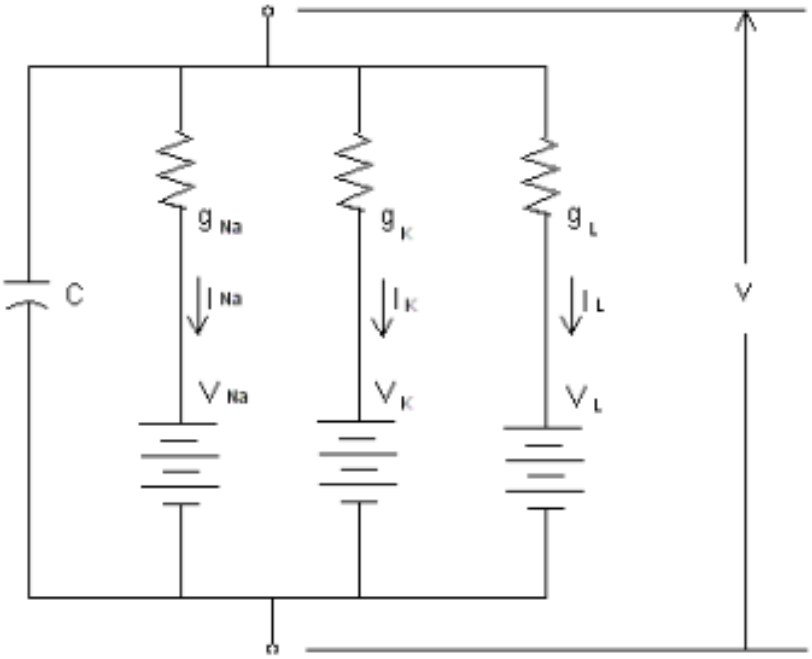


Рисунок 1.6 – Схема нейрону Ходжкіна-Хакслі

Хоча модель Ходжкіна-Хакслі гарно емулюють хімічні процеси біологічних нейронів, вона не підходить для створення великих систем AI та використання ML. Основна проблема – обчислювальні потреби такої емуляція, яка тільки збільшиться при додаванні інших іонних каналів або параметрів.

1.2.2.2 Модель Іжикевича

У своїй роботі [14] Євген Іжикевич запропонував більш високий рівень абстракції у моделі SNN. Він відійшов від хімічних властивостей певних іонних каналів до загальних фізичних закономірностей.

Він помітив, що основна різниця між струмом в іонах – частота імпульсів. Також він один з перших помітив важливість залежності передачі інформації у імпульсах від часу та інтервалу.

Розроблена їм універсальна модель нейрону описується наступними диференціальними рівняннями:

$$v' = 0.04v^2 + 5v + 140 - u + I, \quad (1.15)$$

$$u' = a(bv - u), \quad (1.16)$$

активація нейрону трапляється при перетині заданого порогу:

$$if\ v \geq 30mV\ then\ reset\ v = c\ and\ u = u + d, \quad (1.17)$$

де $v$ – мембранний потенціал;

$u$ – відновлення мембрани;

$a$ – шкала часу змінної відновлення $u$;

$b$ – чутливість змінної відновлення $u$;

$c$ – значення скидання після спайку мембранного потенціалу $v$;

$d$ –скидання після спайку змінної відновлення u.

В залежності від значень параметрів, змінюється частота імпульсів, формуючи «шаблони» імпульсів. Приклади наведені на рисунку 1.7:

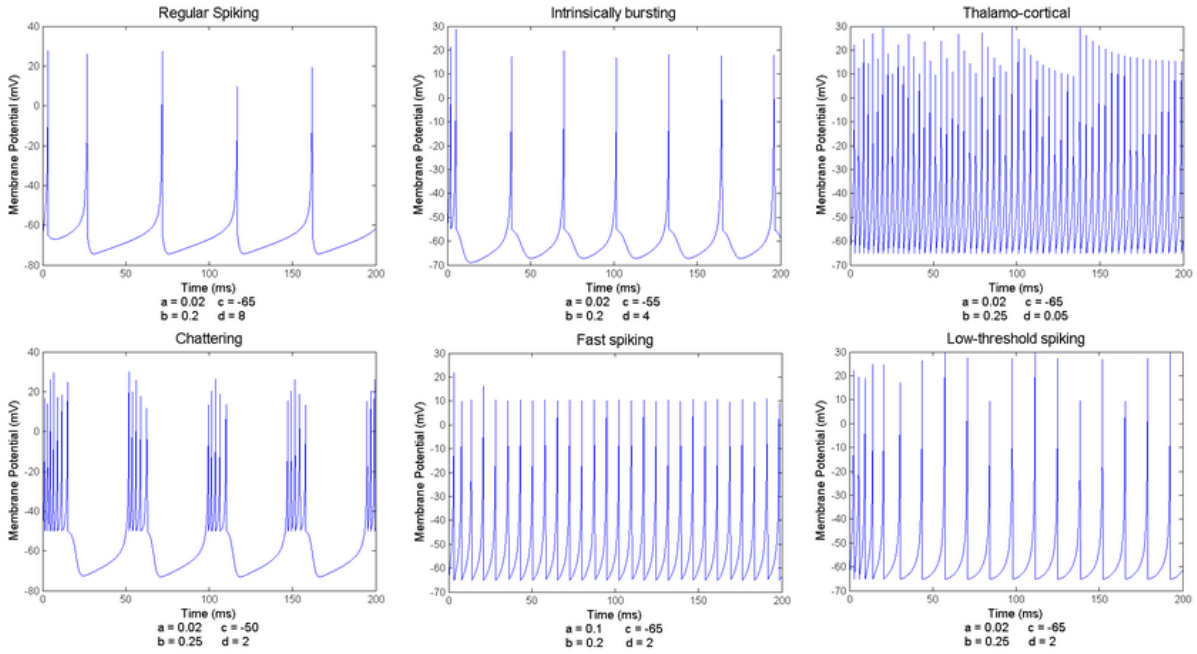


Рисунок 1.7 – Зміна частоти імпульсів від параметрів моделі

### 1.2.2.3 Leaky Integrate-and-Fire

Модель Leaky Integrate-and-Fire (LIF) є спрощеною версією моделі Іжикевича й є однією з найпростіших [15]. Вона досягає балансу між біологічно-правдоподібністю та ефективністю в обчисленнях. Саме ця модель та її модифікації є найрозповсюдженішою з усіх. Її активно використовують у дослідженнях та прикладних завданнях змагаючись із найкращими моделями ANN.

Формула моделя проста й описується наступним диференціальним рівнянням:

$$\tau \frac{du}{dt} = -u(t) + RI(t), \tag{1.18}$$

Цю модель та її модифікацію, що буде розглянуто у розділі 2, буде використано у подальшому дослідження та аналізі.

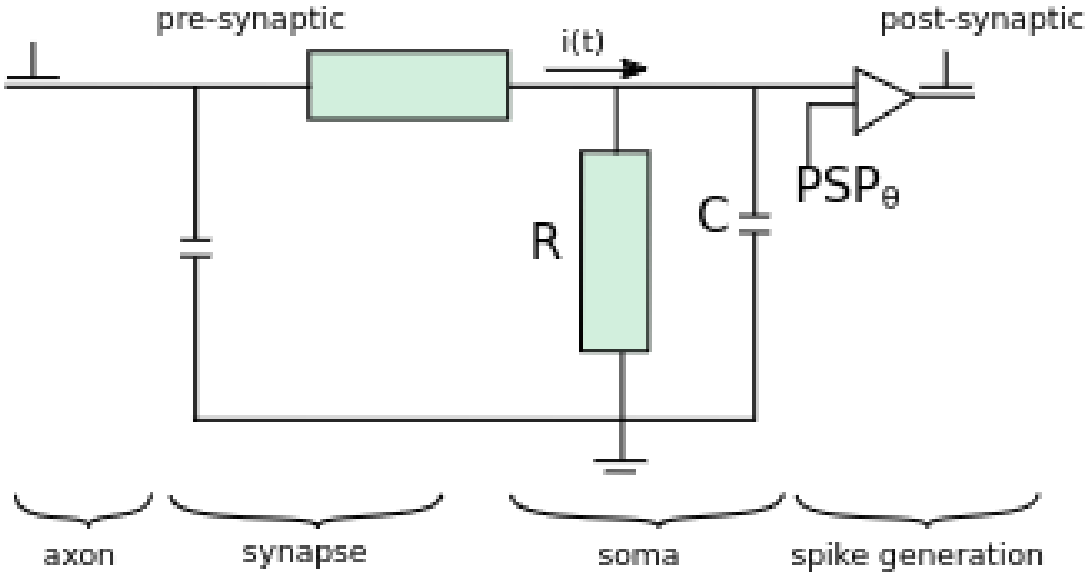


Рисунок 1.8 – Схема моделі LIF

### 1.2.3 Переваги використанням SNN над ANN

Хоч SNN ще тільки починають повноцінно використовуватись та розвиватись, і мають перелік власних проблем, вони мають ряд явних переваг. Ці переваги особливо помітні при використанні SNN у спеціалізованих сенсорних системах або на нейроморфних платформах, де кожен нейрон є окремим обчислювальним процесором. До основних переваг, порівняно із ANN, входять:

– висока паралелізація обчислень – як було вже описано, математичні моделі нейронів будуються на основі фізичних явищ, що гарно переноситься у електросхеми та сенсори реального часу. У поєднанні із передачою інформації тільки при активації, це дозволяє будувати спеціалізовані обчислювальні системи;

– обмін інформації тільки при активації нейрону – більшість активаційних функцій звичайних ANN постійно передають інформацію іншим нейронам, тільки модифікувавши їх значення, що є ресурсоємним та викликає такі потенційні проблеми, як катастрофічне забування [16]. Сам

процес обміну інформації не обов’язково потребує множення матриць – його можна реалізувати за допомоги додавання та зсувів;

– подійна та просторово-часова природа обробки інформації – це надає можливість збирати інформацію у режимі реального часу, як показано у дослідженнях [17], також це дозволяє ефективно аналізувати послідовності даних;

– псевдо-одночасна обробка інформації – через передачу інформації імпульсами й тільки при активації, є можливість спрогнозувати активність наступних шарів спостерігаючи тільки за першим [18], що може пришвидшити навчання;

– висока енергоефективність – SNN потребує в 1.72 рази менше енергії та пам’яті [3], що зумовлено іншими описаними перевагами.

Ці переваги не є єдиними, але вони гарно показують перспективи дослідження SNN із метою подальшої побудови інтелектуальних систем.

### 1.2.4 Перспективи використання SNN у онлайн-навчанні

Як вже зазначалось у попередніх пунктах, більшість моделей ML та ANN не можуть адекватно працювати в умовах OL. Але моделі SNN, зокрема LIF, зустрічають багато вимог онлайн-навчання: вони енергоефективні, легко паралізуються та працюють із часовим виміром. Потоки даних часто мають часові залежності між зразками та класами. Аналіз цієї залежності допомагає визначити відношення між вхідними характеристиками, що дозволяє майбутній моделі адаптуватись до дрейфу концепції або інших загальних змін у потоці.

SNN вже показали свою спроможність фіксувати часові асоціації у потоці даних [19], також SNN дозволяють працювати у асинхронному режимі, коли дані можуть поступити у будь-який момент часу й обробляються миттєво. Ще ефективними є біологічно-правдоподібні

правила навчання SNN, як Spike-timing-dependent plasticity та Hebbian Learning, що спираються на принципи локальності , що зменшує навантаження на обчислювальні ресурси.

Еволюційна та накопичувальна природа SNN робить можливим подолати недолік ANN та RNN у неможливості адекватно отримувати нові та забувати старі знання без необхідності перенавчання моделей.

Усе це, та інші переваги SNN, робить їх привабливими у OL. Також, певні модифікації LIF показали свою спроможність в якості дрейф-детекторів, що отримують конкурентні результати [20].

Протягом цієї роботи додатково й детально буде розглянуто ефективність SNN у аналізі просторово-часових патернів потоків даних для вирішення завдань OL.

### 1.2.5 Існуючі підходи застосування SNN у онлайн-навчання

В цілому, сумарно існує доволі мало проектів та досліджень із практичним застосуванням SNN у завданнях OL. Також вони мають власні недоліки. Серед них є:

– SpikeProp [21] – навчання SNN на основі backpropagation, хоча вони й вирішують важкі задачі класифікації, такий підхід часто впадає в локальний мінімум та погано масштабується. Також сам backpropagation є проблематичним для SNN, про це в деталях піде у пункті 1.2.6;

– Supervised Hebian Rule – цей метод підходить тільки для одношарових мереж і погано адаптується до OL;

– Метод SpikeTemp [22] – базується на навчанні на основі ранжування, будувався спеціально для OL і видає гарні результати;

– Використання еволюційних SNN (Evolving Spiking Neural Networks, eSNN) [23] – базується на використанні вбудованих можливостей LIF до накопичення та зберігання інформації та додаючи динамічну

структуру до них. На даний момент є одним із найкращих методів у OL і про нього ще буде окремо;

Як було зазначено, останній підхід є найефективнішим на даний момент для адаптації (та ідентифікації) дрейфу концепції. Але він більше фокусується на інкрементальному збільшенні нейромережі, що з часом може визвати проблеми. У цій науково-освітній роботі буде більше фокусу саме на властивостях LIF та інших SNN, що надає змогу аналізувати та адаптуватись до змін у просторово-часових патернах.

### 1.2.6 Існуючі проблеми спайкових обчислень

Хоча SNN мають перспективні переваги порівняно із ANN, зокрема у OL, вони мають власні недоліки, що не пов’язані із парадигмою. Але дослідники активно вивчають способи їх дослідження. Основні проблеми SNN це:

– Не конкурентоспроможні показники на типових для ML наборах даних, як той же MNIST [24] або CIFAR [25]. Це пов’язано із самою природою SNN та його «рецепторного поля» – йому потрібні події, які не існують у кадрових зображеннях. Але вже існують нейроморфні аналоги цих датасетів, що краще підлаштовуються під властивості SNN. Приклади датасетів: N-MNIST [26] та CIFAR10-DVS [27];

– Необхідність створення нових правил навчання. Як вже казалось, backpropagation погано працює із нейроморфною природою SNN через основну різницю у передачі інформації порівняно із ANN. Активно створюються нові правила, частина з яких було освітлено у попередньому пункті;

– Гірше працюють на класичних архітектурах процесору та, в ідеалі, потребують спеціалізованого нейроморфного обладнання, яке дуже складно отримати й потребує великої кваліфікації для використання.

Ці проблеми сильно гальмують розвиток SNN, але вони є тимчасовими. Навіть із ними, вже існують архітектури SNN, що конкурують із «state-of-the-art» межами ANN. Деякі з них буде зачеплено у наступних розділах.

## 1.3 Постановка завдання кваліфікаційної роботи

Отже, у цьому розділі було розглянуто основні теоретичні положення, щодо сучасного стану OL, наявних проблем і як SNN намагаються вирішити їх. Було згадано підходи у спайкових обчисленнях, що використовуються в задачах аналізу потоку даних, та були зазначені їх недоліки. Також було виявлено, що існуючі методи не повністю намагаються розкрити потенціал та спроможність SNN у вивчені просторово-часових патернів, що можуть значно покращити процес адаптації та ідентифікації дрейфу концепції, який є важливою перешкодую в системах OL.

Мета цієї роботи полягає у поглибленому дослідженні як просторово-часова та подійна природа SNN використовується або може використатись у розробці інтелектуальних систем аналізу потоку даних у парадигмі OL. Для досягнення поставленої мети необхідно виконати наступні завдання:

– провести теоретичне дослідження підходів та механізмів SNN для вивчення просторово-часових патернів даних;

– дослідити та підсумувати існуючі роботи, щодо можливостей моделей SNN адаптуватись до потоку даних із наявним дрейфом концепції;

– дослідити сучасні спроби використання SNN у якості дрейф-детекторів, їх ефективність та можливі покращення;

– провести аналіз яку роль певні компоненти або аспекти SNN беруть у процесі детекції дрейфу;

– поставити та провести експеримент із власноруч розробленою моделлю SNN-детектора для ідентифікації наявного різкого дрейфу на основі тільки вхідних даних;

– зробити висновки проведеного дослідження та визначити перспективи подальшого розвитку даного напряму.

# 2 СПАЙКОВІ НЕЙРОННІ МЕРЕЖІ У ЗАДАЧАХ ВИВЧЕННЯ ПРОСТОРОВО-ЧАСОВИХ ПАТЕРНІВ

## 2.1 Підходи до обробки даних із просторово-часовими залежностями

У минулому розділі були дані стислі теоретичні положення про SNN та їх основні відмінності від звичайних ANN. Серед цих відмінностей було згадано просторово-часову та подійну поведінку SNN, про часову зміну у диференційних рівняннях та накопичувальну природу потенціалу дії. Це надає вбудовану SNN короткочасову пам’ять, чого принципово не існує у ANN.

ANN досягає пам’яті шляхом імплементацій рекурентних шарів (RNN) або різноманітних блоків пам’яті, як LSTM [28] та GRU [29]. Ці підходи неоднократно демонстрували свою ефективність у завданнях BL, але вони часто не підходять для завдань OL через ресурсоємність, схильність до катастрофічного забування інформації й інших проблем.

SNN має два підходи до обробки просторо-часових патернів. Перший будується на тому ж принципі, що й ANN – використання рекурентних шарів або модифікації блоків пам’яті. Він був одним запропонований першим й був доволі логічним кроком при переході з ANN у SNN. Зокрема, він дуже ефективний при виконанні ANN2SNN конверсії, що є свого роду аналогом transfer learning. Другий, більш сучасний та активно досліджувальний, підхід – використовувати вбудовані можливості короткочасової пам’яті.

Обидва підходи будуть більш детально розглянути у наступних пунктах.

## 2.2 Використання рекурентних SNN

Цей підхід, як казалось у попередньому пункті, був першим запропонованим. Він доволі логічний й легко концептуально переноситься з ANN. Але в нього є ряд проблем.

Перше, що варто виділити, це проблеми використання рекурентних шарів SNN (Recurent Spiking Neural Network, RSNN). RSNN будуються й працюють за тими ж принципами, що й звичайні RNN: нейрони внутрішніх шарів входять у сами себе. Історично, в якості моделі використовували Spike Response Model [2]. Але разом із концептом, RSNN має ті ж самі проблеми, що й RNN: катастрофічне забування інформації, впадання у локальний мінімум, великі затрати в обчислювальних ресурсах. Ці пункти роблять RSNN неефективними як у OL (через наявність проблем, з якими OL не вміє працювати), так й із BL. Також, даний підхід дуже сильно зав’язаний із використанням backpropagation. Вже зазначалось, що backpropagation не є ефективним методом навчання SNN, бо він не розкриває усі унікальності природи SNN та, зазвичай, навчає моделі гірше за ANN (правда backpropagation часто дає адекватні результати й повністю від нього відмовлятись поки сенсу, та можливості, - нема). Але є одна перевага у цього методу – легка ANN2SNN конверсія, правда вона багаторазово показувала свою неефективність й дає тільки сенс у завданнях з використанням transfer learning.

Проблема із backpropagation пов’язана із подійною природою спайкових обчислювань. Алгоритми backpropagation засновані на диференціюванні функцій активацій. Якщо пригадати перший розділ, SNN активуються тільки при різниці потенціалу – коли накопичений заряд перевищує певний поріг. Тобто, функція активації працює із дискретними подіями, що, у свою чергу, не диференціюються.

Як вже було вказано раніше, на даний момент не існує універсального алгоритму навчання SNN. Зазвичай обходяться або ANN2SNN конверсією, підлаштуванням backpropagation під природу SNN (що все одно не є ідеальним), або використання біологічно-можливих алгоритмів, як вже згадані Spike-timing-dependent plasticity [30] або Hebian Learning [31], та також Winner-Takes-All [32] (алгоритми на основі конкуренції). Проблема із останніми, біологічно-можливими алгоритмами, що вони працюють тільки із SNN з маленькою кількістю шарів та тільки у навчанні без вчителя, що робить їх марними для навчання RSNN.

Тобто, у підсумку, сенсу використання RSNN – нема, тим паче у завданнях OL. Другий запропонований підхід – створення спайкових аналогів LSTM та GRU. Цей підхід також виявився неефективним – без модифікацій, що де-факто перетворюють SNN у ANN, вони не дають адекватних результатів [33].

Останнім підходом у використанні рекурентних SNN є створення нових «блоків». Яскравий приклад – Legendre Memory Units (LMU) [34]. Вони були спеціально створенні для роботи із безперервним часом, де диференційні рівняння лінійно відображаються на час відображається поліномом Лежандра до d-1 ступеня. Таким чином їх можна більш-менш ефективно використовувати із backpropagation. Формули мають наступний вигляд:

$$u(t-\theta') \approx \sum_{i=0}^{d-1} P_i\left(\frac{\theta'}{\theta}\right) m_i(t),$$

$$0 \le \theta' \le \theta,$$

$$P_i(r) = (-1)^i \sum_{j=0}^{i} \binom{i}{j}\binom{i+j}{j}(-r)^j. \qquad (2.1)$$

LMU гарно себе показали у завданнях аналізу хаотичних послідовностей (як permuted sequential MNIST [35]), де вони отримують метрики краще за ANN імплементації LSTM та GRU. При цьому LMU мають більшу ємність пам'яті – аж до 100 000 кроків. Також, завдяки схильності LMU вивчати інваріантні масштабом до розміру кроку функції, ці шари потребують значно менше часу на навчання та ресурсів – LMU потребують O(m) часу та пам'яті для m-шаровою мережі.

На абстрактному рівні, LMU має певні подібності із LSTM та GRU: вони також мають прихований стан $h_t$ та гейт пам'яті $m_{t-1}$. Різниця йде далі у обчислені внутрішньої логіки. Стан нейрону описується наступною формулою:

$$h_t = f(W_x x_t + W_h h_{t-1} + W_m m_t), \tag{2.2}$$

де $f$ – нелінійна функція (наприклад, tanh або sigmoid);

$W_x$, $W_h$, $W_m$ – вагові матриці.

Вхідний сигнал $u_t$ обчислюється таким чином:

$$u_t = e_x{}^T x_t + e_h{}^T h_{t-1} + e_m{}^T m_{t-1}, \tag{2.3}$$

де $e_x$, $e_h$, $e_m$ – отримані у процесі навчання кодуючи вектори.

Рішення про запис зміни у пам'ять рахується так:

$$m_t = \overline{A} m_{t-1} + \overline{B} u_t. \tag{2.4}$$

де $\overline{A}$ та $\overline{B}$ – дискретні матриці диференційних рівнянь для певного часового шагу $\Delta t$ та вікна довжиною $\theta$, які відрізняються залежно від обраного методу. Найпопулярніший метод – метод Ейлера.

Архітектура LMU зображена на рисунку 2.1:

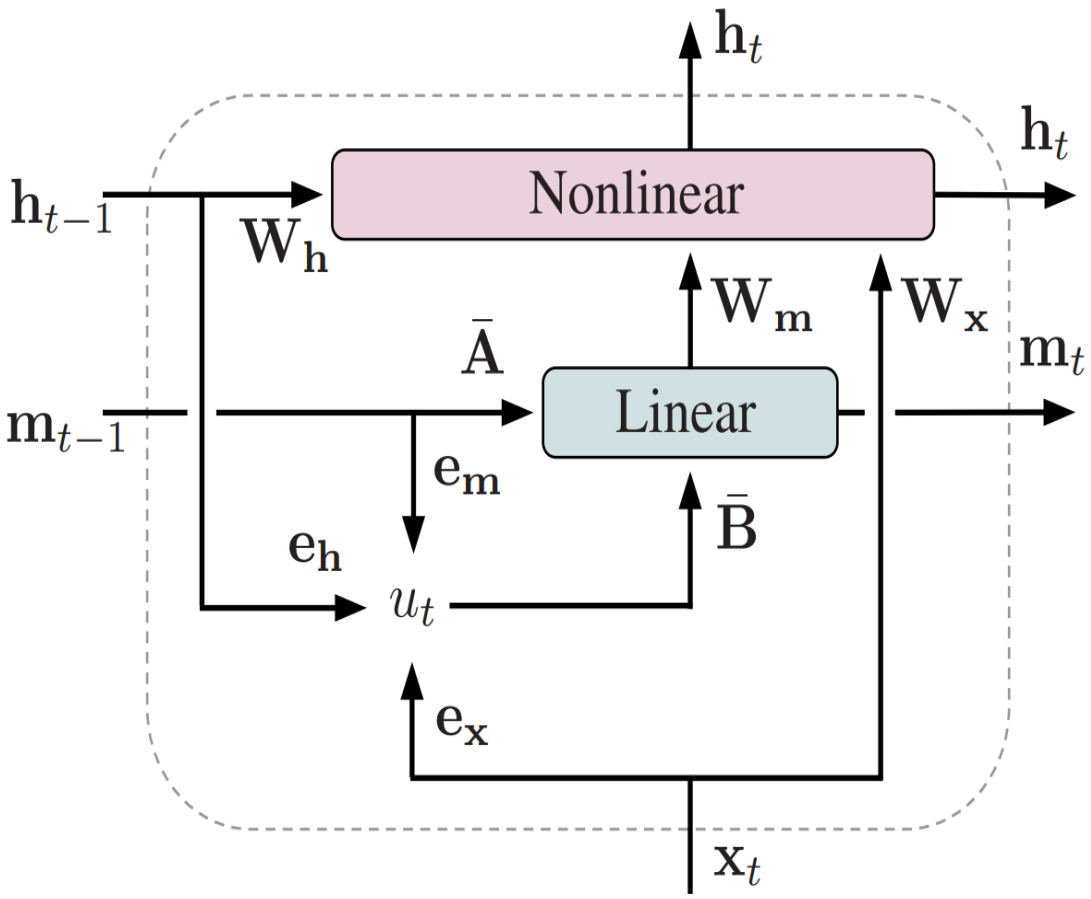


Рисунок 2.1 – Архітектура LMU

У підсумку можна зазначити, що хоч LMU виявив себе ефективним та конкурентно спроможнім у завданнях BL (у певних завданнях навіть вважається state-of-the-art), його важко застосувати у завданнях OL. В основному через необхідність довгого навчання й великих об’ємів даних.

Саме тому останнього часу дослідники почали звертати більше уваги до вбудованої у моделі SNN короткочасової пам’яті, про що піде мова у наступному підрозділі.

## 2.3 Використання вбудованих можливості до короткочасової пам’яті

У першому розділі було коротко згадано про те, що накопичувальна природа SNN дає їм вбудований механізм короткочасової пам’яті. Вікно

пам’яті значно менше ніж у LSTM або LMU, але при цьому цей механізм не потребує жодних змін та енергоефективний.

Насправді, SNN без цього компоненту не зможе нічого проаналізувати. Вже зазначалось, що SNN погано працює із звичними датасетами, як MNIST або CIFAR10. Це пов’язано із тим, що вони працюють з кадровим статичним зображенням. SNN, незалежно від моделі, необхідно дивитись на цей кадр кілька «квантів» часу (ітерацій циклу) перед тим як модель акумулює достатньо заряду для активації всіх шарів. Також їм потрібен якийсь час, щоб усі нейрони стабілізувались перед подачею наступних даних. І навіть при цьому це не ідеальне рішення, бо кадри залишаються статичними.

Нейроморфні датасети принципово відрізняються від цього тим, що замість статичних зображень, вони створюються завдяки руху датчика зору над кожним зображенням оригінального датасету [36]. Датчик формує відео довжина зазвичай в інтервалі від 100 до 500 мілісекунд, що розділяються на сакади (saccades) – заздалегідь визначені рухи камери. У N-MNIST, наприклад, це 360 мілісекунд розділені у 3 сакади по 105 мілісекунди та додаткові 45 мілісекунди, щоб останні кадри встигли пройтись через усю мережу.

Ця методика використовує часову та подійну природу SNN, дає їм зміну вхідного сигналу протягом часу й, тим самим, дає їм правильно активуватись. Це ще раз показує, наскільки короткочасова пам’ять та просторово-часова природа є невід’ємними частинами SNN.

Насправді, модель LIF та її модифікації мають схожу структуру із LSTM та GRU, й певні частини формули можуть бути розглянуті як «гейти» [37]. Для цього ще раз подивимось на формулу LIF (див. формулу 1.18 у підпункті 1.2.2.3) та перепишемо її у наступну форму:

$$H_t = \left(1 - \frac{1}{\tau}\right) V_{t-1} + \frac{1}{\tau} X_t, \tag{2.5}$$

У цій формулі вважається, що потенціал перезавантаження $V_{reset} = 0$. Ця формула складається з двох частин, розділених операцією додавання. Перша частина – це емуляція процесу витоку, що дозволяє LIF забувати стару інформацію. Друга частина – це емуляція процесу інтеграції, що дозволяє запам'ятати нову інформацію. Тобто, вони відіграють ту ж саму роль, що й гейти у LSTM. В даному випадку мембрана константа часу τ виступає «балансувальником» процесу запам'ятовування та забування.

Збільшення, або зменшення мембранної константи несе за собою зміни у швидкість здобуття та забування знань, та активацій нейронів в цілому. Менші значення константи роблять SNN більш чутливою: швидкість накопичування заряду зростає, як й швидкість повернення до спокою. Це робить нейрон значно легшим для активації. Така чутливість є доречною при необхідності ідентифікуванні миттєвих змін у поточному сигналі. Ця поведінка схожа із зміною матриці ваги між шарами, що також робить зв'язки більш чутливими. Закономірності у зміні мембранної константи та ваги зображені на рисунку 2.2:

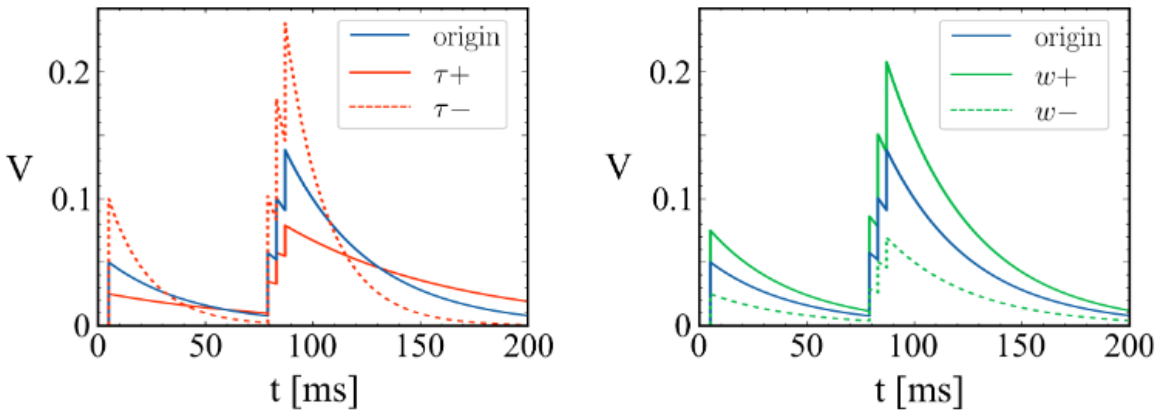


Рисунок 2.2 – Зміна чутливості при змінах параметрів моделі

Зазначені протягом цього розділу особливості дозволяє SNN, зокрема LIF та її модифікаціям, запам'ятовувати та аналізувати просторово-часові

патерни у вхідному потоці сигналів та даних. Поєднуючи це із дешевизною LIF, порівняно із RSNN, та іншими перевагами робить їх ідеальним кандидатами для OL та роботи із сенсорами (IoT).

Але існують також і різній модифікації LIF, що краще розкривають цей потенціал. Про одну з таких ми поговоримо у наступному підрозділі

## 2.4 Parametric Leaky Integrate-and-Fire нейрон

Модель LIF є неймовірно чудовим алгоритмом. Його жартівливо називають «ReLU світу спайкових обчислювань». Як й ReLU, він є «обчислювано мінімалістичним» та при цьому настільки ж ефективний, як й інші версії. Але, як й у ReLU, існують певні недосконалості, які активно намагаються виправити, створюючи певні модифікації.

Однією з таких модифікацій є модель нейрону Parametric Leaky Integrate-and-Fire (PLIF) [37]. Модифікація є доволі проста із мінімальною кількістю змін. Звичайний LIF під час навчання налаштовує тільки синаптичні ваги нейромережі. Усі інші важливі параметри, як поріг активації або потужність сигналу під час потенціалу дії, вказуються вручну і виступають в якості гіперпараметрів. Є різні версії LIF, де певні гіперпараметри переводять у тренувальні параметри. PLIF у свою чергу є такою модифікацію: окрім синаптичних ваг, ця модель тренує ще й мембрану часову константу $\tau$, про просторово-часову роль якої розповідалось у попередньому підрозділі.

В цілому, PLIF будується на наступних правилах:

– Мембрана часова константа є параметром, що оптимізується самостійно під час навчанням;

– Мембрана часова константа є уніфікованою для всього шару, що є біологічно-обґрунтованим, бо сусідні біологічні нейрону мають подібні властивості;

– Мембрана часова константа відрізняється між шарами.

Також дана модифікація збільшує гетерогенність мережі за мінімальні збільшення у обчислювальних затратах. Але існує проблема, що дана модель сильно спирається на backpropagation, що, як вже відомо, не є найоптимальнішим алгоритмом для SNN.

У оригінальній статті експериментально було доведено, що PLIF однакового розміру із LIF видає краще результати у завданнях комп'ютерного зору, незалежно від нейроморфності набору даних. Наприклад, при погано налаштованій мембранній константі, PLIF у аналізі CIFAR-10 має точність 93.23% порівняно із 47.50% (LIF), та для нейроморфного CIFAR10-DVS різниця є 70.50% та 62.40%. Також PLIF потребує доволі маленькі інтервали часу для аналізу зображення, що не перевищують 20 відміток часу, коли певні моделі потребують по кілька сотень.

На даний момент PLIF є однією з state-of-the-art моделлю для аналізу послідовних даних, видаючи конкурентні результати до більш складних моделей (як LMU), але потребуючи значно менше ресурсів та часу на обробку.

## 2.5 Висновки щодо використання SNN у аналізі просторово-часових патернів у потоці даних

Отже, у цьому розділі ми більш детально розглянули аспекти, що допомагають SNN вивчати складні просторово-часові патерни у потоках даних. Було розглянуто існуючі підходи до вирішення таких завдань, були зазначені спроби відтворити рекурентні шари у SNN та чому вони не набули ефективність. Було обговорено нові спайкові блоки пам'яті як LMU, їх особливості та відмінності, та їх переваги й успіхи у вирішенні практичних завдань.

Також було детально розглянуто вбудовані можливості спайкових обчислень до формування короткочасової пам’яті. На прикладі LIF було визначено які частини дозволяють нейрону зберігати та забувати інформацію, як мембрана константа часу може виступати в якості аналогу гейтів LSTM, як її зміни впливають на чутливість нейрону. До того ж, була розглянута модифікація PLIF, що додає можливість тренування мембранної константи.

Знаючи про можливості SNN вивчати просторово-часові патерни, ми тепер можемо дослідити як спайкові обчислення (зокрема LIF та PLIF) показують себе у завданнях OL, як їх вбудована короткочасова пам’ять дозволить ефективно із мінімальними ресурсами адаптуватись до дрейфу концепцій та ідентифікувати його. Саме це й буде розглянуто упродовж наступних розділів.

# 3 АДАПТАЦІЯ СПАЙКОВИХ МАШИН ДО ДРЕЙФУ КОНЦЕПЦІЇ

## 3.1 Стан сучасних досліджень у адаптації SNN до дрейфу концепції

У пункті 1.2.5 було обговорено існуючі підходи та алгоритми SNN у OL. Більшість з них, хоч і вміють працювати в завданнях OL, вони не спроможні це робити ефективно, як і звичайні ANN. Вони страждають від довгого навчання, великого розміру що кличе за собою високу ресурсоємність. За малим винятком, як SpikeTemp [22] (що базується на методах нейронної пластичності), у сценаріях OL бракує ефективних і масштабованих алгоритмів SNN.

Було вже згадано про активні та пасивні підходи до будівлі алгоритмів ML, що адаптуються до дрейфу концепцій. Активний підхід базуються на використанні допоміжних методів та моделей для детекції дрейфу та тільки після цього проводить перенавчання моделей, коли пасивний підхід будує моделі, що навчаються постійно.

На жаль, більшість моделей ML, зокрема ANN, не вміють ефективно та пасивно адаптуватись. Ті методики що працюють зазвичай будують на основі ансамблевих моделей, як, наприклад, у дереві Хофдінга [38]. У ANN існують підходи із використанням LSTM для аналізу послідовностей, але він потребує серйозної оптимізації гіперпараметрів, та має ряд недоліків, що були вказані у минулому розділі.

Останнім часом, дослідники в області спайкових обчислень почали звертати більше уваги до вбудованих можливостей певних моделей SNN як LIF, про які те ж йшла мова у попередньому розділі. До найуспішніших у OL модифікацій SNN відносять eSNN. Ми її вже згадували, але тепер ми детальніше роздивимось у наступному підрозділі.

## 3.2 Evolving SNNs

Ключова особливість цієї модифікації полягає у інкрементальній еволюції нейромережі, тобто із збільшенням кількості спайкових нейронів із часом, з метою виявлення часових патернів. Еволюційні мережі будуються на основі принципу еволюційних систем зв'язку (коннекціоністські системи). «Оригінальна» eSNN будувалась на основі моделей Торпа [39]. Але з часом почали з'являтись eSNN  на основі інші моделей спайкових нейронів, як LIF. Також зміни у вхідному потоці за-замовченням кодується у спайки – бінарні події. Згідно з досліджень [40], це є найоптимальніший спосіб кодування для адаптації до дрифту. Архітектура eSNN зображена на рисунку 3.1:

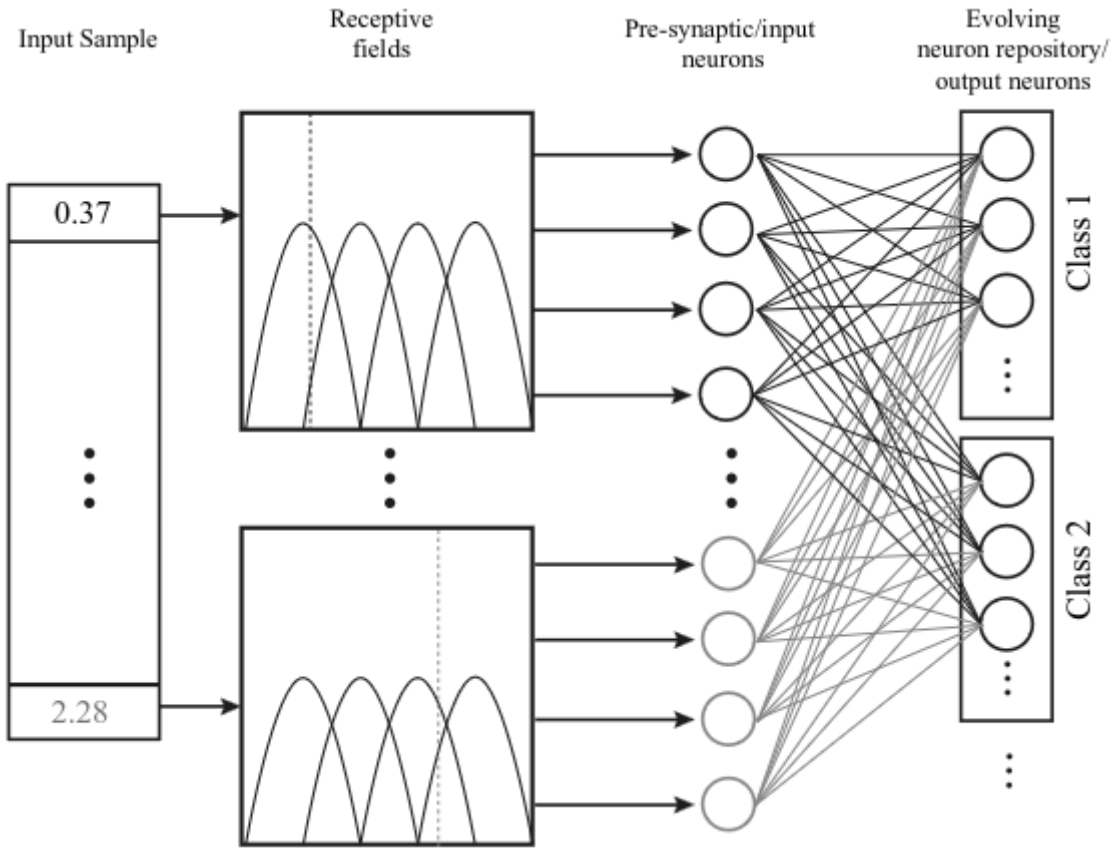


Рисунок 3.1 – Архітектура eSNN

Згадана стаття [40] також запропонована власну модифікацію eSNN. Всі перелічені підходи, й eSNN також, не враховують обмеження OL у пам’яті та ресурсах. Жоден з цих алгоритмів не намагається обмежити розмір репозиторію нейронів, що може призвести до поступового збільшення потреб в ресурсах, поки ці потреби не почнуть перевищувати задані обмеження, що призведе до колапсу всій системи.

Запропонована модифікація виправляє цей момент додаючи обмеження максимального розміру, а також адаптуючи методики зменшення даних для отримання набору даних меншого розміру з однаковою ступеню узагальнення. Ці методики зменшення даних основані на різних типах кластеризації методом сусідів або ієрархічної кластеризації. Ці методики зменшення даних застосовували до нейронного сховища в разі його заповнення, що дає можливість вивчення нової інформації.

Окрім цього вони також використовують спайкове кодування й дають можливість використання методику розсувного вікна для детекції дрейфу. Свою модифікацію автори назвали Online Evolving Spiking Neural Network (OeSNN).

Загальну схема роботи OeSNN наведено на рисунку 3.2:

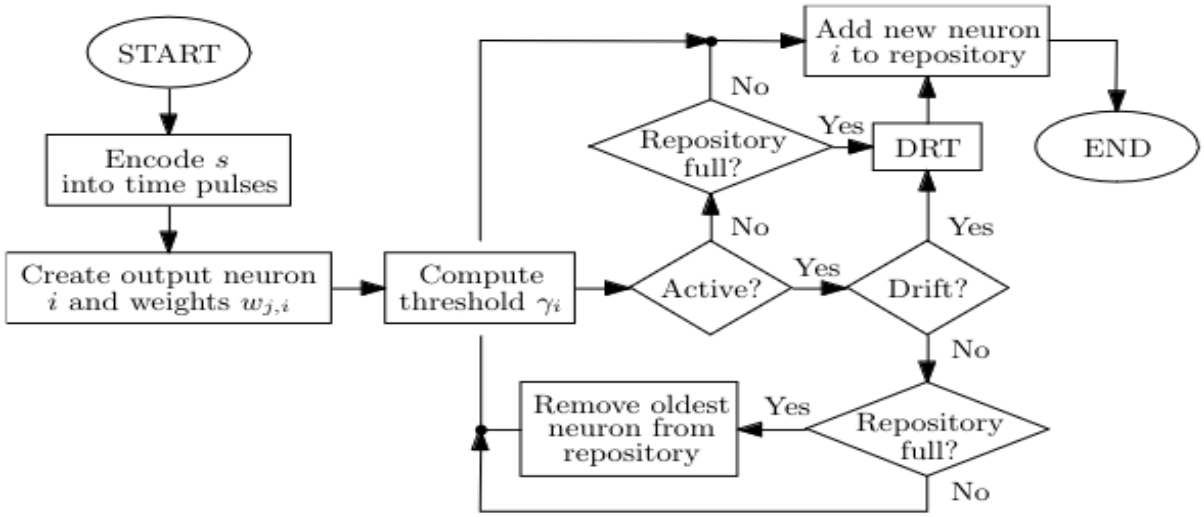


Рисунок 3.2 – Схема роботи OeSNN

## 3.3 Адаптація eSNN до дрейфу концепцій

Автори оригінальної статті OeSNN провели кілька емпіричних експериментів із штучними наборами даних CIRCLE, LINE, SIHEN, SINEV та ELEC2. Під час проведення експериментів, автори підтримувались стратегії максимально реалістичної емуляції процесі OL, зокрема, той факт, що у реальному сценарії не можливо заздалегідь оптимізувати параметри моделі, бо нема апріорних знань про природу потокових даних та статистичні розподіли в них. Також окрім точності, автори враховували та оптимізували такі метрики, як пластичність та відсоток зменшення розміру даних.

У результаті експерименту, було виявлено, що без використання методик зменшення розмірності даних, розмір нейронного сховища забивався за пару сотень ітерацій (дивись рисунок 3.3)

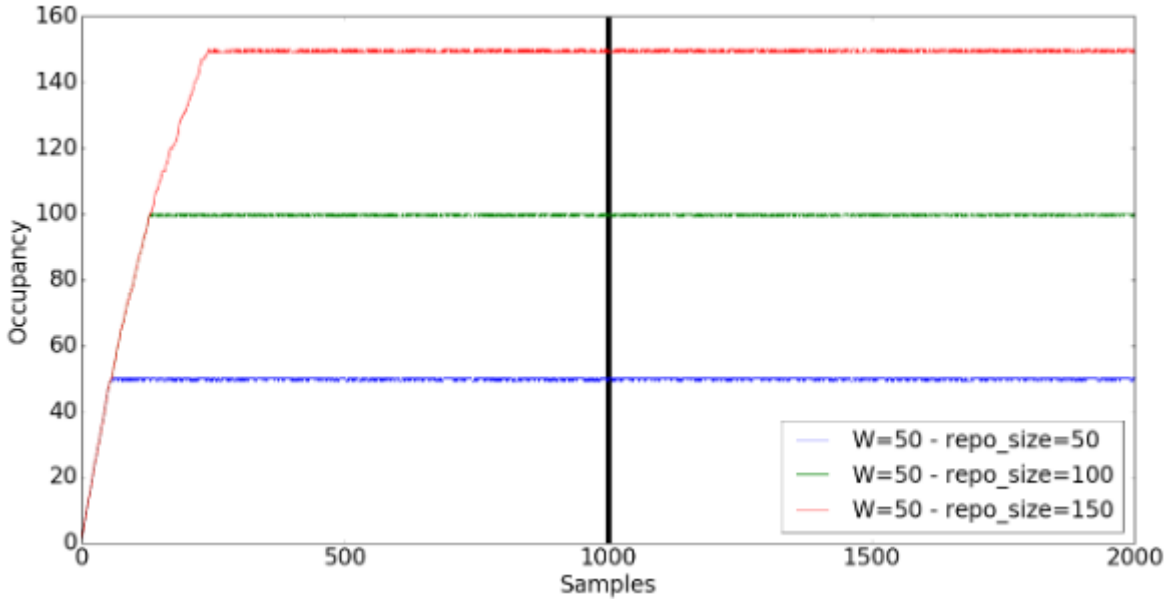


Рисунок 3.3 – Заповнення нейронного сховища без методик зменшення розмірності

Це призводить до того, що нейромережа перестає діставати нові знання з потоку даних та забувати старі. Також це ще раз вказує на проблему оригінального eSNN, що його нейронне сховище буде швидко зростати й потребувати все більше ресурсів.

При додаванні описаних методик зменшення розмірності даних на нейронне сховище в разі його заповнення, дана проблема вирішується (див. рисунок 3.4).

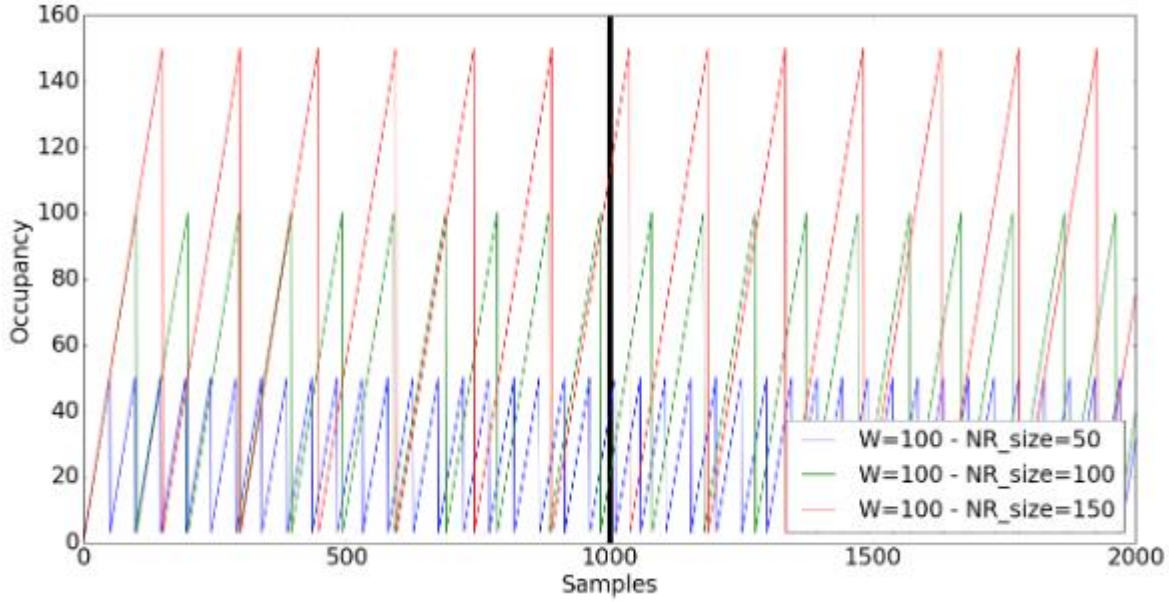


Рисунок 3.4 – Заповнення нейронного сховища з методиками зменшення розмірності

У результаті навчання OeSNN із методикою зменшення розмірності нейронного сховища було отримано маленьку мережу із сховищем до 50 LIF нейронів, що отримали точність у 81.5% на штучному наборі даних ELEC2. Цей результат є конкурентно спроможним із state-of-the-art результатами ансамблевих методів (що мають по 100 моделей у ансамблі). При цьому 50 нейронів будуть витрачати значно менше ресурсів. Порівняльні результати можна побачити у таблиці 3.1:

Таблиця 3.1 – Порівняння метрик OeSNN та ансамблевих методів у завданнях із адаптацією до дрейфу концепцій на наборі даних ELE2

| Модель | Тип моделі | Точність |
|---|---|---|
| OeSNN | Нейромережа (50 нейронів) | 81.5% |
| Adaptive RFs | Ансамбль (100 моделей) | 88.5% |

Продовження таблиці 3.1

| Модель | Тип моделі | Точність |
|---|---|---|
| Online Boosting | Ансамбль (100 моделей) | 90.01% |
| Online Bagging | Ансамбль (100 моделей) | 82.5% |

## 3.4 Висновки щодо адаптації SNN до дрейфу концепції

Отже, в цьому розділі було розглянуто сучасний стан досліджень у сфері адаптації моделей SNN до дрейфу концепції. Було зазначено існуючі моделі, їх особливості та недоліки. Детально було розглянуто модель eSNN та її модифікації, зокрема OeSNN. Було визначено як оригінальна модель надає можливість пасивної адаптації до дрейфу концепції завдяки нейронному сховищу, що інкрементально збільшується, а також як модифікація обмежує цей процес для економії ресурсів.

Також було проведено аналіз експерименту авторів моделі OeSNN, показано як імплементована ними процес зменшення нейронного сховища допомагає моделі вивчати нові знання та забувати старі із мінімальними затратами. Окрім цього, було зазначено конкурентно спроможність їхнього методу із state-of-the-art рішеннями з використанням ансамблевих моделей, що вважаються найспроможнішими у адаптації до дрейфу концепції на даний момент.

Але окрім адаптації до дрейфу, SNN спроможні й виступати в якості дрейф-детектору. У наступному розділі ми розглянемо як вказані алгоритми справляються із цією задачею та, також, плавно підведемо до нашого власного експерименту.

# 4 ВИКОРИСТАННЯ СПАЙКОВИХ НЕЙРОННИХ МЕРЕЖ У ІДЕНТИФІКАЦІЇ ДРЕЙФУ КОНЦЕПЦІЇ

## 4.1 Сучасний стан спайкових дрейф-детекторів

В цілому, сучасний стан спайкових дрейф-детекторів перетинається із сучасним станом спайкових мереж, що адаптуються до дрейфу. Зазвичай, ті ж самі механізми, що дозволяються моделям пасивно адаптуватись, також дозволяє їм й активно ідентифікувати його.

Алгоритми дрейф-детекторів кількісно характеризують події дрейфу шляхом визначення точок або проміжків часу із зміною концепції. В цілому, механізми цих детекторів розділяють на три основні групи [10]:

– Перший механізм використовує методи послідовного аналізу, як, наприклад, послідовний тест на відношення ймовірностей із використанням кумулятивної суми;

– Другий механізм – статичне управління процесом, що відстежує і контролює розвиток процесу навчання, як, наприклад, робить метод експоненціально зваженої ковзної середньої;

– Третій основний підхід – моніторинг розподілу даних у двох різних вікнах: якщо розподіли не збігаються, то вважається, що трапився дрейф концепції.

Третій підхід вважається найпопулярнішим на даний момент. Також є й метод шляхом аналізу потоку даних використовуючи нейроні мережі, але він має той же самий перелік проблем, що й при адаптації.

Що стосується SNN, то дослідження на даний момент фігурують навколо тих же eSNN. Наявний state-of-the-art підхід [41], що було розроблено авторами OeSNN описаний у попередньому розділі, це нейромережа Evolving Spiking Neural Network for Drift Detection (eSNN-DD). Їхня ідея базується на моніторингу стану нейронного сховища: вони

дивляться на процес з'єднання нейронів і оголошують дрейф тільки в ситуаціях, якщо жодного нейрону не було з’єднано. Єдиний виняток – якщо не було досягнуто максимального розміру сховища, то дрейф не буде оголошено.

Приклад цього процесу можна побачити на рисунку 4.1. На ньому зображені наявність процесу з’єднання й дрейф концепції на 1000 зразку. До дрейфу, процеси злиття відбувались періодично, тоді як одразу після дрейфу злиття не відбуваються доти, поки новий концепт не буде захоплений алгоритмом.

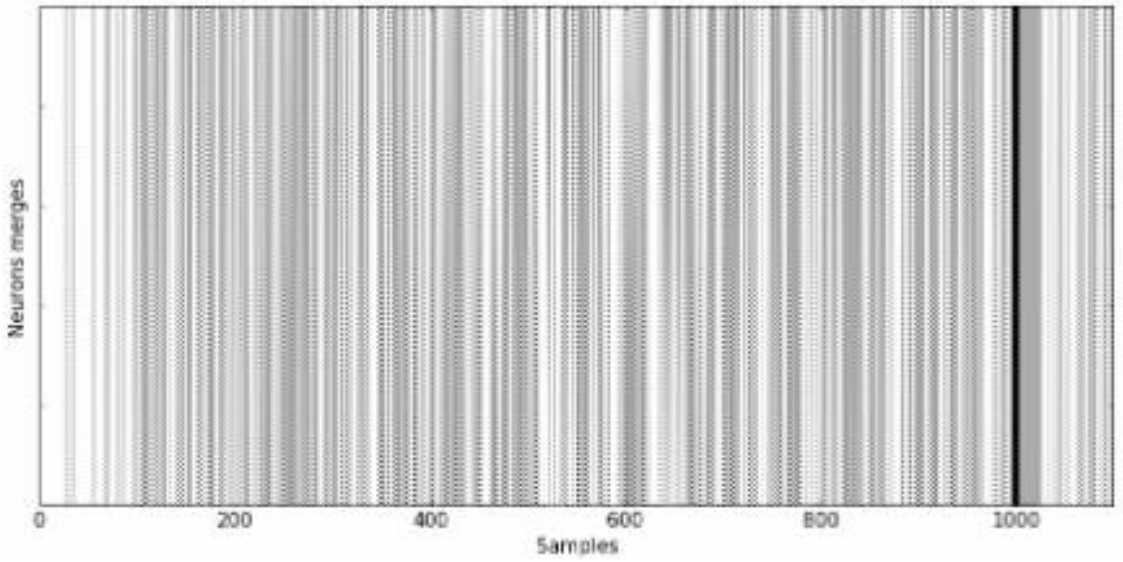


Рисунок 4.1 – Процес детекції дрейфу у eSNN

В оригінальній статті [41] було проведено порівняльний аналіз із методами виявлення дрейфу на основі границь Хофдінга, деякі результати з яких наведені у таблиці 4.1:

Таблиця 4.1 – Порівняння метрик eSNN-DD та дрейф-детектору на основі границь Хофдінга на штучному наборі даних CIRCLE

| Метод | Ітерацій перед детекцією | False Positive | True Positive | False Negative |
|---|---|---|---|---|
| Hoeffding | 310.5±127.76 | 4 | 4 | 1 |
| eSNN-DD | 191.8±107.35 | 19 | 5 | 0 |

Як видно з результатів, eSNN-DD успішно ідентифікував усі дрейфи та зробив це швидше, але при цьому зробив багато False Positive помилок.

## 4.2 Проблеми у наявних рішеннях

Загально, перелік проблем у дрейф-детекторів є аналогічним як й у пасивній адаптації. Модель eSNN-DD показує конкурентно спроможні результати із ансамблевими методами на основі границь Хофдінга, що є найпоширенішою моделлю дрейф-детектору. Але є один вагомий мінус у eSNN-DD – велика кількість False-Positive помилок. Це не є настільки критичним, як висока наявність False Negative, але у активних системах OL це може призвести до надлишкових перетренування моделей ML, що у свою чергу приведе до непотрібного використання обчислювальних ресурсів або часу спеціаліста на дослідження нової концепції. Авторами статті було зазначено, що кількість False Positive помилок може бути зменшено шляхом зменшення інтенсивності мережі, що призведе до більшого інтервалу детекції.

## 4.3 Висновки щодо сучасних методів застосування SNN в якості дрейф-детекторів

Отже, у цьому розділі було розглянуто сучасний стан застосування SNN в якості дрейф-детекторів. Було помічено, що загальний стан не сильно відрізняється від стану SNN у пасивної адаптації до дрейфу.

Окремо було проаналізовано одну з найкращих моделей SNN в якості дрейф-детектора, а саме eSNN-DD від авторів вже розглянутого OeSNN. Було виявлено, що eSNN-DD показує конкурентно спроможні результати із поширеними методами, як ансамблі на основі границі Хофдінга, але страждає від високої кількості False Positive помилок порівняно із кількості дрейфів.

Дані помилки викликають певні проблеми. Дрейф-детектори виступають в якості допоміжної моделі у активних системах адаптації до дрейфу. У більшості ситуацій, при детекції дрейфу модель або сама автоматично перенавчається, або спеціаліст починає вивчати дрейф і робити зміни у системі. Помилки False Positive призведуть до надлишкового перетренування основної моделі, що буде витрачати обчислювальні ресурси та час.

У наступному, останньому розділі буде запропоновано та описано власне рішення даної проблеми, а також буде проведено експеримент із порівнянням розробленої системи до state-of-the-art детекторів.

# 5 ЕКСПЕРИМЕНТ З ВИКОРИСТАННЯМ СПАЙКОВОЇ МАШИНИ У ІДЕНТИФІКАЦІЇ РІЗКОГО ДРЕЙФУ КОНЦЕПЦІЇ

## 5.1 Постановка експериментальної задачі

У попередньому розділі ми детальніше розглянули сучасний стан SNN-детекторів. Було виявлено, що існує, з одного боку, доволі мало досліджень й розробок у цій сфері; але, з іншого боку, існують методики побудови SNN алгоритмів, які є конкурентно спроможними із state-of-the-art ансамблевими рішеннями. При цьому було зазначено, що існує проблема із наявністю великої кількості False Positive помилок, коли модель ідентифікує неіснуючі дрейфи концепції. Ця проблема може призвести до надмірного перенавчання моделі, що буде впливати на стабільність всієї системи.

Фокус цієї експериментальної задачі полягає у розробці методу та алгоритму навчання, а також моделі SNN-детектора, що буде мінімізувати False Positive помилки, при цьому не погіршуючи знаходження справжніх дрейфів. Буде використовуватись дещо інший підхід для навчання, ніж у eSNN – замість спостерігання за зміною внутрішніх параметрів під час пасивної адаптації, буде пряма робота із вхідними даними, а в якості цільової зміни виступає наявність дрейфу. Загальні характеристики розробленого алгоритму навчання є такими:

– Ознаки та класи вхідного набору даних об’єднуються у один вектор та подаються на вхід моделі, при цьому модель є агностичною до того, який вхідний нейрон був ознакою чи класом. Таким чином модель буде вивчати залежності між параметрами агностично до їх джерела, тобто мережа буде вивчати та ідентифікувати як реальний, так й віртуальний дрейф одночасно;

– Модель SNN буде навчатись оптимізуючи метрику L1-Loss, яка об’єднує в собі метрику precision та recall. Тобто, модель буде вчитись мінімізувати False Positive й False Negative помилки;

– Із заданим інтервалом буде траплятись різкий дрейф концепції, відлік нового інтервалу не починається поки не було ідентифіковано наявний дрейф;

– Для моделі SNN не дається можливість прийти до стану спокою між ітераціями, як це зазвичай робиться;

– Модель повинна бути нескладною.

Розроблених підхід намагається напряму експлуатувати накопичувальні можливості SNN та використовувати якомога менше параметрів. В якості моделі нейрону було обрано PLIF за його простоту та ефективність.

Для тестування алгоритму, було обрано набір даних «A streaming ensemble algorithm» [42]. Він генерує 3 числові атрибути, що мають значення від 0 до 10, при цьому тільки 2 з них мають вплив на цільову зміну. Ключова зміна має два унікальних значення, які залежать від налаштованої функції, які перемикаються під час різкого дрейфу концепції. Ці функції наведено у формулах 4.1-4.4:

$$if(att1 + att2 \geq 8)then1else0, \quad (4.1)$$

$$if(att1 + att2 \geq 9)then1else0, \quad (4.2)$$

$$if(att1 + att2 \geq 7)then1else0, \quad (4.3)$$

$$if(att1 + att2 \geq 9.5)then1else0, \quad (4.4)$$

Потік даних був налаштований без балансування класів та без додаткового шуму. Також, як було зазначено раніше, ознаки та клас потоку об’єднується в єдиний вектор.

Щодо структури моделі, був обраний підхід використання мережі мінімального розміру. Розроблена SNN має всього два шари: перший шар є вхідним, а другий складається з PLIF для проведення обчислення й вивчення просторо-часових закономірностей. Другий шар також виступає в якості вихідного шару. Тобто, розроблена мережа має всього 7 шарів та 16

тренувальних параметрів, 14 з яких відповідають за синапси (матриця ваги) та 2 за внутрішні мембрану зміну часу. Вибір такого розміру зумовлений простотою набору даних, складністю навчання глибинних SNN та для наочності потужності SNN аналізувати просторово-часові патерни. Серед параметрів моделі було обрано крок навчання 0.001, початкове значення мембранної зміни часу у 5 та оптимізацію алгоритмом Adam.

Для порівняння, також був використаний ансамбль дерев Хофдінга, він також вчиться за описаним алгоритмом. Дерево Хофдінга активно використовується як у дослідженнях, так й у реальних практичних задачах.

В якості засобів реалізації було обрано мову програмування Python, для інструментів OL був обраний модуль scikit-multiflow [43] та для роботи із спайковими обчисленнями був обраний модуль SpikingJelly [44] завдяки його зручному інтерфейсу, гарною інтеграцією до PyTorch [45], а також за наявність імплементації PLIF (PLIF та SpikingJelly мають спільних авторів). Повний код програми наведено у додатку А.

Результати експерименту описані у підрозділі 5.2.

## 5.2 Результати проведеного експерименту

Кожна з моделей тренувалась 200 ітерацій, оптимізуючи метрику L1-Loss. В середньому SNN мала значно краще результати, а саме меншу кількість ітерацій перед ідентифікацією дрейфу, а також значно меншу кількість False Positive. Детальну статистику можна побачити у таблиці 5.1:

Таблиця 5.1 – Результати проведеного експерименту

| Модель | Ітерації перед ідентифікацією дрейфу | Precision | Accuracy | False Positives |
|---|---|---|---|---|
| SNN | 14 | 83% | 58% | 1 |
| Hoeffding Tree | 17.4 | 34% | 54% | 10 |

Як видно з результату, хоча точність розроблених моделей близькі один до одного, середній час відгука SNN менший (й час поступово зменшується) та precision значно більше. Якщо збільшити кількість ітерацій, то метрики у дерева Хофдінга залишаються незмінними; а у SNN зменшується кількість ітерацій перед ідентифікацією дрейфу до 10.32, але й зменшується precision до 67%.

Графіки зміни точності, а також моменти настання дрейфу, його ідентифікації та False Positive помилки можна побачити на рисунках 5.1 та 5.2:

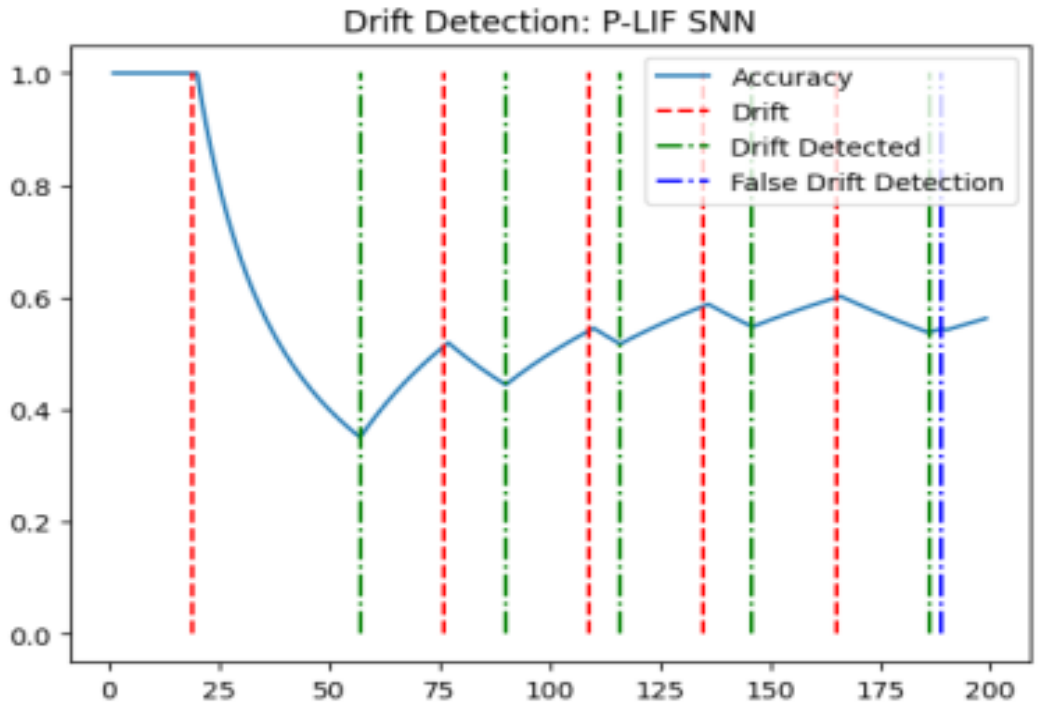


Рисунок 5.1 – Процес навчання моделі SNN

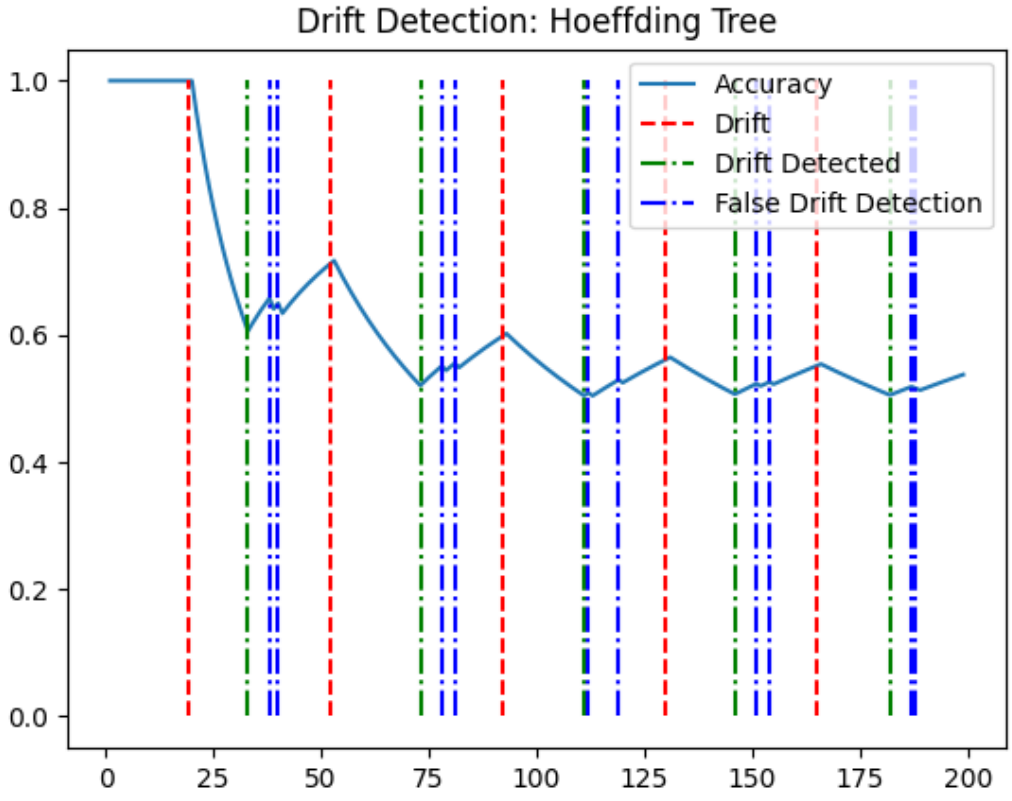


Рисунок 5.2 – Процес навчання дерева Хофдінга

Але результати SNN різняться в залежності від параметрів мембранної зміни часу: чим більше значення, тим менше False Positive помилок, але при цьому зменшується інтервал перед детекцією. Із додатковими ітераціями ця різниця поступово зменшується завдяки оптимізації. Наприклад, якщо зменшити зміну з 5 до 2, то будуть наступні результати (див. рисунок 5.3):

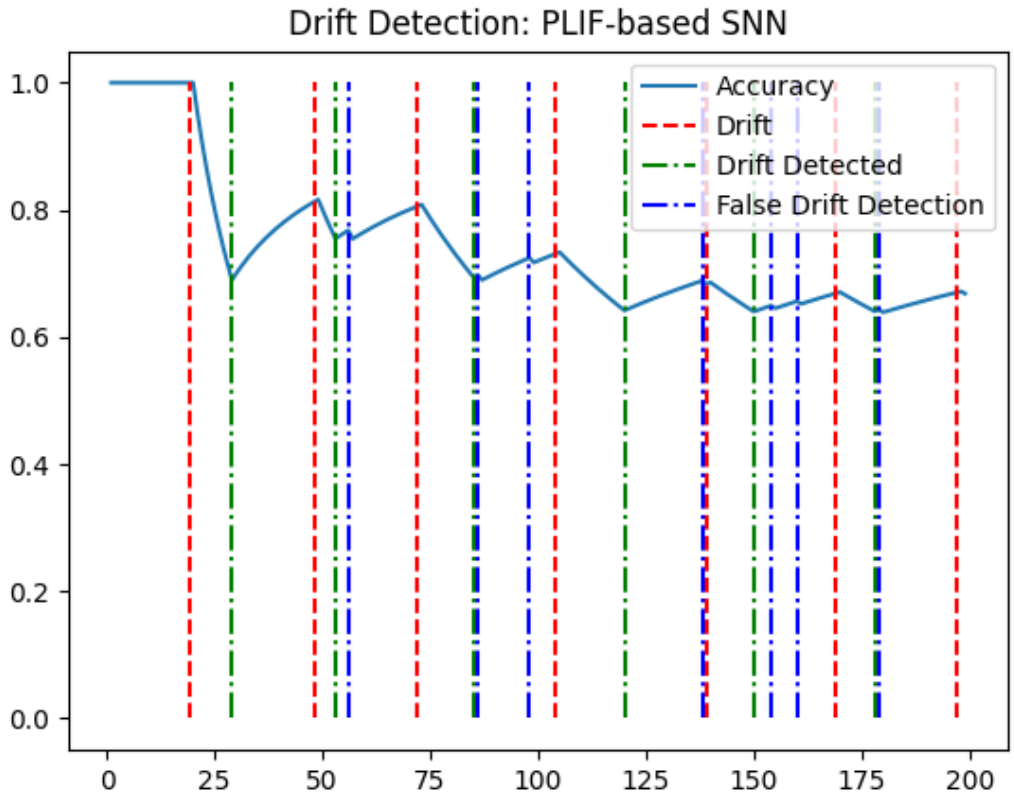


Рисунок 5.3 – Процес навчання моделі SNN із меншим τ

При τ = 2, precision падає до 46%, але й інтервал також зменшився до 10.6 ітерацій. Додатково з графіків видно, що SNN на відміну від Хофдінга має меншу проблему з повторними знаходження неіснуючого дрейфу одразу після попереднього. У SNN менше ця проблема через рефрактерний період, але існують моменти, коли обидва вихідні нейрони активуються близько

один до одного або паралельно, що викликано недосконаліть архітектури. У наступному підрозділі будуть наведені потенційні покращення архітектури та інші перспективи розвитку.

## 5.3 Перспективи розвитку

Розроблена архітектура нейромережі гарно себе показала на тренувальних даних, не зважаючи на власну простоту. Але результати далекі від оптимальних і рішення потребує покращень, зокрема можна виділити наступні перспективи:

– Проведення додаткових експериментів із різними даними, а також додавання більше моделей для порівняння;

– Провести оптимізацію гіперпараметрів, як падіння напруги або чутливість нейронів. Експерименти показали, що початковий параметр $\tau$ напряму впливає на кількість False Positive помилок та інтервал між появою дрейфу та його детекцією. Також це покращення може бути досягнуто використанням інших моделей LIF, що оптимізують більше параметрів;

– Дослідити спроможність SNN-детекторів ідентифікувати дрейфи інших типів. На даний момент, більшість досліджень сфокусовано на різкого дрейфі. Один з проведених під час виконання даної кваліфікаційної роботи показав неформалізований експеримент, що PLIF може навчитись адаптуватись до рекурсивного дрейфу, коли вихідна зміна має певне постійне значення протягом певного інтервалу незалежно від вхідних даних, після чого змінюється на інше значення, яке має ідентичну поведінку. PLIF через кілька таких змін навчився з ідеальною точністю прогнозувати цільову зміну. Цей експеримент не був формалізованим й потребує подальшого дослідження;

– Покращити архітектуру – на даний момент архітектура є занадто простою. Це дає мережі перевагу у OL тим, що вона споживає мало ресурсів

і зокрема може бути корисною у інтеграції з сенсорами, але у більшості завдань мережа може дозволити бути більшою. Потенційне рішення – інтеграція механізмів eSNN у розроблений принцип навчання;

– Змінити структуру цільової зміни – у нашому рішення є два класи: наявність чи відсутність дрейфу. Але з цим є проблема: обидва вихідні нейрони можуть активуватись паралельно; на даний момент це вирішується за допомоги взяття argmax від напруги виходів, але це може призвести до ігнорування детекції дрейфу. Можлива альтернатива – залишити тільки один вихідний нейрон, активація якого означатиме наявність дрейфу;

– Використати інші алгоритми навчання – backpropagation не є оптимальним рішенням, біологічно-подібні правила як STDP можуть краще підійти для такого роду завдань.

## 5.4 Висновки з проведеного експерименту

Отже, у цьому розділі було описано проведений експеримент із навчанням простої PLIF-моделі в якості різкого дрейф-детектору. Побудований алгоритм навчання є агностичним до джерела вхідних даних тим самим аналізуючи як зміни серед ознак, так й зміни серед цільової зміни потоку даних. Також алгоритм оптимізує L1-Loss, тобто намагається мінімізувати як False Positive, так й False Negative помилки.

Результати навченої моделі було порівняно із найпоширенішим ансамблевим OL-методом. Навчана SNN виявилось ефективною, отримуючи однакову точність із деревом Хофдінга, при цьому маючи кращий precision та менший середній інтервал перед детекцією. Було проаналізовано як деякі параметри та гіперпараметри впливають на ефективність мережі.

Додатково було проведено аналіз наявних проблем у запропонованому рішенні та виділено перспективи майбутнього його

розвитку та потенційні покращення. Було визначено недоліки реалізованого алгоритму, як можливість обох вихідних нейронів які відповідають за наявність чи відсутність дрейфу концепції, що є антагоністичними класами. Було вказано потенційні рішення, як використання тільки одного вихідного нейрону, що відповідає за наявність дрейфу в разі його активації, а в інших моментах часу, коли нейрон знаходиться у процесі накопичування заряду, концепція вважається незмінною. Ще було вказано можливі покращення у архітектурі та алгоритму в цілому, а також необхідність проведення експериментів для ідентифікації нерізких дрейфів.

Незважаючи на виявлені проблеми та простоту моделі, алгоритм показав свою адекватність та конкурентно-спроможність. Даний підхід можливо використовувати у практичних задачах онлайн-навчання та аналізу потоку даних в умовах потенційної наявності дрейфу. Зокрема, модель та алгоритм можуть бути ефективними в ситуації край обмежених обчислювальних ресурсів, як, наприклад, у сенсорах систем IoT.

# ВИСНОВКИ

Під час виконання даної освітньої-наукової кваліфікаційної роботи було проведено аналізу сучасного стану OL, існуючі проблеми та обмеження цієї парадигми, а також як певні аспекти та особливості SNN можуть допомогти у вирішенні їх. Було досліджено подійну природу SNN та як окремі її компоненти дозволяють вивчати просторо-часові патерни у потоці даних. Також було проведено власне дослідження із використання просторо-часової природи спайкових обчислень для ідентифікації різкого дрейфу у сценаріях OL використовуючи виключно дані потоку із агностичним ставленням моделі до джерела кожного окремого елементу вхідного вектору (тобто без розуміння моделі де ознаки, чи ключове значення оригінального потоку). Завдання, що були поставлені у початку кваліфікаційної роботи, були виконані повністю.

Проведене експериментальне дослідження активно використовує сучасні методики онлайн-навчання та state-of-the-art рішення у області спайкових обчислень. Розроблений алгоритм навчання дає конкурентно-спроможні результати із малими розмірами навчальної моделі, та доповнює існуючі наукові знання, намагаючись виправити нехтування просторо-часовою природою та вбудованою короткочасовою пам’яттю у наявних дослідженнях, а також нехтування мінімізацією False Positive помилок.

Результати експериментального дослідження було опубліковано у молодіжному форумі ХНУРЕ 2023 року у конференції «Інформаційні інтелектуальні системи», секції №1 «Сучасні проблеми обчислювального й штучного інтелекту» (УДК 004.8:[004.89]). У цій кваліфікаційній роботі було додатково проведено аналіз недоліків розробленого алгоритму, а також було зазначено перспективи розвитку та потенційні модифікації. Зокрема, мова йшла про покращення архітектури, виправлення проблеми паралельної активації вихідних нейронів в умовах їх несумісності, та використання SNN

для детекції дрейфів інших типів, зокрема у детекції рекурсивних дрейфів, що можуть мати циклічну чи іншу просторово-часову поведінку.

Отримані систематизація знань та результати проведеного дослідження можна використати у подальшому розвитку предметної галузі, для застосування у практичних задачах реального світу та в якості освітнього матеріалу у закладах вищої освіти за напрямом комп’ютерних наук або штучного інтелекту. Розроблений алгоритм та модель зокрема рекомендується застосовувати у ситуаціях надзвичайно обмежених обчислювальних ресурсів, як у системах IoT, зокрема в ситуаціях необхідності вбудування інтелектуальної автоматизованої системи в окремі сенсори та електронні компоненти.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. A survey on data preprocessing for data stream mining: Current status and future directions / S. Ram´ırez-Galleg et al. *Neurocomputing*. 2017. No. 239. P. 39–57.

2. Gerstner W., Kistler W. Spiking neuron models: Single neurons, populations, plasticity. Cambridge : Cambridge University Press, 2002.

3. Davidson S., Furber S. B. Comparison of Artificial and Spiking Neural Networks on Digital Hardware. *Frontiers in Neuroscience*. 2021. Vol. 15. URL: https://doi.org/10.3389/fnins.2021.651141.

4. Voelker A., Rasmussen D., Eliasmith C. A Spike in Performance: Training Hybrid-Spiking Neural Networks with Quantized Activation Functions. *Arxiv*. 2020.

5. Spiking Neural Networks and Online Learning: An Overview and Perspective / J. Lobo et al. *ecno- logico Bizkaia,*. 2019. No. 700.

6. Online Spatio-Temporal Learning in Deep Neural Networks / T. Bohnstingl et al. *IEEE Transactions on Neural Networks and Learning Systems*. 2022. P. 1–15. URL: https://doi.org/10.1109/tnnls.2022.3153985 (date of access: 30.04.2023).

7. Out-of-Core GPU-Accelerated Causal Structure Learning / C. Schmidt et al. *Algorithms and Architectures for Parallel Processing*. Cham, 2020. P. 89–104. URL: https://doi.org/10.1007/978-3-030-38991-8_7 (date of access: 30.04.2023).

8. Wang S., Minku L. L., Yao X. A Systematic Study of Online Class Imbalance Learning With Concept Drift. *IEEE Transactions on Neural Networks and Learning Systems*. 2018. Vol. 29, no. 10. P. 4802–4821. URL: https://doi.org/10.1109/tnnls.2017.2771290 (date of access: 30.04.2023).

9. Characterizing concept drift / G. I. Webb et al. *Data Mining and Knowledge Discovery*. 2016. Vol. 30, no. 4. P. 964–994. URL: https://doi.org/10.1007/s10618-015-0448-4 (date of access: 30.04.2023).

10. A survey on concept drift adaptation / J. Gama et al. *ACM Computing Surveys*. 2014. Vol. 46, no. 4. P. 1–37. URL: https://doi.org/10.1145/2523813 (date of access: 30.04.2023).

11. Oliveira G., Minku L. L., Oliveira A. L. I. Tackling Virtual and Real Concept Drifts: An Adaptive Gaussian Mixture Model Approach. *IEEE Transactions on Knowledge and Data Engineering*. 2021. P. 1. URL: https://doi.org/10.1109/tkde.2021.3099690 (date of access: 30.04.2023).

12. Minku L. L. Online ensemble learning in the presence of concept drift : thesis. 2011. URL: http://etheses.bham.ac.uk//id/eprint/1334/ (date of access: 30.04.2023).

13. Hodgkin A. L., Huxley A. F. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*. 1952. Vol. 117, no. 4. P. 500–544. URL: https://doi.org/10.1113/jphysiol.1952.sp004764 (date of access: 30.04.2023).

14. Izhikevich E. M. Simple model of spiking neurons. *IEEE Transactions on Neural Networks*. 2003. Vol. 14, no. 6. P. 1569–1572. URL: https://doi.org/10.1109/tnn.2003.820440 (date of access: 30.04.2023).

15. Thorpe S., Gautrais J. Rank Order Coding. *Computational Neuroscience*. Boston, MA, 1998. P. 113–118. URL: https://doi.org/10.1007/978-1-4615-4831-7_19 (date of access: 30.04.2023).

16. Schak M., Gepperth A. A Study on Catastrophic Forgetting in Deep LSTM Networks. *Lecture Notes in Computer Science*. Cham, 2019. P. 714–728. URL: https://doi.org/10.1007/978-3-030-30484-3_56 (date of access: 30.04.2023).

17. Kasabov N. K. NeuCube: A spiking neural network architecture for mapping, learning and understanding of spatio-temporal brain data. *Neural Networks*. 2014. Vol. 52. P. 62–76. URL: https://doi.org/10.1016/j.neunet.2014.01.006 (date of access: 30.04.2023).

18. Comparison between Frame-Constrained Fix-Pixel-Value and Frame-Free Spiking-Dynamic-Pixel ConvNets for Visual Processing / C. Farabet et al.

*Frontiers in Neuroscience*. 2012. Vol. 6. URL: https://doi.org/10.3389/fnins.2012.00032 (date of access: 30.04.2023).

19. SPAN: SPIKE PATTERN ASSOCIATION NEURON FOR LEARNING SPATIO-TEMPORAL SPIKE PATTERNS / A. MOHEMMED et al. *International Journal of Neural Systems*. 2012. Vol. 22, no. 04. P. 1250012. URL: https://doi.org/10.1142/s0129065712500128 (date of access: 30.04.2023).

20. Drift Detection over Non-stationary Data Streams Using Evolving Spiking Neural Networks / J. L. Lobo et al. *Intelligent Distributed Computing XII*. Cham, 2018. P. 82–94. URL: https://doi.org/10.1007/978-3-319-99626-4_8 (date of access: 30.04.2023).

21. Bohte S. M., Kok J. N., La Poutré H. Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing*. 2002. Vol. 48, no. 1-4. P. 17–37. URL: https://doi.org/10.1016/s0925-2312(01)00658-0 (date of access: 30.04.2023).

22. SpikeTemp: An Enhanced Rank-Order-Based Learning Approach for Spiking Neural Networks With Adaptive Structure / J. Wang et al. *IEEE Transactions on Neural Networks and Learning Systems*. 2017. Vol. 28, no. 1. P. 30–43. URL: https://doi.org/10.1109/tnnls.2015.2501322 (date of access: 30.04.2023).

23. SOLTIC S., KASABOV N. KNOWLEDGE EXTRACTION FROM EVOLVING SPIKING NEURAL NETWORKS WITH RANK ORDER POPULATION CODING. *International Journal of Neural Systems*. 2010. Vol. 20, no. 06. P. 437–445. URL: https://doi.org/10.1142/s012906571000253x (date of access: 30.04.2023).

24. Gradient-based learning applied to document recognition / Y. Lecun et al. *Proceedings of the IEEE*. 1998. Vol. 86, no. 11. P. 2278–2324. URL: https://doi.org/10.1109/5.726791 (date of access: 30.04.2023).

25. Krizehvsky A. Learning Multiple Layers of Features from Tiny Images. 2009.

26. Converting Static Image Datasets to Spiking Neuromorphic Datasets Using Saccades / G. Orchard et al. *Frontiers in Neuroscience*. 2015. Vol. 9. URL: https://doi.org/10.3389/fnins.2015.00437 (date of access: 30.04.2023).

27. CIFAR10-DVS: An Event-Stream Dataset for Object Classification / H. Li et al. *Frontiers in Neuroscience*. 2017. Vol. 11. URL: https://doi.org/10.3389/fnins.2017.00309 (date of access: 30.04.2023).

28. Sherstinsky A. Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network. *Physica D: Nonlinear Phenomena*. 2020. Vol. 404. P. 132306. URL: https://doi.org/10.1016/j.physd.2019.132306 (date of access: 30.04.2023).

29. Empirical Evaluation of Gated Recurrent Neural Network Architectures in Aviation Delay Prediction / J. Chung et al. 2014. P. 9.

30. Markram H., Gerstner W., Sjöström P. J. Spike-Timing-Dependent Plasticity: A Comprehensive Overview. *Frontiers in Synaptic Neuroscience*. 2012. Vol. 4. URL: https://doi.org/10.3389/fnsyn.2012.00002 (date of access: 30.04.2023).

31. Choe Y. Hebbian Learning. *Encyclopedia of Computational Neuroscience*. New York, NY, 2014. P. 1–5. URL: https://doi.org/10.1007/978-1-4614-7320-6_672-1 (date of access: 30.04.2023).

32. Ferré P., Mamalet F., Thorpe S. J. Unsupervised Feature Learning With Winner-Takes-All Based STDP. *Frontiers in Computational Neuroscience*. 2018. Vol. 12. URL: https://doi.org/10.3389/fncom.2018.00024 (date of access: 30.04.2023).

33. Conversion of Continuous-Valued Deep Networks to Efficient Event-Driven Networks for Image Classification / B. Rueckauer et al. *Frontiers in Neuroscience*. 2017. Vol. 11. URL: https://doi.org/10.3389/fnins.2017.00682 (date of access: 30.04.2023).

34. Voelker R., Kajic I., Eliasmith C. Legendre Memory Units: Continuous-Time Representation in Recurrent Neural Networks. 2020. P. 11.

35. Goodfellow I., Mirza M., Xiao D. An Empirical Investigation of Catastrophic Forgetting in Gradient-Based Neural Networks. *Arxiv*. 2013.

36. Iyer L. R., Chua Y., Li H. Is Neuromorphic MNIST Neuromorphic? Analyzing the Discriminative Power of Neuromorphic Datasets in the Time Domain. *Frontiers in Neuroscience*. 2021. Vol. 15. URL: https://doi.org/10.3389/fnins.2021.608567 (date of access: 30.04.2023).

37. Incorporating Learnable Membrane Time Constant to Enhance Learning of Spiking Neural Networks / W. Fang et al. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, Montreal, QC, Canada, 10–17 October 2021. 2021. URL: https://doi.org/10.1109/iccv48922.2021.00266 (date of access: 30.04.2023).

38. Hulten G., Spencer L., Domingos P. Mining time-changing data streams. *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. 2001. P. 97–106.

39. Thorpe S., Delorme A., Van Rullen R. Spike-based strategies for rapid processing. *Neural Networks*. 2001. Vol. 14, no. 6-7. P. 715–725. URL: https://doi.org/10.1016/s0893-6080(01)00083-1 (date of access: 30.04.2023).

40. Evolving Spiking Neural Networks for online learning over drifting data streams / J. L. Lobo et al. *Neural Networks*. 2018. Vol. 108. P. 1–19. URL: https://doi.org/10.1016/j.neunet.2018.07.014 (date of access: 30.04.2023).

41. Drift Detection over Non-stationary Data Streams Using Evolving Spiking Neural Networks / J. L. Lobo et al. *Intelligent Distributed Computing XII*. Cham, 2018. P. 82–94. URL: https://doi.org/10.1007/978-3-319-99626-4_8 (date of access: 30.04.2023).

42. Street N., Kim Y. A streaming ensemble algorithm (SEA) for large-scale classification. *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. 2001. P. 377–382.

43. scikit-multiflow. *scikit-multiflow*. URL: https://scikit-multiflow.github.io/ (date of access: 30.04.2023).

44. (SpikingJelly) – spikingjelly alpha. *(SpikingJelly) – spikingjelly alpha*. URL: https://spikingjelly.readthedocs.io/ (date of access: 30.04.2023).

45. PyTorch. *PyTorch*. URL: https://pytorch.org/ (date of access: 30.04.2023).

# ДОДАТОК А

Програмний код проведеного експерименту

Лістинг А.1 – Програмний код моделі SNN

```
import torch
from spikingjelly.activation_based import layer, neuron,
surrogate
from torch import nn

class SingleLayerLIF(nn.Module):
    def __init__(self, n_input, n_output, tau=1):
        super().__init__()
        self.n_input = n_input
        self.n_output = n_output
        self.tau = tau

        self.layers = nn.Sequential(
            layer.Flatten(),
            layer.Linear(n_input, n_output, bias=False),
            neuron.ParametricLIFNode(
                init_tau=tau,
                surrogate_function=surrogate.ATan()
            ),
        )

    def forward(self, x: torch.Tensor):
        return self.layers(x)
```

Лістинг А.2 – Програмний код алгоритму навчанням

```
import random
import matplotlib.pyplot as plt
import numpy as np
import torch
import torch.nn.functional as F
```

```
from skmultiflow.data import SEAGenerator
from spikingjelly import visualizing
from tqdm import tqdm

random.seed(0)
torch.manual_seed(0)

NUM_OF_ITERATIONS = 200
DRIFT_STEP = 20
LEARNING_RATE = 1e-4
TAU = 5.0

stream = SEAGenerator(
    random_state=145,
    balance_classes=False,
    noise_percentage=0
)

from models.snn.single_layer_lif import SingleLayerLIF

n_input = stream.n_features + 1
n_output = 2
model = SingleLayerLIF(n_input=n_input,
n_output=n_output, tau=TAU)


def train_concept_drift_detection(model, stream,
num_of_iterations=NUM_OF_ITERATIONS,
drift_step=DRIFT_STEP, T=10, to_plot=True):
    optimizer = torch.optim.Adam(model.parameters(),
lr=LEARNING_RATE)

    metric_history = []

    true_positives = 0
```

```
    false_positives = 0
    false_negatives = 0

    drifts = []
    drift_detected = []
    false_drift_detected = []

    is_stream_drifted = False
    last_drift_detection_i = 0
    last_drift_i = 0
    drift_detection_time = []
    l1_loss = []
    for i in tqdm(range(num_of_iterations)):
        optimizer.zero_grad()
        x, stream_y_true = stream.next_sample()
        X = np.concatenate((x, [stream_y_true]), axis=1)

        X = torch.from_numpy(X).float()

        out_fr = 0.
        for t in range(T):
            out_fr += model(X)
        out_fr = out_fr / T

        y_pred_drift = bool(out_fr.argmax(1)[0])
        metric_history.append(int(is_stream_drifted ==
y_pred_drift))  # update the metric history

        # loss = F.mse_loss(out_fr,
torch.Tensor([int(is_stream_drifted)]))
        y_true = torch.Tensor([0, 1]) if
is_stream_drifted else  torch.Tensor([1, 0]) # TODO:
remake it to one hot
        # loss = F.mse_loss(out_fr, y_true)
        loss = F.l1_loss(out_fr, y_true)
```

```
            l1_loss.append(loss.item())
            loss.backward(retain_graph=True)
            optimizer.step()

            # stream will be drifted only after previous
drift was detected and drift_step steps passed
            if (i - last_drift_detection_i + 1) >= drift_step
and is_stream_drifted is False:
                drifts.append(i)
                stream.generate_drift()  # y is drifted
                is_stream_drifted = True
                last_drift_i = i
            if is_stream_drifted is False and y_pred_drift is
True:
                false_drift_detected.append(i)
                false_positives += 1
            elif is_stream_drifted is True and y_pred_drift
is True:
                drift_detection_time.append(i - last_drift_i)
                last_drift_detection_i = i
                drift_detected.append(i)
                is_stream_drifted = False
                true_positives += 1
            elif is_stream_drifted is True and y_pred_drift
is False:
                false_negatives += 1

        accuracy = [sum(metric_history[:i]) /
len(metric_history[:i]) for i in range(1,
num_of_iterations)]

        if to_plot:
            num_of_iterations = len(metric_history)
            time = [i for i in range(1, num_of_iterations)]
            # time = [i for i in range(len(l1_loss))]
```

```
        plt.plot(time, accuracy, label="Accuracy")
        plt.vlines(x=drifts, color='r', ymin=0., ymax=1.,
label='Drift', linestyle='--')
        plt.vlines(x=drift_detected, color='g', ymin=0.,
ymax=1., label='Drift Detected', linestyle='-.')
        plt.vlines(x=false_drift_detected, color='b',
ymin=0., ymax=1., label='False Drift Detection',
linestyle='-.')
        plt.title(f"Drift Detection: PLIF-based SNN")
        plt.legend()
        plt.show()


    print(f'last acc: {accuracy[-1]: 0.2f}, avg acc:
{np.mean(accuracy): 0.2f}, max acc: {np.max(accuracy):
0.2f}')
    # print(f'last acc: {accuracy: 0.2f}')
    print(
        f"Precision: {true_positives / (true_positives +
false_positives)}; recall: {true_positives /
(true_positives + false_negatives)}")
    print(f"Average time needed to detect drift:
{sum(drift_detection_time) / len(drift_detection_time)}")


train_concept_drift_detection(model, stream,
num_of_iterations=200, to_plot=True)
```

# ДОДАТОК Б

Відомість кваліфікаційної роботи

| Позначення | Найменування | Дод. відомості |
|---|---|---|
| | Текстові документи | |
| **1.** | Пояснювальна записка | 75 с. |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | Інші документи | |
| | | |
| **2.** | Презентаційні матеріали | __ плакатів |
| | | |
| | | |
| | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | | | |
| | | | | | | | |
| . | . | Прізвище та ініціали. | Підп. | Дата | | | |
| Розробив | | Савенков Д.В. | | | Дослідження застосування spiking нейронних мереж у задачах розпізнавання просторово-часових патернів в онлайн-навчанні | Шифр групи | Код напр./спец. |
| Перевірив | | Семенець В.В. | | | | СШІм-21-1 | 122 |
| Н.контр. | | Малєєва І.А. | | | | ХНУРЕ кафедра ШІ | |
| Затв. | | Філатов В.О. | | | | | |