Київський національний університет імені Тараса Шевченка
Факультет радіофізики, електроніки та комп'ютерних систем
Кафедра комп'ютерної інженерії

ДИПЛОМНА РОБОТА МАГІСТРА

на тему:

«Прогнозування вторинної структури білка за допомогою штучних нейронних мереж»

«Protein secondary structure prediction with artificial neural networks»

Виконав студент 2 курсу ОР «Магістр»
спеціальності 123 «Комп'ютерна інженерія»
Володимир БОРИС
_____(підпис)

Науковий керівник: асистент кафедри
комп'ютерної інженерії
к.ф.-м.н. **Андрій КОНОВАЛОВ**
_____(підпис)

До захисту допускаю
Протокол засідання кафедри від
"___" _____2023р. №_____

Завідувач кафедри
к.ф.-м.н., доцент
Юрій БОЙКО

КИЇВ 2023

Реферат

Дипломна робота магістра містить 34 сторінки, 11 рисунків, 1 таблицю, 1 додаток, використано 16 інформаційних джерел.

ПРОГНОЗУВАННЯ, ВТОРИННА СТРУКТУРА БІЛКА, МАШИННА МОДЕЛЬ, НЕЙРОННА МЕРЕЖА, РЕКУРЕНТНА НЕЙРОННА МЕРЕЖА, LSTM, GRU, НАВЧАННЯ, ГІПЕРПАРАМЕТРИ, ДОСЛІДЖЕННЯ, МЕТОДИ РЕГУЛЯРИЗАЦІЇ, CULLPDB, CB513, PYTHON, KERAS.

Метою роботи є вдосконалення моделі нейронної мережі для 8-станового прогнозування вторинної структури білка.

Результати роботи: вдосконалено машинний класифікатор на базі нейронної мережі для прогнозування вторинної структури білка за допомогою методів регуляризації, а також вибору оптимальної архітектури (5 згортокових шарів з розмірами фільтрів 3, 5, 7, 9, 11 та з 64 фільтрами кожний, дві двонаправлені рекурентні нейронні мережі на основі GRU з 500 та 250 штучними нейронами у першій та другій відповідно) та гіперпараметрів (l1 = 10^{-7}, l2 = 10^{-7}, early stopping patience = 25, learning_rate = 0.0025, rlrop patience = 5, rlpop factor = 0.5, rlpop min_lr = 0.0001) для навчання нейронної мережі з використанням наборів даних CullPDB та CB513.

Створену нейронну мережу можна використовувати для прогнозування вторинної структури білка.

Зміст

Вступ

Білки — це великі органічні молекули, які утворені з амінокислот. Кожен білок має унікальну послідовність амінокислот, визначену генетичним кодом у ДНК. Білки відіграють важливу роль у біологічних процесах, таких як метаболічні реакції, регуляція шляхів, ріст, формування шкіри, волосся, клітин крові, м’язів і кісток.

Біологічні функції білка тісно пов’язані з його структурою. Існує два основні підходи до визначення структури білка: експериментальний і за допомогою машинного прогнозування. Експериментальні методи включають рентгенівську дифракцію та ядерний магнітний резонанс (ЯМР), які можуть бути дорогими, трудомісткими та навіть не застосовуватися до певних білків. Необхідною умовою застосування експериментальних методів є фізична доступність білка. Як альтернатива, машинне прогнозування структури білка може бути виконано для будь-якого білка на основі заданої послідовності амінокислот без необхідності фізичної доступності білка, наприклад, перед його штучним синтезом. Такий підхід є дешевшим і ефективнішим, хоча зазвичай менш точним, ніж експериментальні методи. Найефективнішими наразі є моделі машинного навчання на основі нейронних мереж.

Прогнозування вторинної структури білка є важливим етапом у передбаченні тривимірної структури вцілому, оскільки надає уявлення про наявність та розташування підструктур, які напряму впливають на функціонал білка.

1. Огляд літератури

1.1 Структура білка

Білки [1] утворені з амінокислот, які послідовно зв’язані між собою пептидними зв’язками. Існує 20 амінокислот, які зазвичай зустрічаються в природі. Амінокислота — це органічний композит, який містить аміногрупу (-NH2), карбоксильну групу (-COOH) і молекулу бічного ланцюга (R), зв’язану з асиметричним альфа-атомом вуглецю. Приклад структури амінокислоти наведено нижче на рисунку 1.1. Кожна амінокислота має різні фізичні та хімічні властивості, такі як електростатичний заряд, стан гідрофобності, коефіцієнт кислотного розкладання (pKa), розмір і функціональні групи, що і відіграють важливу роль у визначенні структури білка.

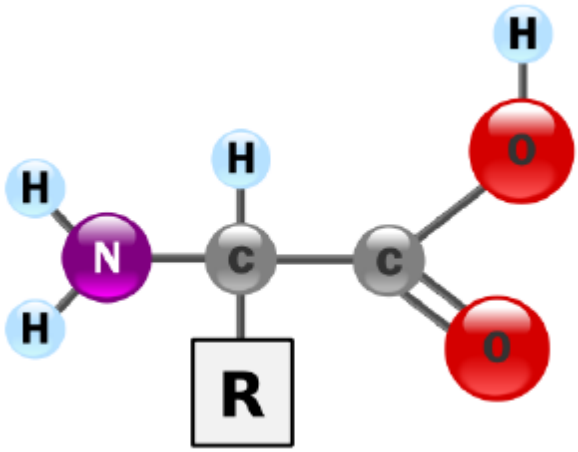

Рис 1.1 — Структура амінокислоти.

Структура білка має 4 основні рівні:

- Первинна — послідовність амінокислот;
- Вторинна — сформована повторюваними моделями водневих зв’язків;
- Третинна — визначає тривимірну структуру одного ланцюжка амінокислот;
- Четвертинна — тривимірна структура декількох ланцюжків амінокислот.

1.1.1 Первинна структура

Пептидні зв’язки, які виникають під час синтезу білка, змушують первинну структуру [1] залишатися разом. У живих істотах ген, який кодує білок, є тим, що визначає первинну структуру. Послідовність амінокислот унікальна для цього білка і визначає його тривимірну структуру та функцію. Послідовність амінокислот, які утворюють білок, можна виділити шляхом трансляції послідовності гена, який кодує білок. Існують також інші методи, які можна використовувати для визначення вмісту амінокислот у білках, такі як деградація Едмана та мас-спектрометрія (МС). На рисунку 1.2 показано первинну структуру білка.

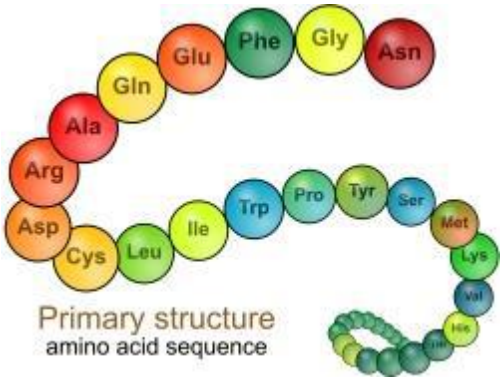

Рис. 1.2 — Первинна структура білка.

1.1.2 Вторинна структура

Вторинна структура [1] містить регулярні водневі зв’язки, утворені між сусідніми амінокислотами. Такі амінокислоти збираються разом, утворюючи сегменти вторинної структури. Існує два типи моделей водневих зв’язків: мотив обертання та мотив містка. У мотиві обертання, який також називають мотивом n-обертання, існує водневий зв’язок між амінокислотою в положенні i та амінокислотою в положенні $i+n$, де n зазвичай приймає значення, що дорівнює 3, 4 або 5. У мотиві містка, існують водневі зв’язки між амінокислотами, які можуть бути не близькими одна до одної. Вторинна структура формується, коли мотиви обертання та містку об’єднуються послідовним і специфічним чином. Наприклад, мотив обертання, який повторюється 4 рази, утворює альфа-спіралі, а мотиви містка, що повторюються, утворюють бета-листи, які можуть містити кілька сегментів бета-

смужки. З іншого боку, петлі зазвичай містять нерегулярні візерунки зв’язування. Третинну структуру білка можна зрозуміти як зібрані разом елементи вторинної структури. На рисунку 1.3 показано вторинну структуру білка.

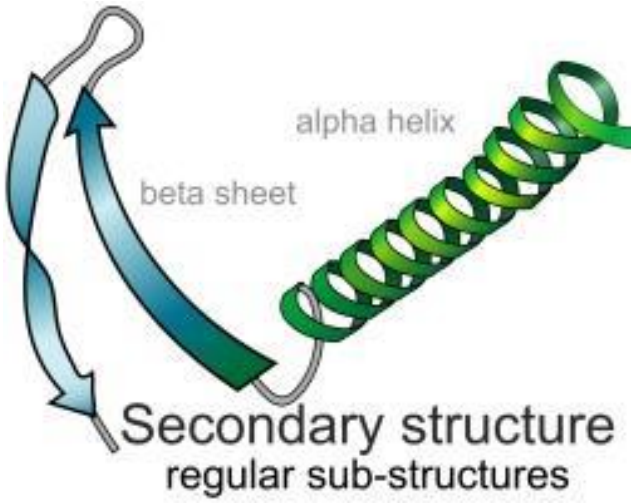

Рис. 1.3 — вторинна структура білка

1.1.2.1 Спіраль

У спіралях основа білка утворює спіральну структуру (рис. 1.4) . Існує три типи спіралей: альфа-спіраль (α-спіраль), 3_{10} спіраль і пі-спіраль (π-спіраль). Спіралі можуть виконувати різні функціональні ролі. Мотиви, які зв’язують ДНК (ланцюг-спіраль-ланцюг, лейцинова застібка-блискавка, цинковий палець) і структури, які проходять через клітинну мембрану (родопсини, рецептори, затиснуті G-білком), є одними з прикладів спіральних структур.

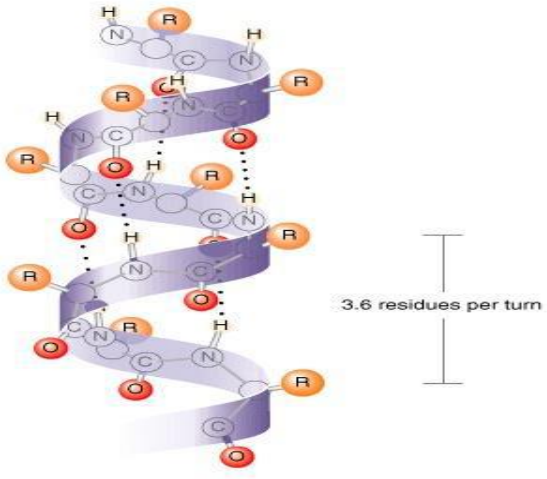

Рис. 1.4 — Альфа-спіраль.

1.1.2.2 Бета-смужка та бета-листок

Бета-листки утворюються бета-смужками, які взаємодіють попарно через водневі зв’язки (рис. 1.5).

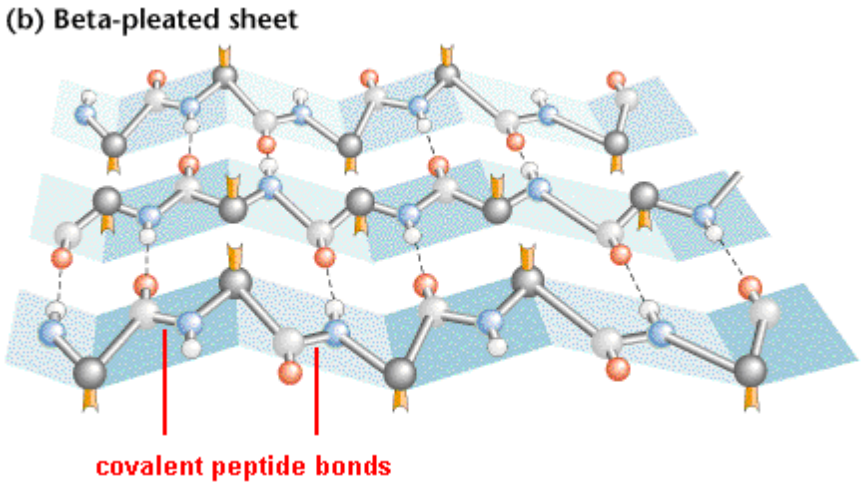

Рис. 1.5 — Бета-листок.

Бета-листок повинен містити принаймні дві бета-смужки, і кожна бета-смужка потребує принаймні 2 або 3 водневі зв’язки, які з’єднуються з сусідньою смужкою. Сегмент бета-смужки зазвичай має довжину від 3 до 10 амінокислот. Взаємодіючі амінокислоти в сегментах бета-смужки можуть бути як близько один до одного, так і далеко один від одного відповідно до послідовності амінокислот. Ті бета-смужки, які розташовані далеко одна від одної на основі одновимірної послідовності, можуть наблизитися, коли молекула білка згортається у свою тривимірну структуру. Агрегація білків і фібрили, які утворюються внаслідок злиття бета-листків, відіграють роль у різних захворюваннях, таких як хвороба Альцгеймера.

1.1.2.3 Петля

Петлі — це структури, які здебільшого присутні між спіралями та бета-листами, з різною довжиною та конфігурацією. Зазвичай вони розташовані на поверхні білків. Петлі не накладають сильних обмежень на вирівнювання вторинних структур, тому що в структурах петель може бути більше мутацій, ніж у спіралях

або бета-смужках. Петлі, як правило, мають заряджені та поляризовані амінокислоти і зазвичай знаходяться у функціонально більш активних областях. Є 3 типи петель: поворот, вигин і випадкове кільце.

1.2 Прогнозування вторинної структури білка

На сьогоднішній день проведено ряд досліджень, наприклад, [2, 3] щодо прогнозування структури білка, проте задача залишається невирішеною. Через проблеми безпосереднього пошуку найкращої тривимірної структури задачу було розділено на підзадачі. Спочатку цільовий білок порівнюється з білками в базі даних за допомогою різних алгоритмів вирівнювання, які можна використовувати для обчислення матриць статистичних профілів на основі підрахунку частоти появи амінокислот у певних позиціях. Ці матриці можна використовувати як вхідні характеристики для прогнозування певних властивостей структури білка, таких як вторинна структура, двогранні кути, доступність розчинника, невпорядковані області, контактні карти. На наступному етапі вибираються фрагментні структури для сегментів цільового білка, що перекриваються. Ці прогнозування та фрагменти надають обмеження та значно скорочують простір пошуку алгоритмів прогнозування 3D-структури.

```
MSNTTWGLQRDITPRLGARLVQEGNQLLA
LLLLEEEEELLLHHHHHHHLLLLLEEELL
```

Рис. 1.6 — 3-станове прогнозування вторинної структури. Перший рядок — послідовність амінокислот, другий рядок — мітки класу вторинної структури.

Прогнозування вторинної структури білка має на меті призначити мітку класу вторинної структури кожній амінокислоті білка (рис. 1.8). Існує два типи прогнозування: 3-станове та 8-станове. У 3-становому передбаченні наявні такі класи:

- спіраль (H),
- бета-смужка (E),
- петля (L).

У 8-становому передбаченні наявні наступні класи:

- альфа-спіраль (H, для 3-станового H),
- бета-смужка (E, для 3-станового E),
- петля (L, для 3-станового C),
- бета-поворот (T, для 3-станового C),
- вигин (S, для 3-станового C),
- 3_{10}-спіраль (G, для 3-станового H),
- бета-місток (B, для 3-станового E),
- π-спіраль (I, для 3-станового C).

Зазвичай для прогнозування вторинної структури використовуються методи машинного навчання з вчителем. З цією метою білки, вторинна структура яких відома, використовуються для навчання моделі, яка передбачає вторинну структуру білка.

1.3 Набори даних

Набори даних, призначені для створення моделей машинного навчання для прогнозування вторинної структури білка, зазвичай мають різний набір білків, але мають подібну форму ознак, а саме:

- Кожен запис складається з послідовності 700 амінокислот (якщо де-факто їх у білку менше, то залишок позицій заповнюється нулями)
- Кожна амінокислота представляє собою 42 ознаки (20 для мітки амінокислоти та 22 на інші ознаки, такі як надання переваги амінокислоти одному типу вторинної підструктури іншому тощо).

1.3.1 CullPDB та ASTRAL

Набір CullPDB був створений у 2005 році спеціально для задач з прогнозування вторинної та третинної структури білків, складався з 12228 білків, як приклади для навчання машинних моделей, і, при цьому, білки мали схожість не більше аніж 25%. На ICML 2014 (International Conference on Machine Learning,

Інтернаціональна конференція з машинного навчання, 2014 рік) [5] було представлено оновлену, доповнену та дещо зменшену версію набору CullPDB. Наразі набір включає в себе 6133 білка зі схожістю менше аніж 20%.

Набір ASTRAL [6] створений у 2014 році у спробі замінити набір CullPDB. Включає в себе 6892 білка зі схожістю близько 25%.

1.3.2 CB513

Для перевірки точності роботи моделі машинного навчання для прогнозування вторинної структури білка використовувався набір даних CB513, оскільки, наразі, він є найпопулярнішим та найпоказовішим набором для тестування створеної моделі [7]. Цей набір був запропонований на ICML 2014 та, по суті, являє собою вибіркою певних білків з набору CullPDB зі схожістю менше аніж 25%.

1.3.3 CASP

CASP (Critical Assessment of protein Structure Prediction, критична оцінка прогнозування структури білка) — масштабний проект що проводиться кожні два роки, починаючи з 1994 року та ставить на меті об’єктивну оцінку існуючих методів прогнозування структури білка. Для цього кожного разу створюється набір даних, що складається зі спеціально підібраних білків для проведення оцінки наявних методів прогнозування. Найновішим набором цього проекту наразі є так званий CASP15 [8], але, зазвичай, використовуються і більш старі набори такі як CASP9, CASP10, CASP11 тощо.

1.4 Огляд моделі машинного навчання у літературі

У статті [2] наведено модель машинного навчання, а саме рекурентна нейронна мережа. Архітектуру моделі показано на рисунку 1.7. Ця нейронна мережа навчалася на наборі даних CullPDB та створювалася спеціально для отримання покращених результатів точності 8-станового прогнозування вторинної структури білка, аніж у роботі [3] (точність 67,4%). На тестовому наборі даних CB513 вона показує точність прогнозування близько 68,7%.

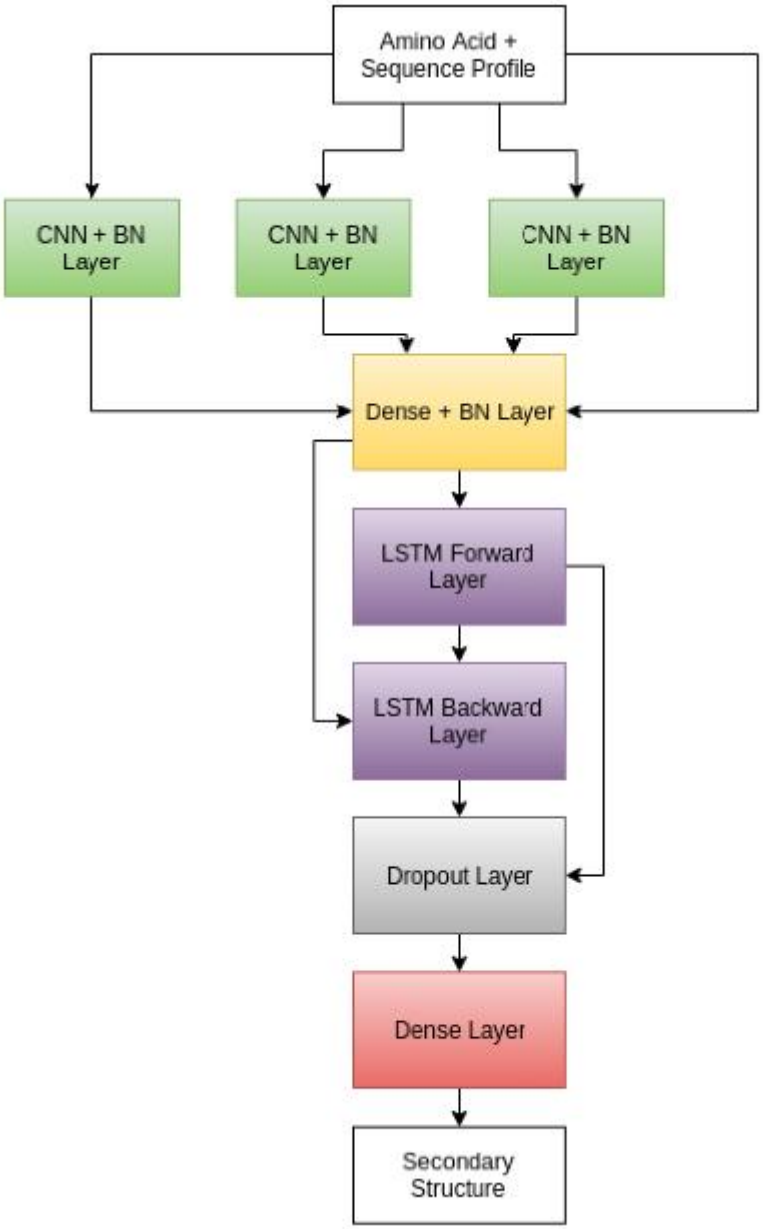

Рис 1.7 — Архітектура нейронної мережі, представленої у статті [2].

Особливостями саме цієї нейронної мережі є наявність шарів згортки, які працюють паралельно та двонаправлена рекурентна частина мережі на основі рекурентних шарів типу LSTM з вертикальними з’єднаннями. Задля покращення точності роботи можна спробувати замінити рекурентні шари типу LSTM, на якісь інші, наприклад GRU.

1.5 Рекурентні шари

1.5.1 Long Short-Term Memory

Long Short-Term Memory (LSTM, довгострокова короткочасна пам’ять) [4] – це особливий вид рекурентних нейронних мереж, здатний вивчати довгострокові послідовності. Вперше були описані Шмідхубером і Хохрайтером у 1997 році та спеціально розроблені для уникнення довгострокових проблем залежності.

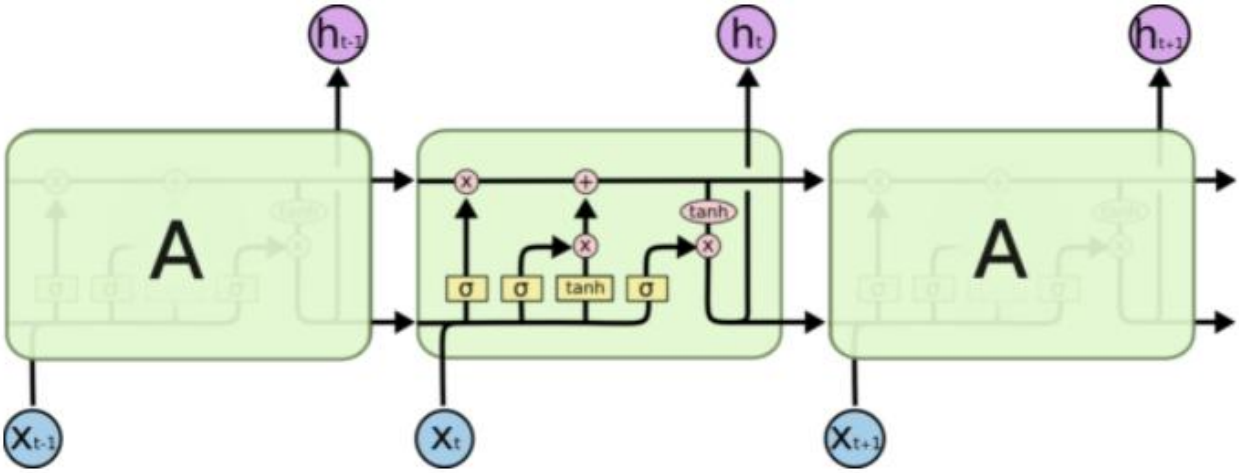

Рис. 1.8 — Схема роботи LSTM

У комірках звичайних рекурентних нейронних мереж вхідні дані про позначення часу та прихований стан із попереднього кроку передаються через рівень активації для отримання нового стану. У той час як у LSTM процес дещо складніший: кожного разу комірка приймає вхідні дані з трьох різних станів, таких як поточний стан введення, короткочасна пам’ять із попередньої комірки та довгострокова пам’ять.

Ці комірки використовують так звані вентилі для регулювання інформації, яку потрібно зберігати або відкидати під час роботи циклу перед передачею довгострокової та короткострокової інформації до наступної комірки. Загалом існує три вентилі:

- Вхідний вентиль: вирішує, яка інформація буде зберігатися в довгостроковій пам'яті. Він працює лише з інформацією з поточного

введення та короткочасної пам’яті з попереднього кроку. На цьому вентилі фільтрується інформація, яка не є корисною.

- Вентиль забування: вирішує, яку інформацію з довгострокової пам’яті зберігати чи відкидати, що робиться шляхом множення вхідної довгострокової пам’яті на так званий вектор забування, створений поточними вхідними даними та короткочасною пам’яттю.
- Вихідний вентиль: створює нову короткочасну пам’ять на основі поточних вхідних даних, попередньої короткочасної пам’яті та нещодавно обчисленої довгострокової пам’яті, яка буде передана на наступному часовому кроці. Також з цього вентилю можна отримати вихідні дані поточного часового стану

1.5.2 Gated Recurrent Unit

Gated Recurrent Unit (GRU, вентильна рекурентна одиниця) [4] має подібний принцип роботи до звичайної рекурентної нейронної мережі, але різниця полягає в наявності у GRU вентилів. Щоб вирішити проблему, з якою стикається стандартний RNN, GRU включає два механізми роботи вентиля, які називаються вентилем оновлення та вентилем скидання.

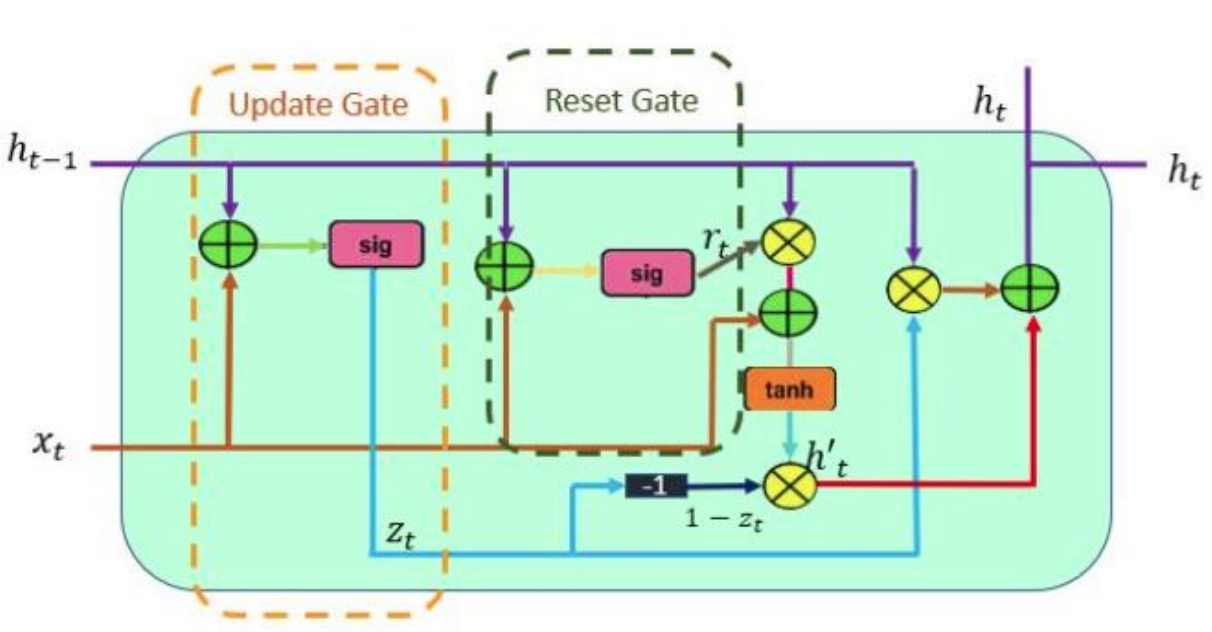

Рис. 1.9 — Схема роботи GRU

- Вентиль оновлення: відповідає за визначення обсягу попередньої інформації, яку потрібно передати в наступний стан. Це дуже потужна здібність, оскільки модель може вирішити скопіювати всю інформацію з минулого та усунути ризик зникнення градієнта.
- Вентиль скидання: вирішує, якою частиною минулої інформації потрібно знехтувати.

Спочатку активується вентиль оновлення, що зберігає відповідну інформацію з минулого кроку в новому вмісті пам’яті. Потім множиться вхідний вектор і прихований стан на їхні ваги. Далі він обчислює поелементне множення між воротами скидання та попередньо прихованим кратним станом. Після підсумовування вищезазначених кроків застосовується нелінійна функція активації та генерується наступна послідовність.

1.5.3 Порівняння LSTM та GRU

Основні відмінності LSTM:

- Має три вентилі
- Потребує більше пам’яті під час навчання
- Потребує більше часу для навчання
- Краще працює з великими наборами даних

Основні відмінності GRU:

- Має 2 вентилі
- Потребує менше пам’яті під час навчання
- Потребує менше часу для навчання
- Краще працює з невеликими наборами даних

1.6 Постановка задач дослідження

Метою дипломної роботи є вдосконалення представленої у літературі моделі нейронної мережі для 8-станового прогнозування вторинної структури білка.

Для досягнення сформульованої мети були поставлені такі задачі:

1. Модифікація моделі нейронної мережі [2] шляхом заміни шарів LSTM на GRU.
2. Застосування різних методів регуляризації нейронної мережі (рання зупинка, L_1- і L_2-регуляризації, шари Dropout) для підвищення її узагальнюючої здібності.
3. Підбір архітектури нейронної мережі та гіперпараметрів її навчання для максимізації точності її роботи на перевірочній вибірці набору даних CullPDB.
4. Оцінювання побудованої моделі нейронної мережі на незалежному наборі даних CB513 і порівняння отриманих результатів з наведеними у літературі.

2. Методика дослідження

2.1 Середовище розробки

2.1.1 Google Colab

Google Colaboratory [9] — це безкоштовний хмарний сервіс, створений компанією Google та в якому наявно все, що потрібно для машинного навчання. Він працює на базі інтерактивного інструменту розробки Jupyter Notebook та використовує мову програмування Python. Основною перевагою сервісу є безкоштовний доступ до потужних GPU для обробки даних (наприклад, Nvidia Tesla T4 з 2880 ядрами, графічною пам'яттю 16 GB і швидкодією 65 терафлопс)). Окрім того, на віртуальній машині, що видається у користування (Intel Xeon 2.2 GHz, RAM ~16 GB, дискова пам'ять ~80 GB), вже одразу встановлені найпопулярніші бібліотеки машинного навчання та інші бібліотеки мови Python.

2.1.2 Бібліотеки навчання

Для створення машинної моделі з прогнозування вторинної структури білка було обрано бібліотеку Keras [10], що є частиною відкритої платформи з машинного навчання Tensorflow [11]. Вона надає великі можливості для створення нейронних мереж, маючи реалізації різних алгоритмів навчання, прихованих шарів, обрахунку метрик якості тощо. До основних переваг бібліотеки можна віднести можливість навчання моделей з використанням GPU, що значно пришвидшує навчання, та гнучкі способи створення моделі машинного навчання та можливість створення нейронної мережі пошарово, при цьому маючи можливість гнучко налаштувати кожен шар окремо.

2.2 Підготовка даних

Для навчання нейронної мережі було обрано набір даних CullPDB оскільки він включає в себе не лише білки людини та білки, що використовувалися для його творення, мають нижчу схожість між собою, аніж білки, наявні у інших наборах даних.

Для тестування точності роботи нейронної мережі було обрано набір даних CB513 через наявність більш різноманітних білків у порівнянні, наприклад, з наборами даних CASP, через що результати точності роботи нейронної мережі є більш показовими.

При роботі з цими наборами даних важливо правильно видобути ознаки та окремо розділити набір CullPDB на навчальну та валідаційну вибірки, як зазначено у пояснювальній записці [12].

2.3 Методи регуляризації

Оскільки набір даних має відносно невеликий розмір, нейронна мережа може швидко почати перенавчатися. Для зменшення ефекту перенавчання було використано декілька методів регуляризації.

2.3.1 Метод ранньої зупинки

Метод ранньої зупинки [13] було обрано оскільки він є найпростішим методом регуляризації. Він дає змогу зупинити навчання нейронної мережі, якщо почалося перенавчання, що також дозволяє зекономити на часі навчання. Слідкування за тим, чи не почалося перенавчання робилося на основі зміни точності роботи нейронної мережі при перевірці на валідаційній вибірці, яку було виділено з набору даних CullPDB.

2.3.2 L1- та L2-регуляризації

Методи L1- та L2-регуляризації [14] було обрано оскільки вони дозволяють зменшити ефект перенавчання під час усього навчання нейронної мережі. Ці методи є дуже корисними, як і в нашому випадку, під час використання наборів даних з невеликою кількістю прикладів, оскільки алгоритм навчання не так ефективно підлаштовує внутрішні параметри мережі під цей набір, що дає змогу отримувати кращі результати точності на тестових вибірках.

2.3.3 Dropout

Прихований шар типу Dropout [15] випадково встановлює вхідний сигнал шару в 0 із імовірністю, яка визначається параметром rate на кожному кроці під час навчання. Вхідні сигнали, для яких не встановлено значення 0, масштабуються на 1/(1 - rate), щоб сума всіх вхідних сигналів не змінювалася.

2.3.4 Зменшення параметра швидкості навчання на плато

Метод зменшення параметра швидкості навчання на плато [16] працює під час всього навчання і, як можна зрозуміти з назви, зменшує значення параметра швидкості навчання, якщо значення показника якості (функції втрат або певної метрики якості) на перевірочній вибірці перестало покращуватися. Моделі часто покращуються від зниження параметра швидкості навчання в 2-10 разів, коли навчання стагнує (рис. 2.1).

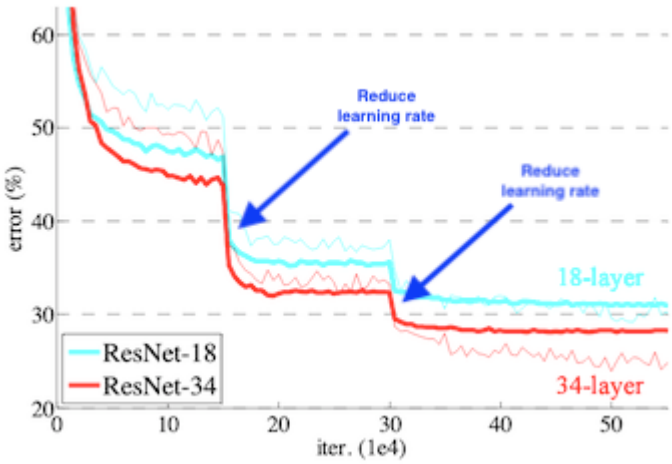

Рис. 2.1 — Приклад покращення моделей після зменшення параметра швидкості навчання.

2.4 Створення та навчання мережі

Для дослідження точності роботи нейронної мережі виконувалися наступні варіанти змін двонаправленої рекурентної частини:

- Зміна кількості штучних нейронів у двонаправленій рекурентній мережі.
- Зміна кількості модулей двонаправлених рекурентних мереж.

Щодо прихованих шарів типу Dense, то виконувалися варіанти змін:

- кількості штучних нейронів, окремо для кожного шару;
- кількості прихованих шарів;
- положення прихованих шарів відносно двонаправленої рекурентної частини (перед нею, чи після неї).

Параметри методів регуляризції змінювалися наступним чином:

- Метод ранньої зупинки: зміна значення параметра patience (терпіння).
- L1- та L2-регуляризації: зміна значення параметрів l1 та l2, але враховуючи що зазвичай ефективніше, коли l2 > l1.
- Dropout: зміна кількості й положення цих шарів, зміна значення параметра rate для кожного окремо для кожного шару.
- Зменшення параметра швидкості навчання на плато: зміна значення параметрів patience, factor та min_lr.

Для згортових шарів виконувалися зміни:

- кількості згортових шарів;
- розмірів фільтру (різний для кожного шару);
- кількоств фільтрів у кожному шарі.

Для параметрів навчання були варіанти змін:

- алгоритму навчання (RMSprop, Adam);
- значень/початкових значень параметра швидкості навчання;
- кількості епох навчання (було обрано 1000, оскільки метод ранньої зупинки повинен закінчувати навчання раніше).
- розміру пакету (batch size; було обрано рівним 2).

3 Результати дослідження

3.1 Результати дослідження

У результаті проведення дослідження було знайдено оптимальні архітектуру, набір параметрів навчання та параметрів регуляризації. Створена модель має точність роботи **69.4%.**

Архітектуру створеної мережі наведено на рисунку 3.1.

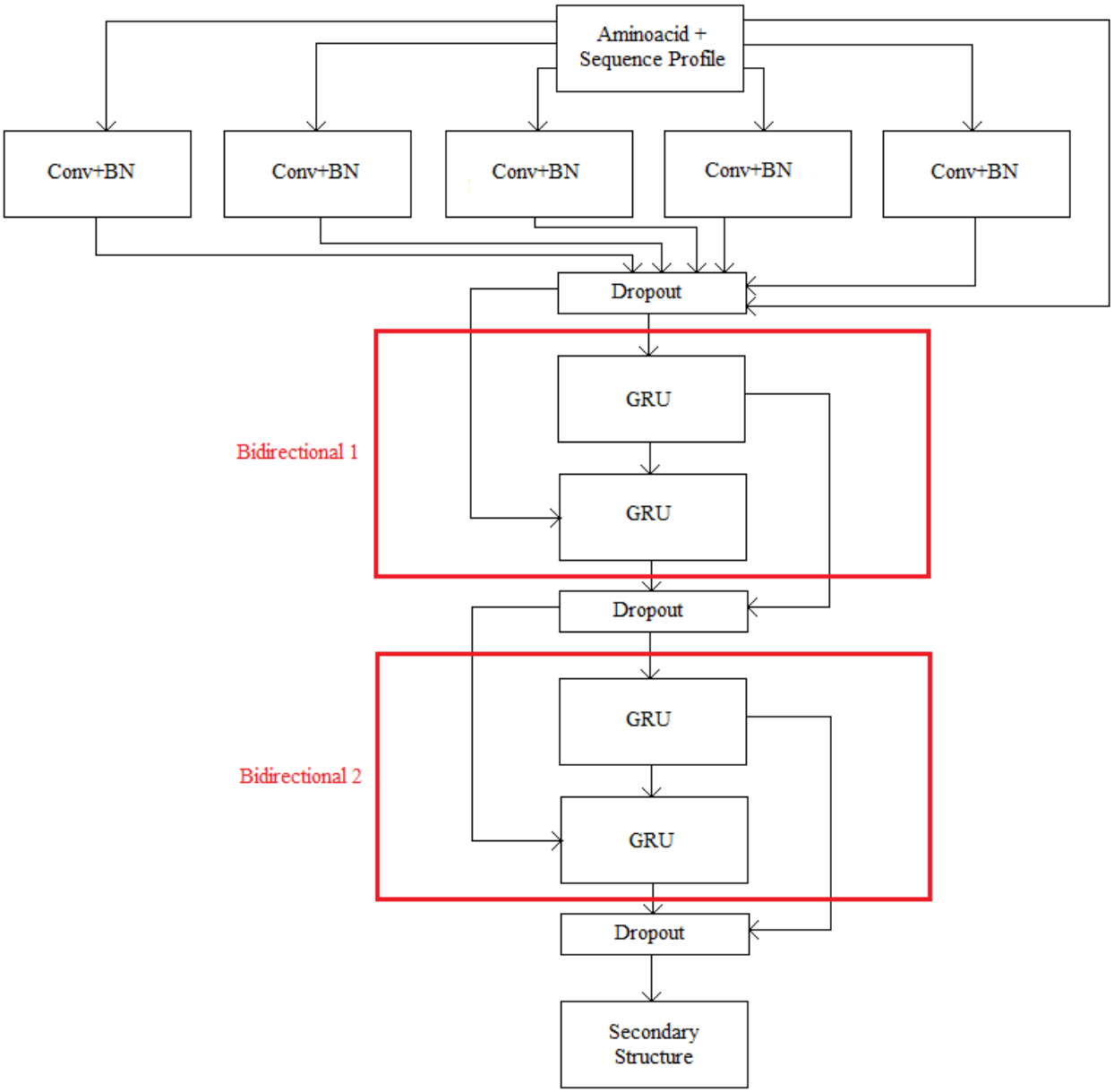

Рис. 3.1 — Архітектура створеної нейронної мережі.

У таблиці 3.1 наведено значення гіперпараметрів навчання та елементів створеної нейронної мережі.

Таблиця 3.1 — Значення гіперпараметрів навчання створеної мережі

Назва	Параметри
Згорткові шари	1. Кількість фільтрів: 64 Розмір фільтрів: 3 2. Кількість фільтрів: 64 Розмір фільтрів: 5 3. Кількість фільтрів: 64 Розмір фільтрів: 7 4. Кількість фільтрів: 64 Розмір фільтрів: 9 5. Кількість фільтрів: 64 Розмір фільтрів: 11
Рекурентні шари GRU	Кількість штучних нейронів у кожному шарі GRU для: Bidirectional 1: 500 Bidirectional 2: 250
Метод ранньої зупинки	Значення параметра patience: 25
L1- та L2-регуляризації	Значення параметра l1: 10^{-7} Значення параметра l2: 10^{-5}
Dropout	Значення параметра rate: 0.5
Зменшення параметра швидкості навчання на плато	Значення параметра patience: 5 Значення параметра factor: 0.5 Значення параметра min_lr: 0.0001
Алгоритм навчання	RMSprop Значення параметра lr: 0.0025

3.2 Аналіз результатів

Точність роботи, що рівна 69.4%, створеної нейронної мережі на 0.7% вища, аніж точність роботи мережі, яка наведена у статті [2]. Це невеликий приріст точності, але це означає що використання рекурентних мереж на основі GRU для прогнозування вторинної структури білка є більш перспективним, ніж використання рекурентних мереж на основі LSTM.

Шари згортки є важливою частиною нейронної мережі для виконання задачі прогнозування вторинної структури білка. Як видно з рисунку 3.2, оптимальним рішенням було використання саме п’яти шарів згортки.

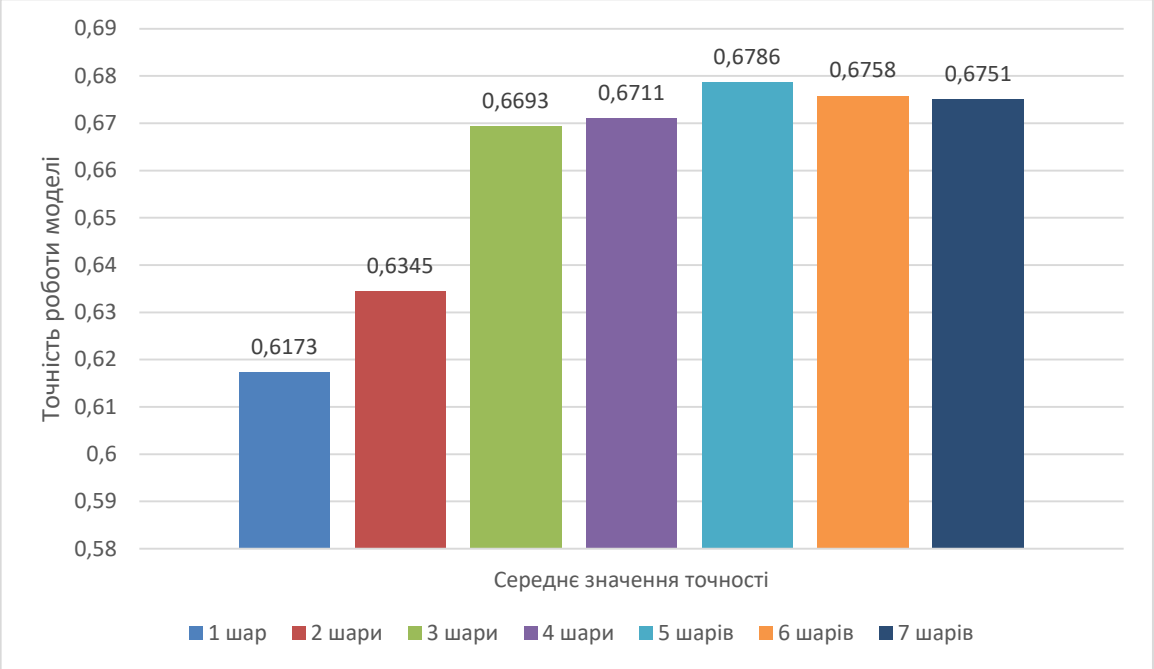

Рис. 3.2 — Діаграма залежності середньої точності роботи нейронної мережі від кількості шарів згортки.

З діаграми на рисунку 3.3 видно, оптимальною кількістю фільтрів для шарів згортки є 64 фільтри.

Порівнюючи архітектуру мережі зі статті [2] та створеної мережі на рисунках 2.1 та 3.1 відповідно, можна помітити, що було повністю прибрано приховані шари типу Dense, оскільки вони виявилися найбільш схильними до перенавчання.

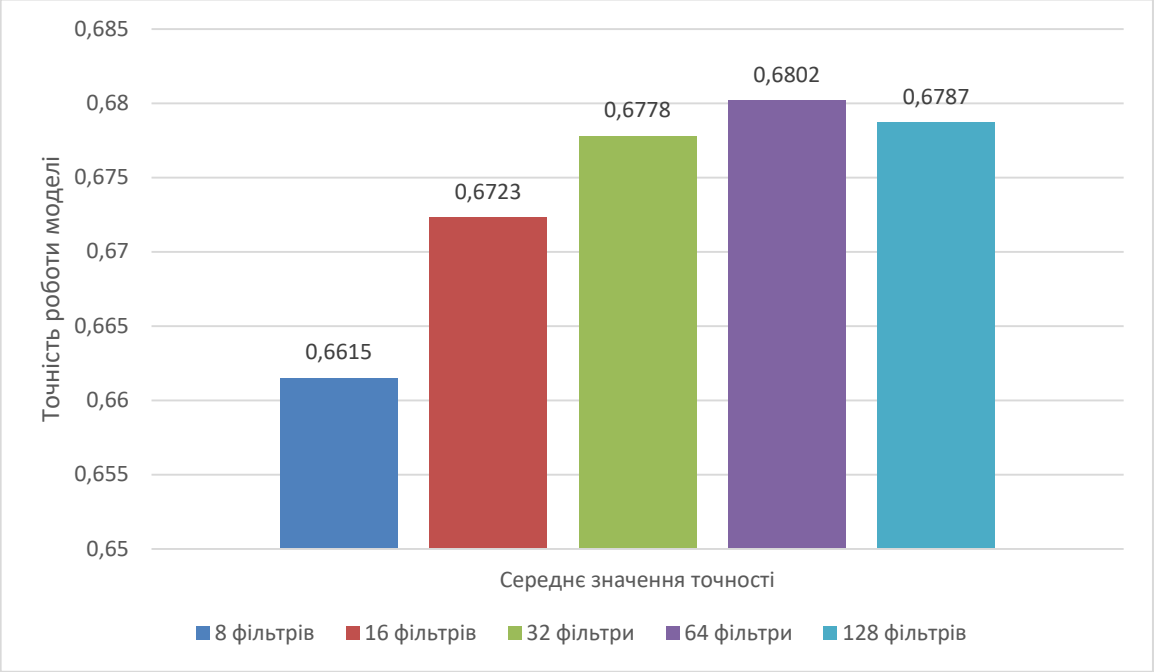

Рис. 3.3 — Діаграма залежності середньої точності роботи нейронної мережі від кількості фільтрів у шарах згортки.

Як можна побачити на рисунку 3.4, два двонаправлені рекурентні шари на основі GRU є оптимальним рішенням, оскільки при збільшенні їх кількості точність роботи мережі хоч і може зростати, але не на відносно великі значення, при цьому час навчання класифікатора значно збільшується.

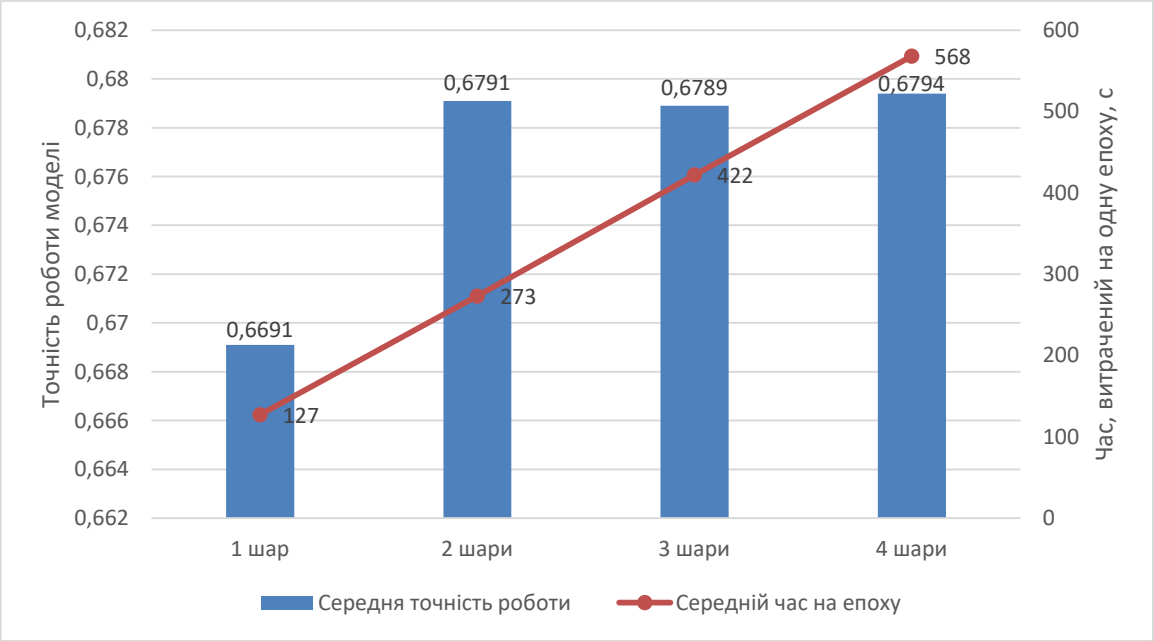

Рис. 3.4 — Діаграма точності роботи та часу навчання за одну епоху мережі від кількості двонаправлених рекурентних шарів на основі GRU.

На рисунку 3.5 зображено порівняльну діаграму точності роботи класифікатора, створеного в цій роботі та зі статей [2, 3].

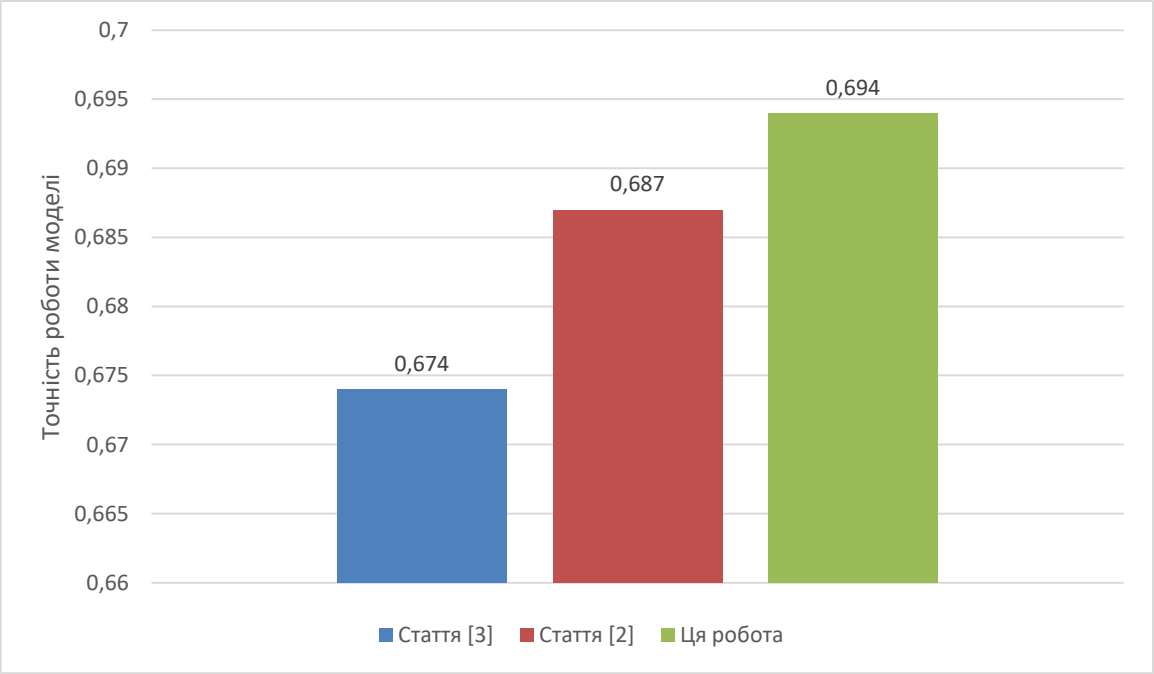

Рис. 3.5 — Діаграма точності роботи моделей в цій роботі та зі статей [2, 3].

Висновки

1. Проведені дослідження щодо ефективності роботи нейронної мережі при заміні рекурентних шарів LSTM на GRU.
2. Досліджена ефективність роботи нейронної мережі із застосуванням різних методів регуляризації та виявлено оптимальні параметри для методу ранньої зупинки, методів L1- та L2-регуляризації, шарів типу Dropout та методу зменшення параметра швидкості навчання на плато.
3. Знайдено оптимальну архітектуру (дві двонаправлені рекурентні нейронні мережі на основі GRU з 500 та 250 штучними нейронами у першій та другій відповідно) та значення гіперпараметрів.
4. Проведено оцінювання побудованої моделі нейронної мережі на незалежному наборі даних CB513, яке показало точність роботи створеної мережі рівною 69.4%, що на 0.7% більше аніж у статті [2] та на 2% більше аніж у статті [3].

Перелік джерел посилання

1. Ulutaş A. E. Protein fragment selection using machine learning / Ulutaş Alperen Emre – Кайсер, 2018. – 85 с.

2. Johansen A. R. et al. Protein and secondary structure prediction with convolutions and vertical-bi-directional RNNs / A. R. Johansen, S. K. Sønderby, O. Winther. – Copenhagen, 2016. – 5 с.

3. Zhou J. and Troyanskaya O. G., “Deep supervised and convolutional generative stochastic network for protein secondary structure prediction,” / Zhou J. and Troyanskaya O. G. – arXiv preprint arXiv:1403.1347, 2014.

4. LSTM Vs GRU in Recurrent Neural Network: A Comparative Study [Електронний ресурс] // Analytics India Magazine. – 2021. – URL: https://analyticsindiamag.com/lstm-vs-gru-in-recurrent-neural-network-a-comparative-study/. (дата звернення: 14.04.2023)

5. International Conference on Machine Learning [Електронний ресурс] // ICML, Beijin – 2014. – URL: https://icml.cc/2014/ (дата звернення: 14.04.2023)

6. Fox N. K. et al., Brenner SE, Chandonia JM. 2013. “SCOPe: Structural Classification of Proteins — extended, integrating SCOP and ASTRAL data and classification of new structures.” / Fox N. K., Brenner S. E., Chandonia J.M // Nuclear Acids Research – 2013. – №10

7. Protein Secondary Structure Prediction on CB513 [Електронний ресурс] // Papers with Code. – 2023. – URL: https://paperswithcode.com/sota/protein-secondary-structure-prediction-on-1. (дата звернення: 14.04.2023)

8. Success Stories From Recent CASPs [Електронний ресурс] // Protein Structure Prediction Center. – 2022. – URL: https://predictioncenter.org/index.cgi. (дата звернення: 14.04.2023)

9. Google Colaboratory [Електронний ресурс] – URL: https://colab.research.google.com (дата звернення: 14.04.2023)

10. Keras [Електронний ресурс] – URL: https://keras.io/ (дата звернення: 14.04.2023)

11. TensorFlow [Електронний ресурс] – URL: https://www.tensorflow.org/ (дата звернення: 14.04.2023)

12. Dataset_readme [Електронний ресурс] // ICML2014 – 2014. – URL: https://www.princeton.edu/~jzthree/datasets/ICML2014/dataset_readme.txt (дата звернення: 22.02.2023)

13. EarlyStopping [Електронний ресурс] // Keras – URL: https://keras.io/api/callbacks/early_stopping/ (дата звернення: 14.04.2023)

14. Layer weight regularizers [Електронний ресурс] // Keras – URL: https://keras.io/api/layers/regularizers/ (дата звернення: 14.04.2023)

15. Dropout layer [Електронний ресурс] // Keras – URL: https://keras.io/api/layers/regularization_layers/dropout/ (дата звернення: 14.04.2023)

16. ReduceLROnPlateau [Електронний ресурс] // Keras – URL: https://keras.io/api/callbacks/reduce_lr_on_plateau/ (дата звернення: 14.04.2023)

Додаток А

Код програми

```
from google.colab import drive
drive.mount('/content/drive')
!mkdir /content/data
!cp /content/drive/MyDrive/PSSP/data/cb513+profile_split1.npy.gz /content/data/c
b513+profile_split1.npy.gz
!cp /content/drive/MyDrive/PSSP/data/cullpdb+profile_6133_filtered.npy.gz /conte
nt/data/cullpdb+profile_6133_filtered.npy.gz


from tensorflow import keras
import tensorflow as tf
import numpy as np
import sklearn
from keras.callbacks import EarlyStopping
from keras.callbacks import ModelCheckpoint
from sklearn.metrics import classification_report
from datetime import datetime
import pickle
from keras.api._v2.keras import callbacks
import matplotlib.pyplot as plt


import gzip

def opener(filename):
    f = open(filename,'rb')
    if (f.read(2) == '\x1f\x8b'):
        f.seek(0)
        return gzip.GzipFile(fileobj=f)
    else:
        f.seek(0)
        return f


TRAIN_PATH = '/content/data/cullpdb+profile_6133_filtered.npy.gz'
TEST_PATH = '/content/data/cb513+profile_split1.npy.gz'


floatX = "float32"


def get_train():
  print("Loading train data ...")
  X_in = np.load(opener(TRAIN_PATH))
  X = np.reshape(X_in,(5534,700,57))
  del X_in
```

```
    labels = X[:,:,22:30]
    mask = X[:,:,30] * -1 + 1

    a = np.arange(0,21)
    b = np.arange(35,56)
    c = np.hstack((a,b))
    X = X[:,:,c]

    num_seqs = np.size(X,0)
    seqlen = np.size(X,1)
    d = np.size(X,2)
    num_classes = 8

    X = X.astype(floatX)
    mask = mask.astype(floatX)

    seq_names = np.arange(0,num_seqs)

    X_train = X[seq_names[0:5278]]
    X_valid = X[seq_names[5278:5534]]
    labels_train = labels[seq_names[0:5278]]
    labels_valid = labels[seq_names[5278:5534]]
    mask_train = mask[seq_names[0:5278]]
    mask_valid = mask[seq_names[5278:5534]]
    num_seq_train = np.size(X_train,0)
    num_seq_valid = np.size(X_valid,0)
    return X_train, X_valid, labels_train, labels_valid, mask_train, \
      mask_valid, num_seq_train


def get_test():
    print("Loading test data ...")
    X_test_in = np.load(opener(TEST_PATH))
    X_test = np.reshape(X_test_in,(514,700,57))
    del X_test_in
    X_test = X_test[:,:,:].astype(floatX)
    labels_test = X_test[:,:,22:30].astype('int32')
    mask_test = X_test[:,:,30].astype(floatX) * -1 + 1

    a = np.arange(0,21)
    b = np.arange(35,56)
    c = np.hstack((a,b))
    X_test = X_test[:,:,c]

    # getting meta
    seqlen = np.size(X_test,1)
    d = np.size(X_test,2)
    e = np.size(labels_test,2)
    num_classes = 8
    num_seq_test = np.size(X_test,0)
    del a, b, c
```

```
  X_add = np.zeros((126,seqlen,d))
  label_add = np.zeros((126,seqlen,e))
  mask_add = np.zeros((126,seqlen))

  X_test = np.concatenate((X_test,X_add), axis=0).astype(floatX)
  labels_test = np.concatenate((labels_test, label_add), axis=0).astype('int32')
  mask_test = np.concatenate((mask_test, mask_add), axis=0).astype(floatX)
  return X_test, mask_test, labels_test, num_seq_test


X_train, X_valid, labels_train, labels_valid, mask_train, mask_valid, num_seq_tr
ain = get_train()
X_test, mask_test, labels_test, num_seq_test = get_test()


batch_size = 2
N_CONV_A = N_CONV_B = N_CONV_C = 64
F_CONV_A = 3
F_CONV_B = 5
F_CONV_C = 7
F_CONV_D = 9
F_CONV_E = 11
N_L1 = N_L2 = 200
N_LSTM_F = N_LSTM_B = 500
n_inputs = 42
num_classes = 8
seq_len = 700
optimizer = keras.optimizers.RMSprop(learning_rate=0.0025)
reg = keras.regularizers.l1_l2(0.0000001, 0.00001)


## Start


input = keras.layers.Input(shape=(seq_len, n_inputs), batch_size=batch_size)


conv_a = keras.layers.Conv1D(N_CONV_A, F_CONV_A, padding='same', strides=1, acti
vation='relu')(input)
conv_a_b = keras.layers.BatchNormalization()(conv_a)

conv_b = keras.layers.Conv1D(N_CONV_B, F_CONV_B, padding='same', strides=1, acti
vation='relu')(input)
conv_b_b = keras.layers.BatchNormalization()(conv_b)

conv_c = keras.layers.Conv1D(N_CONV_C, F_CONV_C, padding='same', strides=1, acti
vation='relu')(input)
```

```
conv_c_b = keras.layers.BatchNormalization()(conv_c)

conv_d = keras.layers.Conv1D(N_CONV_C, F_CONV_D, padding='same', strides=1, acti
vation='relu')(input)
conv_d_b = keras.layers.BatchNormalization()(conv_d)

conv_e = keras.layers.Conv1D(N_CONV_C, F_CONV_E, padding='same', strides=1, acti
vation='relu')(input)
conv_e_b = keras.layers.BatchNormalization()(conv_e)


concats = keras.layers.Concatenate(axis=-
1)([conv_a_b, conv_b_b, conv_c_b, conv_d_b, conv_e_b
                                                ])

concats_all = keras.layers.Concatenate(axis=2)([input, concats])

concats_r = tf.reshape(concats_all,(batch_size,seq_len,n_inputs+N_CONV_A*5))

d_out_dense1 = keras.layers.Dropout(0.5)(concats_r)



forward_layer = keras.layers.GRU(N_LSTM_F, return_sequences=True, kernel_regular
izer=reg)(d_out_dense1)
f_n_d = keras.layers.Concatenate(axis=2)([d_out_dense1, forward_layer])

bigru = keras.layers.GRU(N_LSTM_B, return_sequences=True, go_backwards=True, ker
nel_regularizer=reg)(f_n_d)

forw_bigru = keras.layers.Concatenate(axis=2)([bigru, forward_layer])




d_out_bigru = keras.layers.Dropout(rate=0.5)(forw_bigru)

forward_layer_2 = keras.layers.GRU(int(N_LSTM_F/2), return_sequences=True, kerne
l_regularizer=reg)(d_out_bigru)
f_n_d_2 = keras.layers.Concatenate(axis=2)([d_out_bigru, forward_layer_2])

bigru_2 = keras.layers.GRU(int(N_LSTM_B/2), return_sequences=True, go_backwards=
True, kernel_regularizer=reg)(f_n_d_2)


forw_bigru_2 = keras.layers.Concatenate(axis=2)([bigru_2, forward_layer_2])



bigru_3_r = tf.reshape(forw_bigru_2,(batch_size*seq_len, int((N_LSTM_F+N_LSTM_B)
/2)))
```

```
d_out_bigru_3 = keras.layers.Dropout(rate=0.5)(bigru_3_r)


output = keras.layers.Dense(num_classes, activation='softmax')(d_out_bigru_3)

output_r = tf.reshape(output,(batch_size, seq_len, num_classes))


model = keras.Model(inputs=input, outputs=output_r)


model.compile(
              loss='categorical_crossentropy',
              optimizer=optimizer,
              weighted_metrics=['accuracy']
              )


es = EarlyStopping(monitor='val_accuracy', mode='max',
                   verbose=1, patience=25, restore_best_weights=True)
rlrop = keras.callbacks.ReduceLROnPlateau(monitor='val_accuracy', mode='max', fa
ctor=0.5,
                                patience=5, min_lr=0.00001)

history = model.fit(X_train, labels_train, batch_size=batch_size, sample_weight=
mask_train,
                    epochs=1000, verbose=1, callbacks=[es, rlrop],
                    validation_data=(X_valid, labels_valid, mask_valid)
                    )
```