Київський національний університет імені Тараса Шевченка

Фізичний факультет

Кафедра загальної фізики

**Звіт**

**про науково-виробничу практику**

**з фізики наноматеріалів**

студента 1 курсу магістратури

Завгороднього Олексія Володимировича

Науковий керівник

доктор фіз.-мат. Наук, доцент

ОЛІХ О.Я

кафедра загальної фізики

КИЇВ-2021

**Програма**

**науково-виробничої практики студента 1 курсу магістратури**

**групи "Фізика наносистем" Завгороднього Олексія Володимировича**

***Тема магістерської роботи:* "Застосування нейромережі для визначення концентрації домішкового заліза в кремнієвих сонячних елементах"**

|  |  |  |
| --- | --- | --- |
| № | Вид роботи | Термін виконання |
| 1. | Ознайомлення з методами налаштування нейронної мережі.  1) Функції активації.  2)Ініціалізація значень вагових коефіцієнтів.  3) Типи регуляризації (регуляризатор Dropout).  4) Оптимізатори для нейронної мережі.  5) Метод перехресної перевірки як оцінка роботи оцінювача. | 01.02.2021  -  28.02.2021 |
| 2. | Експериментальна частина.  1) Дослідження впливу кількості нейронних шарів та кількості нейронів в них.  2) Дослідження впливу функції активації та кількості епох.  3) Дослідження впливу розміру блоків навчання.  4) Дослідження впливу ініціалізаторів.  5) Дослідження впливу кількості епох та нормалізації даних.  6) Дослідження впливу оптимізатора та коефіцієнта навчання.  7) Дослідження впливу регуляризації | 1.03.2021  -  10.05.2021 |

Керівник науково-виробничої практики Оліх О.Я

**1. Ознайомлення з методами налаштування нейронної мережі**

**1.1 Функції активації**

1) Функція активації – Сигмоїда

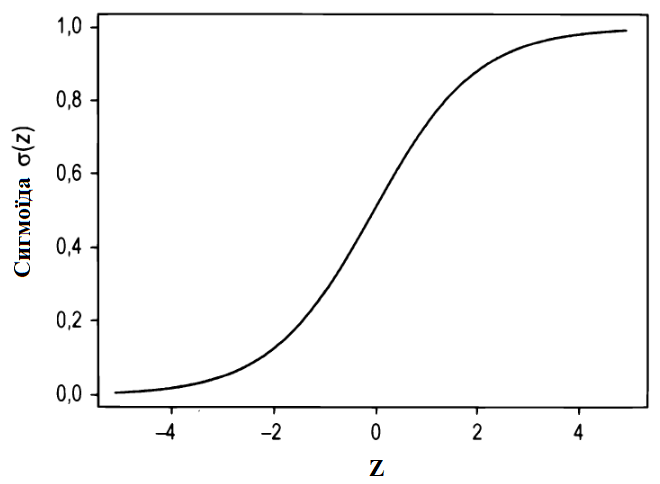


Рис. 1. Сигмоїдальна активаційна функція представляє собою s-образну функцію, яка змінюється в інтервалі між 0 та 1 [1]

Сигмоїдальна функція активації позначається як і визначається як [1]:

|  |  |
| --- | --- |
|  | (1) |

Ця функція активації часто використовується в моделях, в яких на виході нам потрібно передбачити ймовірність. Якщо z є великим числом, то в мові Python може трапиться так, що через помилки округлення функція поверне 0 або 1.

2) Функція активації – гіперболічний тангенс «tanh»

Гіперболічний тангенс позначається як і визначається як [1]:

|  |  |
| --- | --- |
|  | (2) |

Ця функція активації дуже схожа на сигмоїду. Діапазон її значень між -1 і 1.Однак варто відзначити, що градієнт тангенціальної функції більше, ніж у сигмоїди (похідна крутіше). Графік такої функції має наступний вигляд:

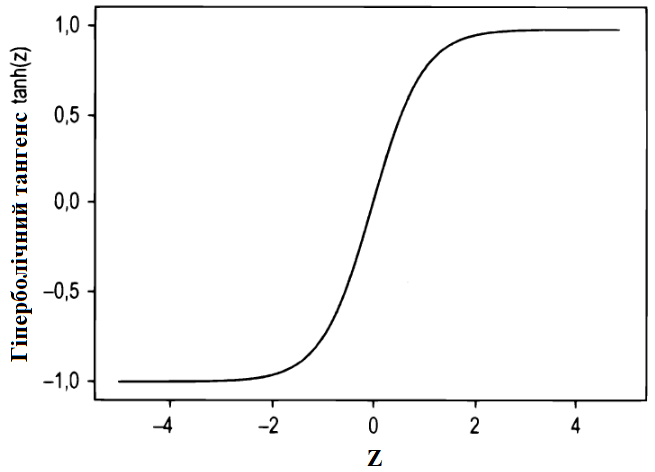


Рис. 2. Гіперболічний тангенс представляє собою s-образну функцію, яка змінюється в інтервалі між -1 та 1 [1]

3) Функція активації – ReLU

Ця функція активації позначається як і визначається як [1]:

|  |  |
| --- | --- |
|  | (3) |

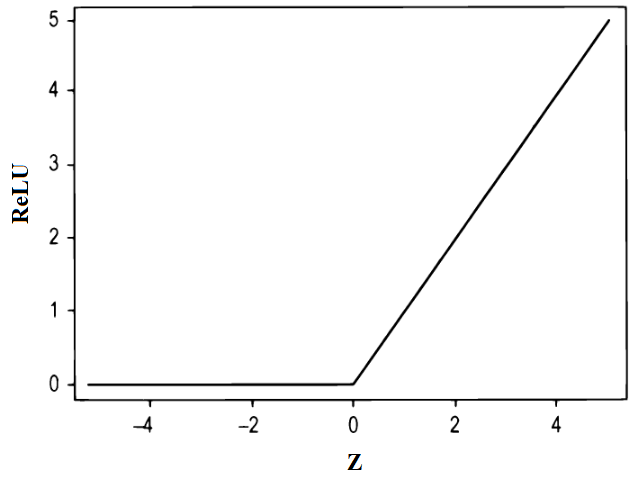
ReLU нелінійна за своєю природою. Така функція є хорошим апроксиматором, так як будь-яка функція може бути апроксимована комбінацією ReLU. 

Рис. 3. Функція ReLU [1]

4) Функція активації – ELU

Ця функція активації позначається як і визначається як [2]:

|  |  |
| --- | --- |
|  | (4) |

Ця функція як ReLU, тільки з кращим виходом для . Гіперпараметр контролює значення, до якого ELU насичується для негативних вхідних значень.



Рис. 4. Функція ELU [2]

5) Функція активації – SELU

Ця функція активації позначається як і визначається як [3]:

|  |  |
| --- | --- |
|  | (5) |

Тут – додатковий гіпермараметр (є регулятором s-образної частини функції).

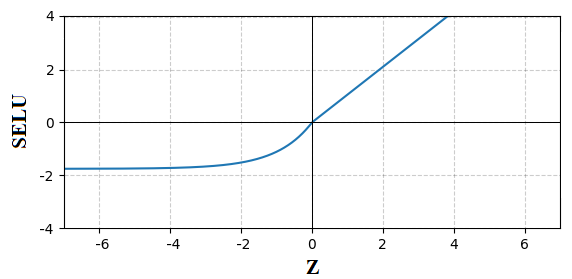


Рис. 5. Функція SELU [3]

6) Функція активації – Лінійна функція (Linear)

Лінійна функція активації позначається як і визначається як [4]:

|  |  |
| --- | --- |
|  | (6) |

Лінійна функція має дві лінійні ділянки, де функція активації дорівнює мінімально допустимому і максимально допустимому значенню. Лінійна активаційна функція дозволяє отримати цілий спектр значень, а не тільки бінарну відповідь. Ми її будемо використовувати тільки на вихідному шарі мережі.

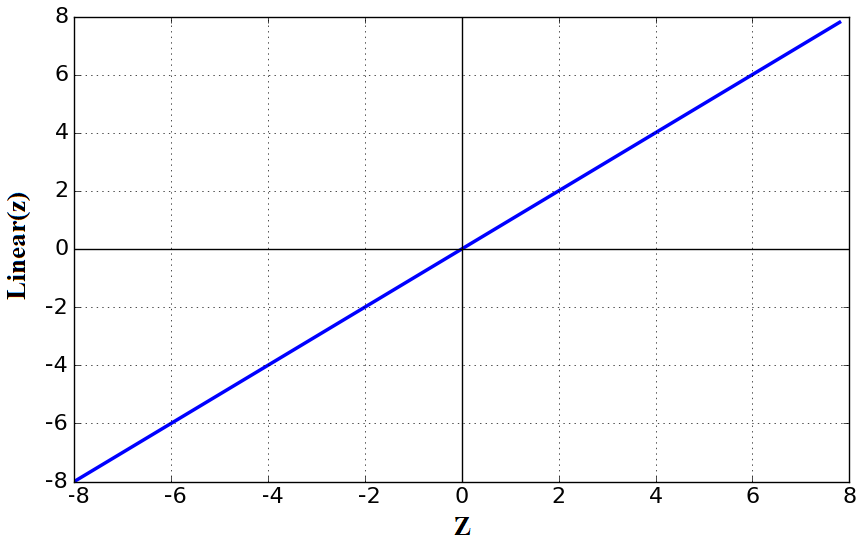


Рис. 5. Лінійна функція активації [4]

**1.2 Ініціалізація значень вагових коефіцієнтів**

Ініціалізатори визначають спосіб встановлення початкових випадкових ваг для шарів нейронної мережі. Ініціалізатори відіграють важливу роль у навчанні нейронних мереж, так як невдалі початкові параметри можуть означати те, що мережа буде навчатися повільно, або зовсім не зможе навчитися. Далі наступні позначення - кількість вхідних одиниць у ваговому тензорі. - кількість вихідних одиниць у ваговому тензорі.

Таблиця 1. Види ініціалізаторів для нейронних мереж [5]

|  |  |
| --- | --- |
| Random Normal | Ініціалізатор, який генерує тензори з нормальним розподілом. |
| Random Uniform | Ініціалізатор, який генерує тензори з рівномірним розподілом. |
| He\_Normal | Отримує вибірки з усіченого нормального розподілу з центром в 0 зі значенням |
| He\_Uniform | Отримує вибірки з рівномірного розподілу в межах , де |
| Ones | Ініціалізатор, який генерує тензори, ініціалізовані на одиницю. |
| Xavier Normal | Вибірки з усіченого нормального розподілу з центром в 0 зі значенням |
| Xavier Uniform | Отримує вибірки з рівномірного розподілу в межах , де |

**1.3 Типи регуляризації (регуляризатор Dropout)**

Перенавчання - одна з проблем глибоких нейронних мереж, що полягає в наступному: модель добре пояснює тільки приклади з навчальної вибірки, адаптуючись до навчальних прикладів, замість того щоб вчитися класифікувати приклади, які не брали участь в навчанні (втрачає здатність до узагальнення) [6].

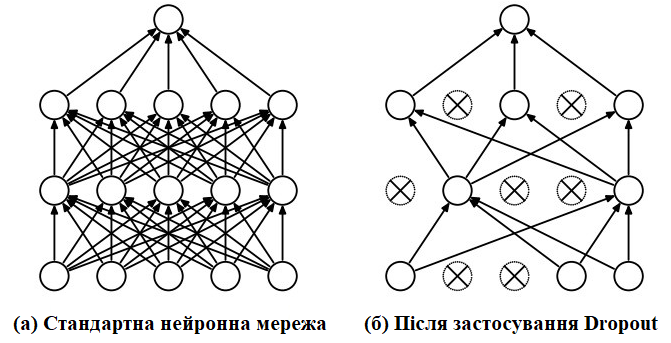


Рис. 6 Графічне представлення методу Dropout [6]

Головна ідея Dropout - замість навчання однієї глибокої нейронної мережі (ГНМ) навчити ансамбль декількох ГНМ, а потім усереднити отримані результати [6]. Мережі для навчання отримуються за допомогою виключення з мережі (dropping out) нейронів з імовірністю , таким чином, ймовірність того, що нейрон залишиться в мережі, становить . "Виключення" нейрона означає, що при будь-яких вхідних даних або параметрах він повертає 0. Виключені нейрони не роблять свій внесок в процес навчання ні на одному з етапів алгоритму зворотного поширення помилки; тому виключення хоча б одного з нейронів рівнозначно навчанню нової нейронної мережі. При умові, що

- - лінійна проекція вхідного - мірного вектора x на - мірний простір вихідних значень;

- – функція активації,

застосування Dropout до даної проекції на етапі навчання можна представити як змінену функцію активації [6]:

|  |  |
| --- | --- |
|  | (7) |

де - мірний вектор випадкових величин, розподілених за законом Бернуллі. має наступний розподіл ймовірностей:

|  |  |
| --- | --- |
| *,* | (8) |

де - всі можливі вихідні значення.

Очевидно, що ця випадкова величина ідеально відповідає Dropout, застосованому до одного нейрона. Дійсно, нейрон вимикають з ймовірністю , в іншому випадку - залишають включеним.

Подивимося на застосування Dropout до і-го нейрона [6]:

|  |  |
| --- | --- |
| *,* | (9) |

де

Так як на етапі навчання нейрон залишається в мережі з ймовірністю , на етапі тестування нам необхідно емулювати поведінку ансамблю нейронних мереж, використаного при навчанні. Для цього на етапі тестування треба помножити функцію активації на коефіцієнт . Таким чином,

На етапі навчання - ,

На етапі тестування -

Тобто, для нашої мережі ми повинні будемо знайти такий коефіцієнт при якому регуляризатор Dropout буде найбільш ефективним і подивитися чи збільшиться ефективність нашої мережі.

**1.4 Оптимізатори для нейронної мережі**

Оптимізація на основі стохастичних градієнтів має ключове практичне значення в багатьох областях науки і техніки. Багато проблем в цих областях можна представити як оптимізацію деякої скалярної параметризованої об'єктної функції, що вимагає максимізації або мінімізації по відношенню до її параметрів. Якщо функція є диференційованою за своїми параметрами, то градієнтний спуск є відносно ефективним методом оптимізації, так як при обчисленні часткових похідних першого порядку всі параметри є однаковими.

1) Оптимізатор - SGD

Градієнтний спуск - це популярний метод оптимізації в машинному навчанні і глибокому навчанні, і він може бути використаний з більшістю, якщо не з усіма алгоритмами навчання. Градієнт - це нахил функції. Він вимірює ступінь зміни змінної у відповідь на зміну іншої змінної. Математично градієнт спуску - це опукла функція, вихід якої є частковою похідною набору параметрів її входів. Чим більше градієнт, тим крутіше нахил. Починаючи з початкового значення, градієнтний спуск запускається ітеративно, щоб знайти оптимальні значення параметрів для знаходження мінімально можливого значення заданої функції витрат [9].

Типи градієнтного спуску:

* Пакетний градієнтний спуск
* Стохастичний градієнтний спуск

- Мініатюрний градієнтний спуск

Слово "стохастичний" означає систему або процес, пов'язаний з випадковою ймовірністю. Таким чином, при стохастичному градієнтному спуску для кожної ітерації випадковим чином вибирається кілька вибірок замість всього набору даних. У градієнтному спуску існує термін, званий "пакетним", який визначає загальну кількість вибірок з набору даних, яке використовується для обчислення градієнта за кожну ітерацію. У типовій оптимізації Gradient Descent, як і в Batch Gradient Descent, пакетним вважається весь набір даних. Хоча використання всього набору даних дійсно корисно для того, щоб дістатися до мінімуму менш шумним і менш випадковим чином, але проблема виникає, коли наші набори даних стають великими [9].

Припустимо, що в вашому наборі даних є мільйон вибірок, так що якщо ви використовуєте типову техніку оптимізації градієнтного спуску, то при виконанні градієнтного спуску вам доведеться використовувати всі один мільйон вибірок для завершення однієї ітерації, і це потрібно робити для кожної ітерації до тих пір, поки не буде досягнуто мінімальне значення. Отже, це стає дуже дорогим з обчислювальної точки зору.

Ця проблема вирішується стохастичним градієнтним спуском. У SGD для виконання кожної ітерації використовується тільки одна вибірка, тобто розмір партії один. Вибірка випадковим чином перетасовується і вибирається для виконання ітерації. SGD алгоритм [9]:

|  |  |
| --- | --- |
| *,* | (10) |

Отже, в SGD ми знаходимо градієнт функції втрат одної вибірки на кожній ітерації, а не суму градієнта функції втрат всіх прикладів. У SGD, так як для кожної ітерації випадковим чином вибирається тільки одна вибірка з набору даних, то шлях, який пройшов алгоритмом для досягнення мінімумів, зазвичай більш галасливий, ніж типовий алгоритм Gradient Descent. Але це не так важливо, тому що шлях, обраний алгоритмом, не має значення, поки ми досягаємо мінімумів зі значно меншим часом навчання.

Слід зазначити, що, оскільки SGD, як правило, «більш шумний», ніж типовий градієнтний спуск, для досягнення мінімуму зазвичай потрібна більша кількість ітерацій через його випадковість. Незважаючи на те, що для досягнення мінімуму потрібно більша кількість ітерацій, ніж при типовому градієнтному спуску, він все одно вимагає менше часу та зусиль, ніж типовий градієнтний спуск. Тому в більшості сценаріїв для оптимізації алгоритму навчання перевага віддається SGD, а не пакетному градієнтному спуску [9].

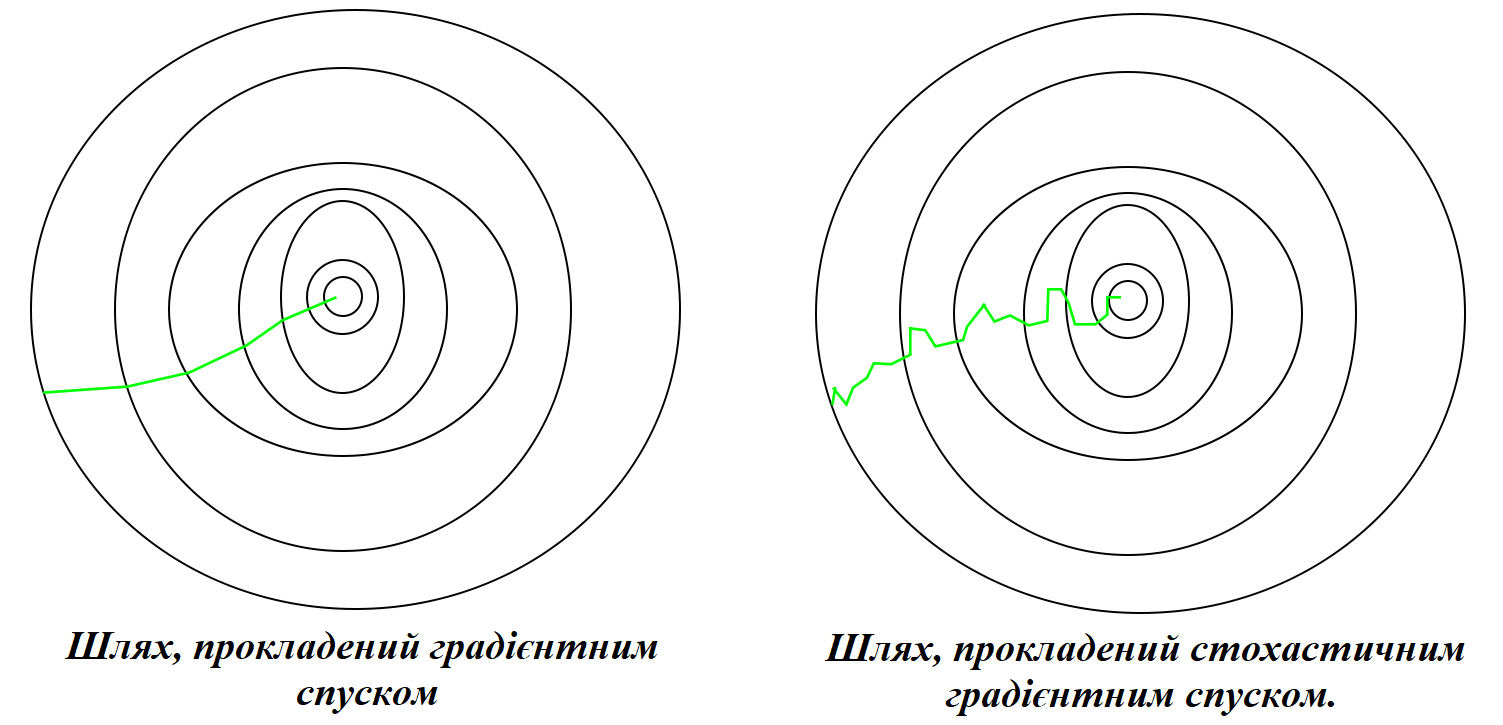


Рис. 7 Порівняння методу градієнтного спуску з методом SGD [9]

2) Оптимізатор – Adam

Оптимізатор Adam – це алгоритм градієнтної оптимізації стохастичних функцій першого порядку, заснований на адаптивних оцінках моментів нижнього порядку. Метод простий в реалізації, ефективний з точки зору обчислень, має невеликі вимоги до пам'яті, інваріантний до діагонального перемасштабування градієнтів і добре підходить для завдань з великим обсягом даних і/або параметрів. Гіперпараметри мають інтуїтивну інтерпретацію і, як правило, вимагають невеликої настройки. Емпіричні результати показують, що Адам добре працює на практиці і відрізняється від інших методів стохастичної оптимізації в позитивну сторону [10].

3) Оптимізатор – Adagrad

Adagrad - це метод стохастичної оптимізації, який адаптує швидкість навчання до параметрів. Він виконує менше оновлень параметрів, пов'язаних з функціями які часто зустрічаються, і більше оновлень параметрів, пов'язаних з функціями, які рідко зустрічаються. Adagrad модифікує загальну швидкість навчання на кожному кроці для кожного параметра , грунтуючись на минулих градієнтах для [10]:

|  |  |
| --- | --- |
|  | (11) |

Перевага AdaGrad полягає в тому, що він позбавляє від необхідності вручну налаштовувати швидкість навчання. Його основним недоліком є накопичення квадратних градієнтів у знаменнику, що призводить до зменшення швидкості навчання.

4) Оптимізатор – Adadelta

Adadelta - це метод стохастичного градієнтного спуску, заснований на адаптивній швидкості навчання на одиницю виміру для усунення двох недоліків [8]:

- Постійне зниження темпів навчання протягом усього навчання

- Необхідність ручного вибору глобальної швидкості навчання

Adadelta - більш надійне розширення Adagrad, яке адаптує швидкість навчання, замість того, щоб накопичувати всі минулі градієнти. Таким чином, Adadelta продовжує навчання навіть тоді, коли зроблено багато оновлень. У порівнянні з Adagrad, в оригінальній версії Adadelta немає необхідності встановлювати початкову швидкість навчання. У цій версії, як і в більшості інших оптимізаторів Keras, можна встановити початкову швидкість навчання.

5) Оптимізатор – RMSprop

RMSprop - це корекція до Adagrad, яка була запропонована незалежно від Adadelta, але розроблена приблизно в той же час. RMSprop еквівалентний до Adadelta, з однією різницею: швидкість навчання ділиться на експоненціально згасаюче середнє значення всіх квадратних градієнтів [10].

6) Оптимізатор – Adamax

Adamax, як і RMSprop та ін., використовує експоненціально затухаючу середньозважену оцінку дисперсії градієнта у своєму формулюванні. Однак, строго кажучи, ніщо не говорить про те, що ми повинні використовувати дисперсію. Дисперсія еквівалентна другому моменту або нормалізації градієнта. Норма визначається як [10]:

|  |  |
| --- | --- |
|  | (12) |

Значення більше після 2-го не дуже корисні, тому що вони не повинні бути чисельно стабільними. А що з приводу нескінченності?

|  |  |
| --- | --- |
|  | (13) |

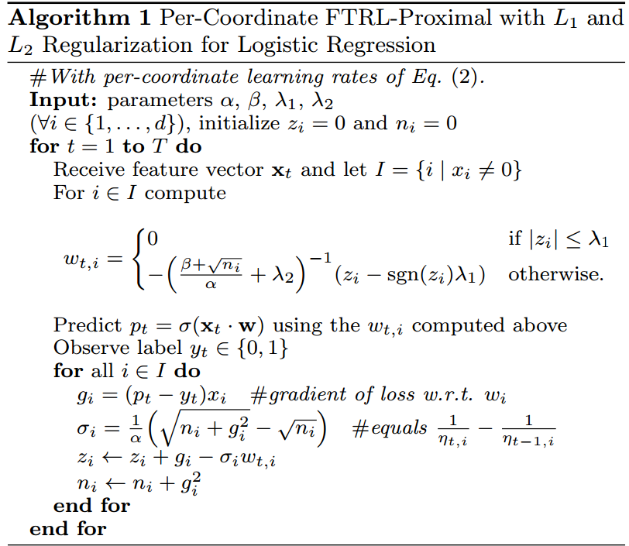
Нескінченна норма чисельно стабільна, так як асимптотично збігається! Тобто Adamax, в порівнянні з Adam, має більшу числову стабільність та більш стійкий до випадкових оновлень градієнту.

7) Оптимізатор – Nadam

Nadam - це Adam, але з Нестеровим імпульсом (спуск в алгоритмі, що використовує метод Нестерового моменту, залежить не тільки від поточного визначення алгоритму, але і від деяких кроків, які він зробив в минулому). Перевага використання імпульсу Нестерова замість звичайного імпульсу таке ж, як і у випадку з SGD [10].

8) Оптимізатор – Ftrl-Proximal

FTRL-Proximal (далі FTRL) є онлайн-алгоритмом (алгоритм, що може обробляти дані в міру їх надходження, не маючи загального доступу до всіх даних з самого початку). В цьому алгоритмі використовуються та регуляризатори.



**1.5 Метод перехресної перевірки як оцінка роботи оцінювача**

Вивчення параметрів функції прогнозування та її тестування на одних і тих же даних є методологічною помилкою: модель, яка просто повторила б значення концентрації заліза щойно побачених вибірок, мала б ідеальний результат, але не змогла б передбачити нічого корисного на ще не бачених даних. Така ситуація називається перенасиченням [11]. Щоб уникнути її, при виконанні (контрольованого) машинного навчального експерименту загальноприйнятою практикою є утримання частини доступних даних у вигляді тестового набору x\_test, y\_test (шукаємо по x\_test значення y\_test концентрації домішкового заліза в сонячному елементі). Нижче наведена схема типового робочого процесу перехресної перевірки (валідації) в навчанні моделі.

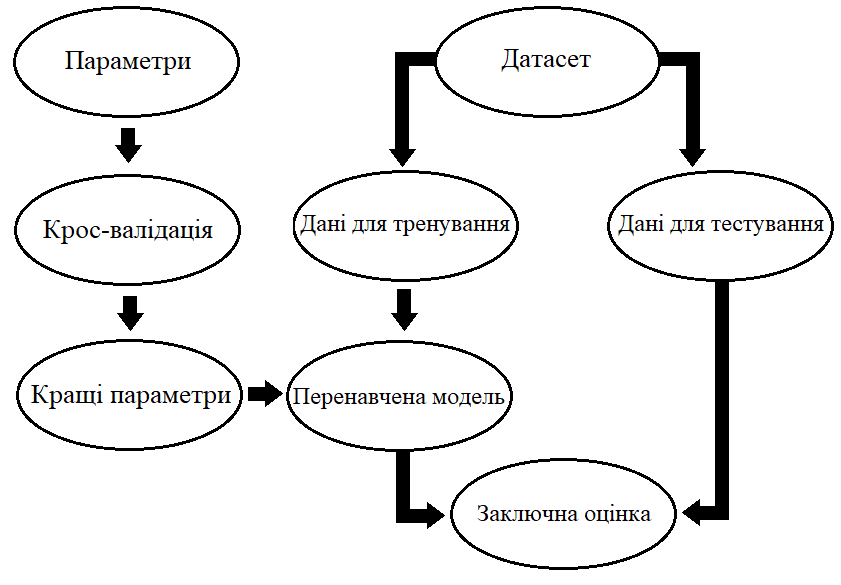


Рис. 8 Схема процесу Крос-валідації в навчанні моделі [11]

При оцінці різних гіперпараметрів все одно існує ризик дублювання в тестовому наборі. Таким чином, знання про тестову установку може "проникнути" в модель, а метрики оцінки більше не будуть повідомляти про продуктивність. Для вирішення цієї проблеми ще одна частина набору даних може бути представлена у вигляді так званого "набору валідації": навчання триває на тренувальному наборі, після чого оцінка проводиться на валідаційному наборі, а коли експеримент здається успішним, остаточна оцінка може бути проведена на тестовому наборі.

Однак, розбиваючи дані на три набори, ми зменшуємо кількість вибірок, які можуть бути використані для навчання моделі. Рішенням цієї проблеми є процедура, яка називається перехресною перевіркою. Набір для тестування все ще повинен використовуватися для остаточної оцінки, але при проведенні перехресної перевірки набір для перевірки більше не потрібен. При базовому підході (k - кратне резюме), тренувальний набір даних розбивається на k менших наборів. Для кожного k набору використовується наступна процедура [11]:

* Тренування з використанням k-1 наборів в якості тренувальних даних.
* Результуюча модель перевіряється на решті даних (тобто вона використовується в якості тестового набору для обчислення такої міри продуктивності, як похибка

Такий підхід може бути затратним з обчислювальної точки зору, але при цьому нам не треба мати дуже велику базу даних.

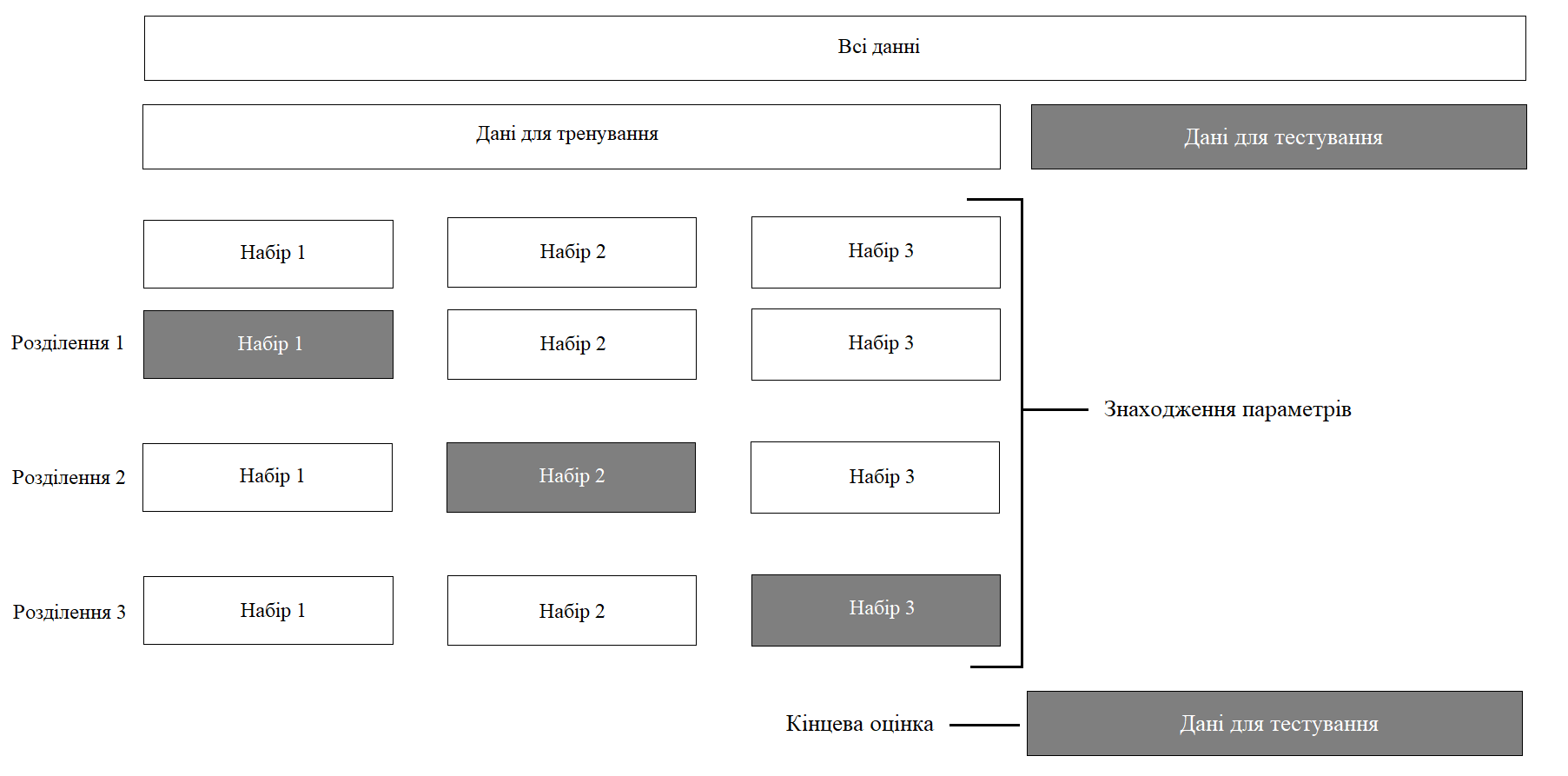


Рис. 9 Схема k – кратної перехресної перевірки (валідації) [11]

**2. Експериментальна частина**

**2.1 Дослідження впливу кількості нейронних шарів та кількості нейронів в них для визначення концентрації домішкового заліза.**

|  |  |
| --- | --- |
| Кількість епох – 100  Розмір міні-вибірки – 32  Оптимізатор – adam  Коефіцієнт навчання – 0.001 | Функція активації – relu  Ініціалізатор – Xavier Normal  Регуляризатор – немає  Архітектура мережі - ? |

В даній роботі ми використали нашу базу даних [12]. На вхід мережі ми подаємо 5 величини: температуру, товщину бази сонячного елемента, концентрацію бора, фактор неідеальності для заліза та для пари залізо-бор . Проводити тестування нашої мережі ми будемо на іншій базі даних, в якій ми створили проміжні значення для всіх вхідних величин (з меншим кроком, в порівнянні з тренувальною базою даних). Введемо наступні позначення:

* MRSE – наша функція втрат (середньоквадратична похибка)
* Aver MRSE – середнє значення нашої функції втрат MRSE для наших п’ятьох вхідних параметрів.
* Std MRSE – стандартне відхилення для наших п’ятьох вхідних параметрів.
* Tvar MRSE – середньоквадратична похибка на тестовому наборі даних в якому варіювалася температура.
* dvar MRSE - середньоквадратична похибка на тестовому наборі даних в якому варіювалася товщина сонячного елемента.
* Bvar MRSE - середньоквадратична похибка на тестовому наборі даних в якому варіювалася концентрація бору.
* Fevar MRSE - середньоквадратична похибка на тестовому наборі даних в якому варіювалася концентрація домішкового заліза.
* Allvar MRSE - середньоквадратична похибка на тестовому наборі даних, який є сумою всіх попередніх тестових наборів даних.

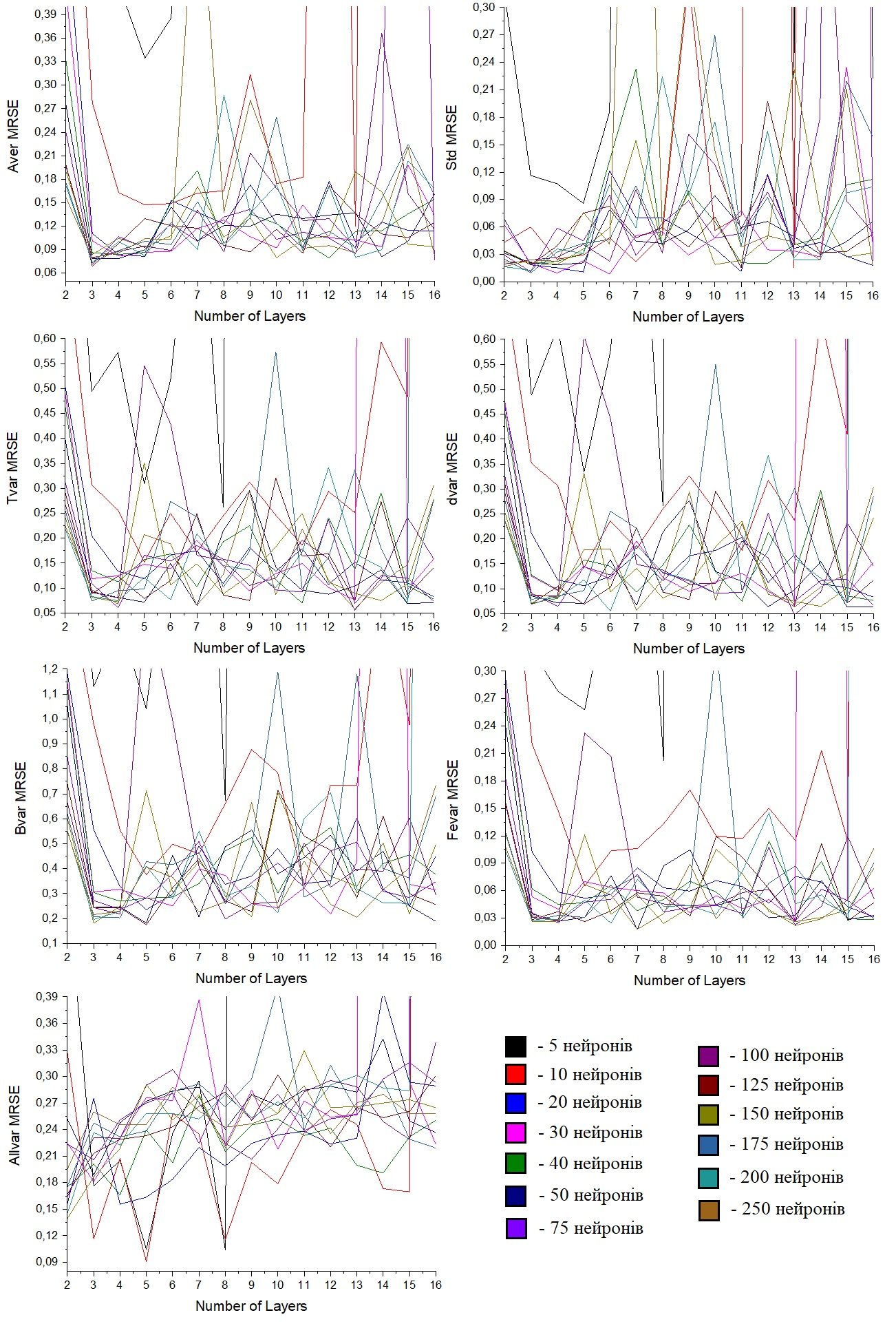


Рис. 10 Дослідження впливу кількості шарів та кількості нейронів в шарах

Отримані результати показали, що більше 4-6 шарів робити немає змісту. Результати після 6-ти шарів нейронів кращі, але якщо ми будемо використовувати 13 шарів нейронів, в кожному з яких буде ще по 50 нейронів, то отримаємо тільки велике навантаження на процесор, що призведе до дуже повільного навчання, особливо в подальшому, коли треба буде перебрати 6 функцій активації та інші гіпермапарметри. Ми втрачаємо не так багато ефективності, а часу втратимо при цьому дуже багато. Крім того використовувати більше 30-40 нейронів в цих шарах теж немає змісту (з тих самих міркувань), тому в подальшому дослідженні інших гіперпараметрів будемо використовувати 2 архітектури нейронної мережі:

1. 4 шари нейронів, по 30 нейронів в кожному.
2. 6 шарів нейронів, по 40 нейронів в кожному.

Отже, як висновок, маємо наступний набір гіперпараметрів, підкресленні гіперпараметри треба розуміти, як експериментально встановленні.

|  |  |
| --- | --- |
| Кількість епох – 100  Розмір міні-вибірки – 32  Оптимізатор – adam  Коефіцієнт навчання – 0.001 | Функція активації – relu  Ініціалізатор – Xavier Normal  Регуляризатор – немає  Архітектура мережі – 30-4, 40-6 |

**2.2 Дослідження впливу функції активації та кількості епох для визначення концентрації домішкового заліза.**

|  |  |
| --- | --- |
| Кількість епох – 100  Розмір міні-вибірки – 32  Оптимізатор – adam  Коефіцієнт навчання – 0.001 | Функції активації – ?  Ініціалізатор – Xavier Normal  Регуляризатор – немає  Архітектура мережі – 30-4, 40-6 |

Дослідимо тепер функції активації. За для зменшення складності завдання ми зупинимось на варіанті, коли всі нейрони, які знаходяться в одному шарі, будуть мати одну і ту саму функцію активації. Дослідимо залежність нашої функції втрат від кількості епох (кількість циклів навчання). Будемо проводити дослідження для двох архітектур (30-4, 40-6). Мета даного дослідження не стільки знайти найефективнішу функцію активації, а скільки відкинути максимальну кількість неефективних функцій активації нейронів. Будемо перебирати 5 функцій активації – elu, relu, selu, sigmoid, tanh. Введемо наступні позначення:

* TrainLoss – значення функції втрат на тренувальному наборі даних.
* ValidLoss – значення функції втрат на тестовому наборі даних.
* Epochs – кількість епох (циклів навчання).

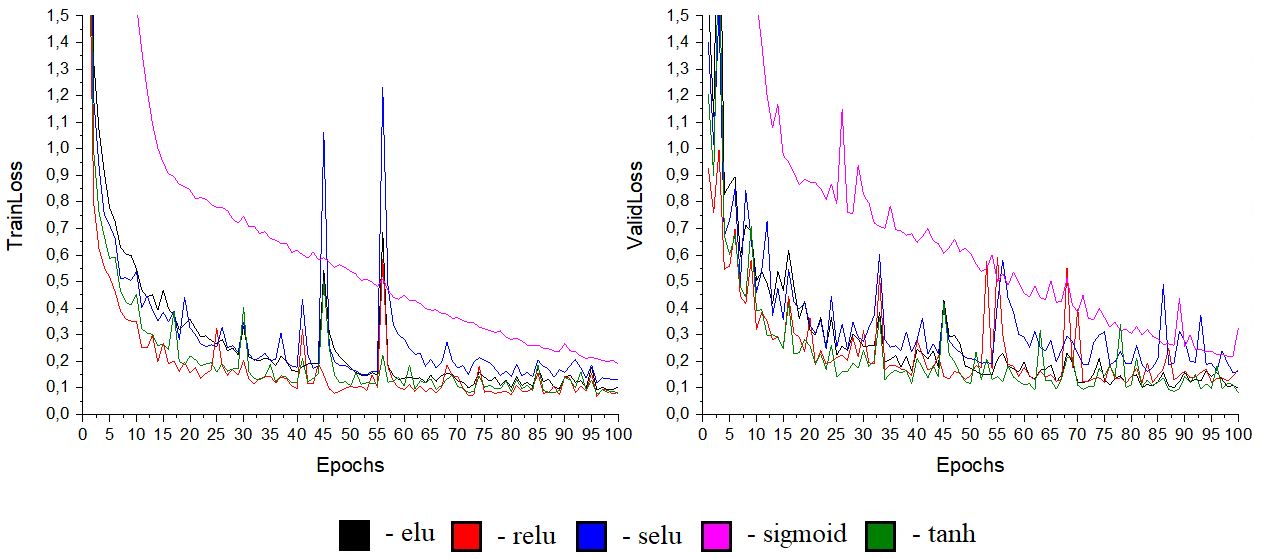


Рис. 11 Дослідження функцій активації для архітектури 4 шари по 30 нейронів

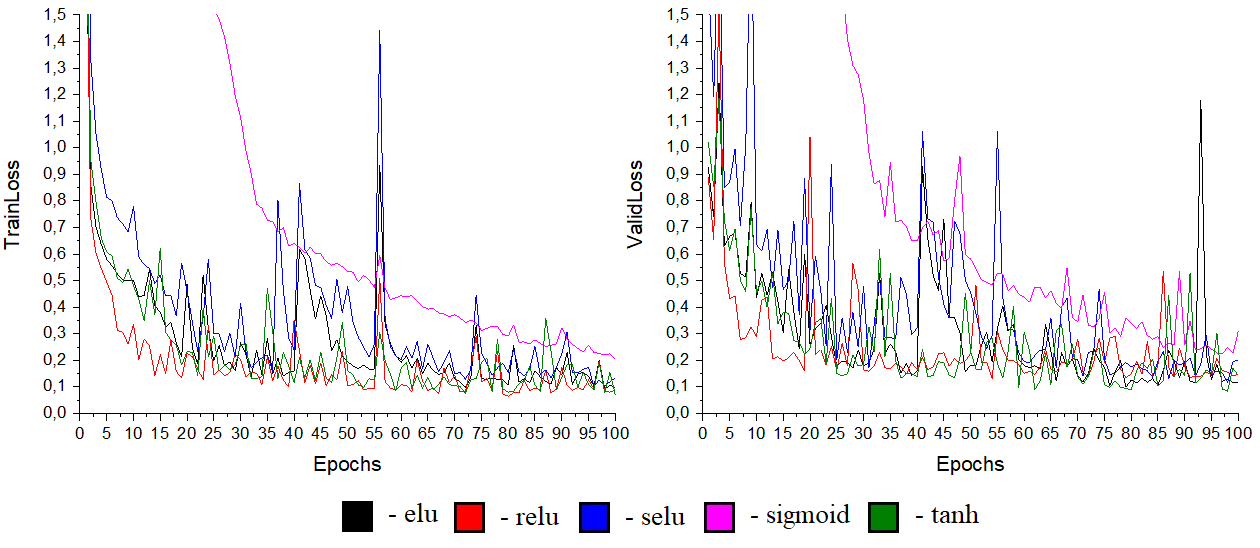


Рис. 12 Дослідження функцій активації для архітектури 6 шарів по 40 нейронів

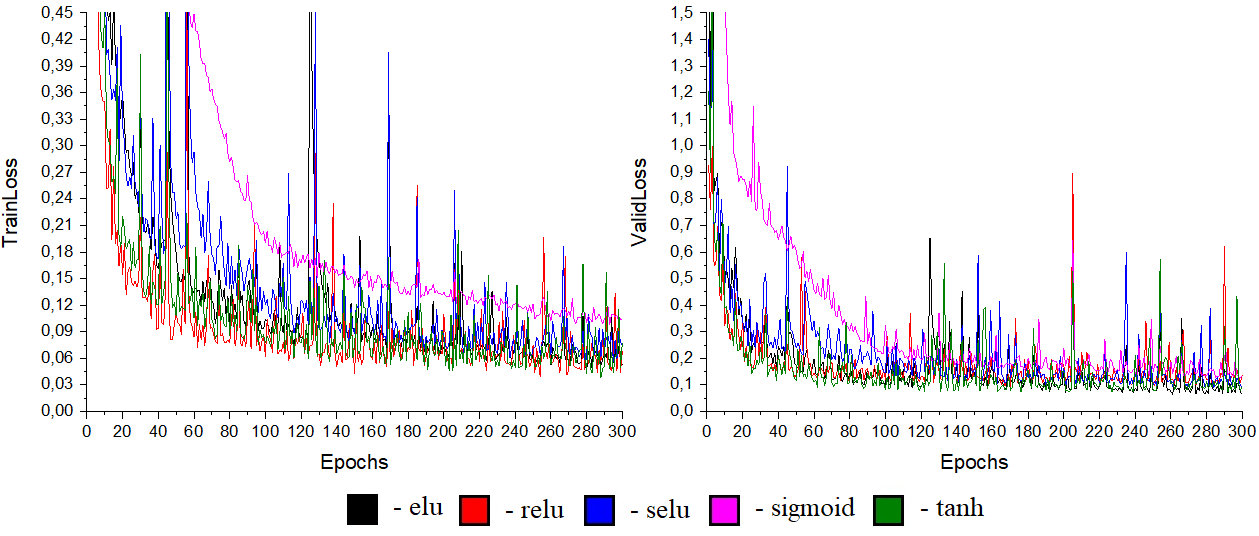
На відміну від інших функцій активації найбільш монотонно спадна є сигмоїда. Сигмоїда з першого погляду найгірша функція активації серед всіх, як і selu. Спробуємо збільшити кількість епох до 300 і дослідити ці залежності знову.

Рис. 11\* Дослідження функцій активації для архітектури 4 шари по 30 нейронів

Збільшення кількості епох зменшило середньоквадратичну похибку. Будемо використовувати тепер 300 епох для подальшого дослідження. Сигмоїда так само, як і при 100 епохах веде себе гірше всіх разом з selu, яка має дуже сильні осциляції на нашому проміжку. Найкраще показали себе relu та tanh, elu теж знаходиться десь поряд з ними по ефективності.

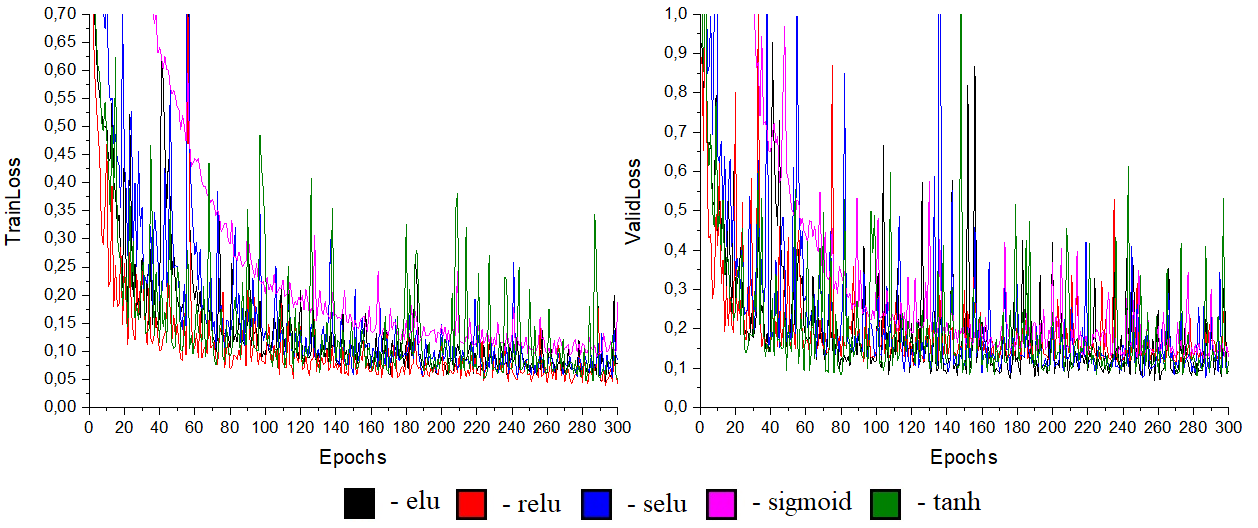


Рис. 12\* Дослідження функцій активації для архітектури 6 шарів по 40 нейронів

|  |  |
| --- | --- |
| Кількість епох – 300  Розмір міні-вибірки – 32  Оптимізатор – adam  Коефіцієнт навчання – 0.001 | Функції активації – relu, tanh, elu  Ініціалізатор – Xavier Normal  Регуляризатор – немає  Архітектура мережі – 30-4, 40-6 |

**2.3 Дослідження впливу розміру блоків навчання (Batch size) для визначення концентрації домішкового заліза.**

|  |  |
| --- | --- |
| Кількість епох – 300  Розмір міні-вибірки – ?  Оптимізатор – adam  Коефіцієнт навчання – 0.001 | Функції активації – relu, tanh, elu  Ініціалізатор – Xavier Normal  Регуляризатор – немає  Архітектура мережі – 30-4, 40-6 |

Дослідимо тепер вплив розміру блоків навчання. Можна побачити, що великі значення Batch size не збільшують ефективність нейромережі. Крім того, з Рис. 13 можна бачити, що функція активації relu помітно менш ефективна за функції активації elu та tanh, тому в подальших дослідженнях будемо використовувати саме elu та tanh. Найкращу ефективність має функція активації tanh при розмірі міні вибірки 170, але таких великих значень Batch size ніхто не використовує, тому будемо обмежуватися малими значеннями розмірів вибірки, які лежать в інтервалі між 20 та 40. Будемо дотримуватися наступної ідеї. Розмір блоку навчання буде мати значення . Число n в нашому випадку візьмемо рівним 5-ти. Це дасть змогу попасти в ефективний діапазон розмірів блоків навчання між значеннями 30 та 40. Отже, маємо наступне:

|  |  |
| --- | --- |
| Кількість епох – 300  Розмір міні-вибірки – 32  Оптимізатор – adam  Коефіцієнт навчання – 0.001 | Функції активації – elu, tanh  Ініціалізатор – Xavier Normal  Регуляризатор – немає  Архітектура мережі – 30-4, 40-6 |

Похибка експерименту Std MRSE на цьому етапі становить близько (4 – 5) % . На наступному етапі будемо досліджувати вплив ініціалізаторів вагових коефіцієнтів.

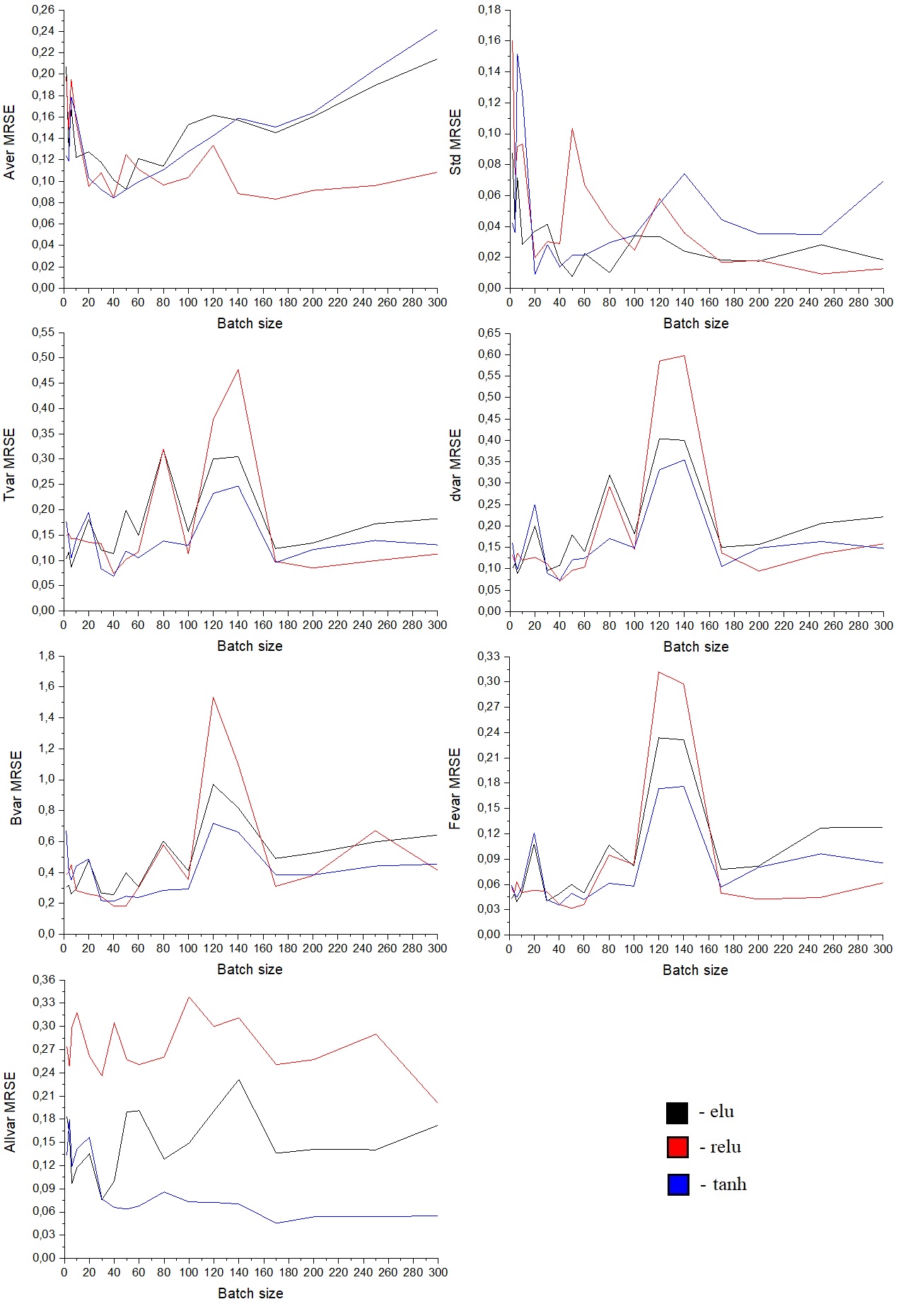


Рис. 13 Дослідження впливу розміру блоків навчання (Batch size)

**2.4 Дослідження впливу ініціалізаторів для визначення концентрації домішкового заліза.**

|  |  |
| --- | --- |
| Кількість епох – 300  Розмір міні-вибірки – 32  Оптимізатор – adam  Коефіцієнт навчання – 0.001 | Функції активації – elu, tanh  Ініціалізатор – Xavier Normal  Регуляризатор – немає  Архітектура мережі – 30-4, 40-6 |

На графіках використанні наступні позначення (дивись главу 1.2):

|  |  |  |
| --- | --- | --- |
| XN - Xavier Normal | HN - He\_Normal | RN - Random Normal |
| XU - Xavier Uniform | HU - He\_Uniform | RU - Random Uniform |

Якщо зосередити увагу на Allvar MRSE, то для функції активації elu найкращим ініціалізатором є Random Uniform як для архітектури 4 схованих шари, в кожному по 30 нейронів, так і для архітектури 6 схованих шарів, в кожному по 40 нейронів, але якщо зосереджувати увагу на Tvar MRSE, dvar MRSE, Bvar MRSE та Fevar MRSE, то для них кращим ініціалізатором є He\_Normal, щодо архітектури 6 шарів 40 нейронів в кожному, то фаворитів серед ініціалізаторів там немає. Якщо дивитись саме на функцію активації – tanh, то в архітектурі 4 шари 30 нейронів в кожному – фавориту немає, а от в архітектурі 6 шарів по 40 нейронів є очевидний фаворит серед ініціалізаторів - He\_Uniform. В середньому найкращі результати показує модель з чотирьома схованими шарами, в кожному з яких по 30 нейронів, найкраща функція активації – elu, найкращий ініціалізатор для такої моделі - Xavier Normal. Отже, обираємо найефективніші ініціалізатори для обох наших функцій активації нейронів:

|  |  |
| --- | --- |
| Кількість епох – 300  Розмір міні-вибірки – 32  Оптимізатор – adam  Коефіцієнт навчання – 0.001 | Функції активації – elu, tanh  Ініціалізатор – XN (elu), HU (tanh)  Регуляризатор – немає  Архітектура мережі – 30-4, 40-6 |

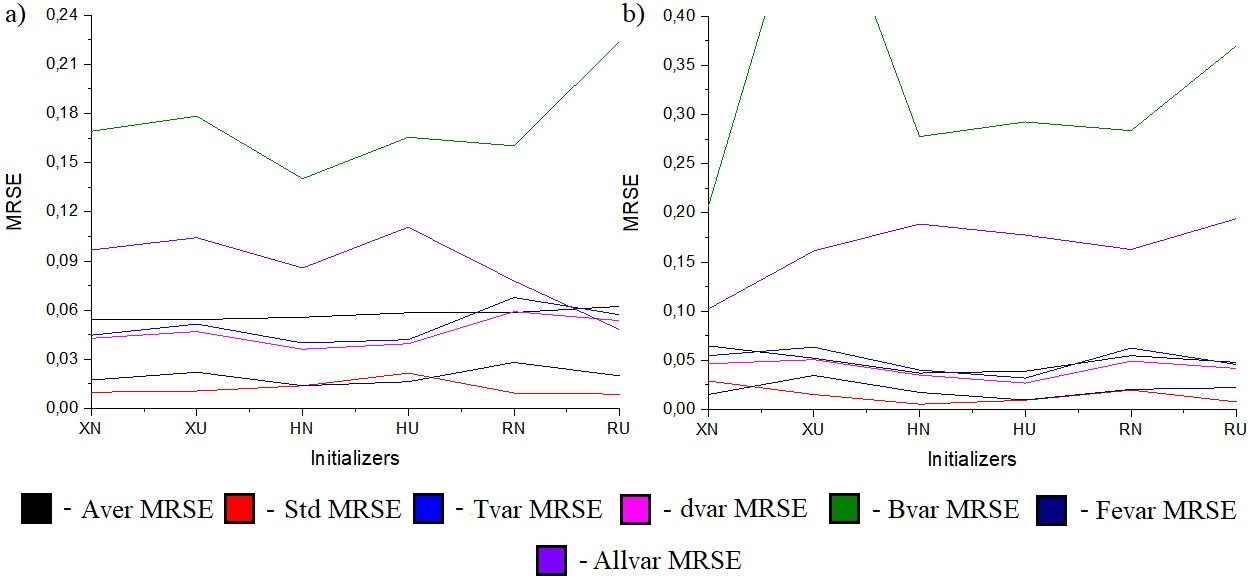
****

Рис. 14.1 Дослідження ініціалізаторів для архітектури 4 схованих шари та 30 нейронів в них для функцій активації elu (a) та tanh (b) при 300 епохах



Рис. 14.2 Дослідження ініціалізаторів для архітектури 6 схованих шарів та 40 нейронів в них для функцій активації elu (a) та tanh (b) при 300 епохах

**2.5 Дослідження впливу кількості епох та нормалізації даних для визначення концентрації домішкового заліза.**

|  |  |
| --- | --- |
| Кількість епох – ?  Розмір міні-вибірки – 32  Оптимізатор – adam  Коефіцієнт навчання – 0.001 | Функції активації – elu, tanh  Ініціалізатор – XN (elu), HU (tanh)  Регуляризатор – ?  Архітектура мережі – 30-4, 40-6 |

На графіку залежності Avar MRSE від кількості епох чітко видно, що найкращі результати досягаються тоді, коли значення кількості епох стає рівним 1500 епох (0.0448 MRSE), але треба розуміти, що при 300 епохах наша функція втрат не дуже відрізняється від значення функції втрат при 1500 епохах (0.0545 MRSE). Отже, на цьому графіку однозначно фаворитом є 40\_6 – tanh – HU при 300 епохах. Якщо ж ми подивимось на залежність Std MRSE від кількості епох, то можемо зробити висновок, що тут на одному рівні йдуть 40\_6 – tanh – HU і 30\_4 – elu – XN, фаворит 30\_4 – elu – XN при 300 епохах (0,01 MRSE). Для графіку залежності Tvar MRSE від кількості епох фаворит 40\_6 – tanh – HU при 300 епохах (0,038 MRSE). Для залежності dvar MRSE від кількості епох фаворит 40\_6 – tanh – HU при 300 епохах (0,035 MRSE). Для залежності Bvar MRSE від кількості епох фаворит однозначно 30\_4 – elu – XN при 300 епохах (0,169 MRSE). Для залежності Fevar MRSE від кількості епох фаворитом є 40\_6 – tanh – HU при 300 епохах (0,013 MRSE). Для залежності Allvar MRSE від кількості епох фаворитом вже виступає оптимізована модель 40\_6 – tanh – HU при 300 епохах (0,047 MRSE). Можна зробити висновок, що нормалізація даних для наших двох моделей просто не потрібна, а найкраща кількість епох – 300. Отже, маємо:

|  |  |
| --- | --- |
| Кількість епох – 300  Розмір міні-вибірки – 32  Оптимізатор – adam  Коефіцієнт навчання – 0.001 | Функції активації – elu, tanh  Ініціалізатор – XN (elu), HU (tanh)  Регуляризатор – немає  Архітектура мережі – 30-4, 40-6 |

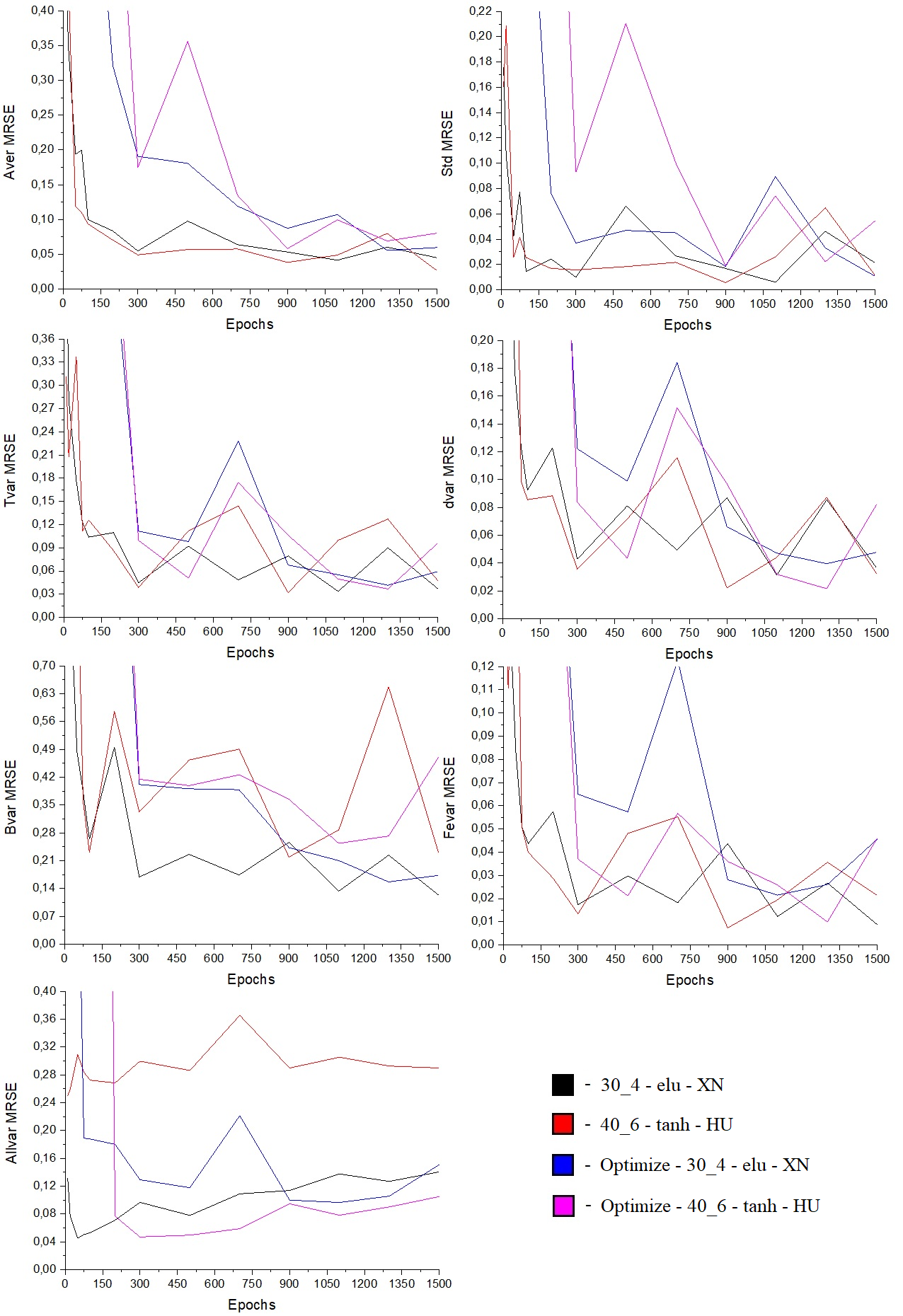


Рис. 15 Дослідження впливу нормалізації даних та кількості епох

**2.6 Дослідження впливу оптимізатора та коефіцієнта навчання (learning rate) для визначення концентрації домішкового заліза.**

Нехай модель, в якої 4 схованих шари по 30 нейронів з функцією активації elu та ініціалізатром XN, буде модель (а), в модель, в якої 6 схованих шари по 40 нейронів з функцією активації tanh та ініціалізатором HU, буде модель (б), тоді з графіків 16.1 та 16.2 можемо отримати наступних «переможців»:

Таблиця 2. Найкращі оптимізатори та коефіцієнти навчання для моделей (а), (б)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 30\_4 – elu - XN | | | 40\_6 – tanh - HU | | |
| Оптимізатор | Коефіцієнт навчання | MRSE | Оптимізатор | Коефіцієнт навчання | MRSE |
| Aver | Adamax | 0.01 | 0.0575 | Nadam | 0.001 | 0.0419 |
| Std | Adamax | 0.00316 | 0.0105 | Adamax | 0.001 | 0.0092 |
| Tvar | Adamax | 0.00316 | 0.0509 | Nadam | 0.000316 | 0.0331 |
| dvar | Adamax | 0.00316 | 0.0471 | Adamax | 0.00316 | 0.0396 |
| Bvar | Adamax | 0.00316 | 0.1618 | Adamax | 0.01 | 0.2366 |
| Fevar | Adamax | 0.00316 | 0.0186 | Nadam | 0.000316 | 0.0102 |
| Allvar | Nadam | 0.00316 | 0.1025 | Ftrl | 0.01 | 0.0402 |

Можемо бачити, що очевидним лідером для моделі (а) є оптимізатор Adamax при коефіцієнті навчання – 0.00316, тоді як для моделі (б) не все так очевидно і там відбувається боротьба між оптимізаторами Nadam та Adamax з точки зору коефіцієнтів навчання теж все не так однозначно. Таку поведінку можна пояснити тим, що однозначний переможець для моделі (а) насамперед з’явився через не існування інших оптимізаторів на великих значеннях коефіцієнта навчання. Можна також зробити висновки, що великі коефіцієнти навчання (більше ніж 0.01) не є ефективними і тільки погіршують ситуацію.

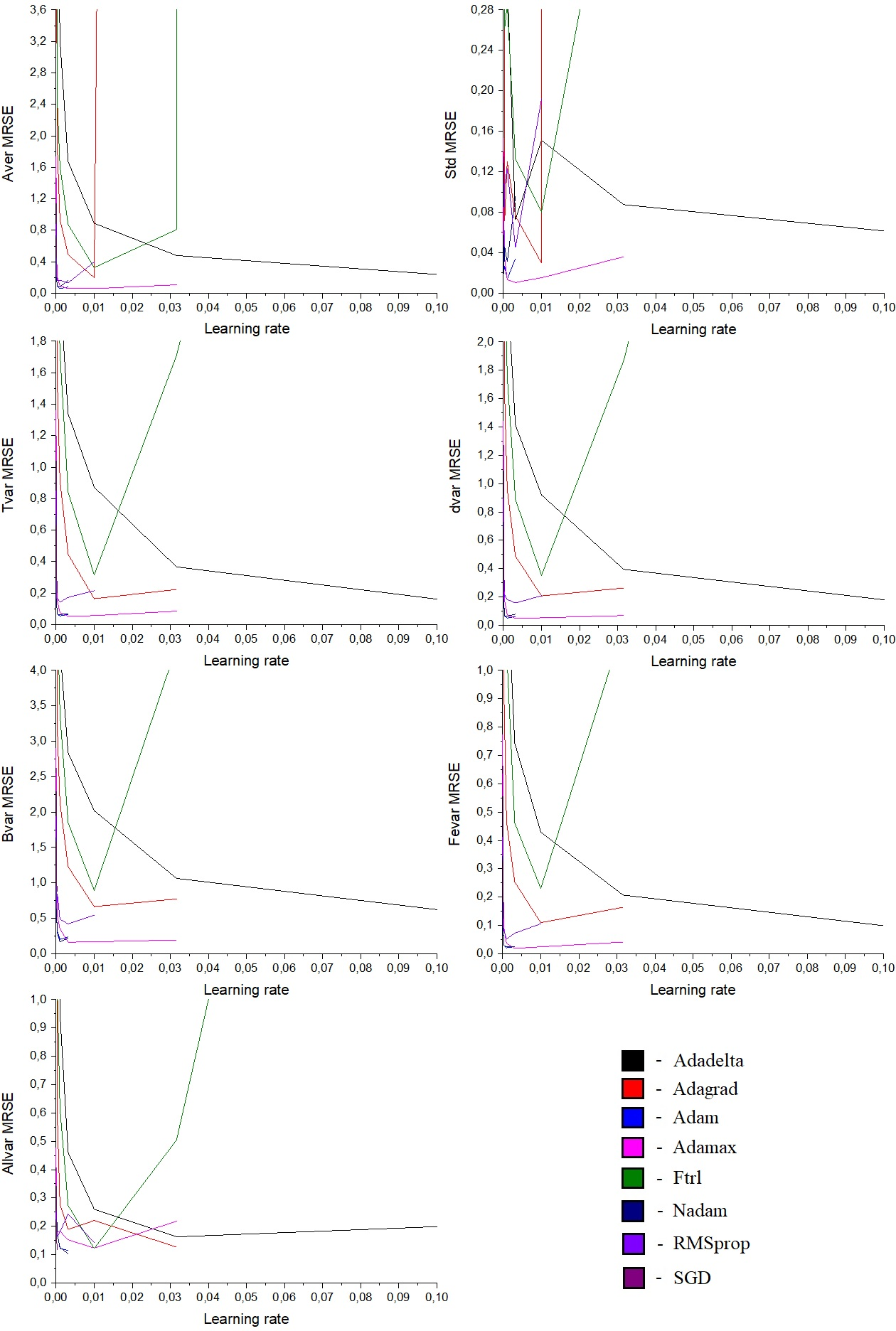
****

Рис. 16.1 Вплив оптимізаторів та коефіцієнта навчання на моделі (а)

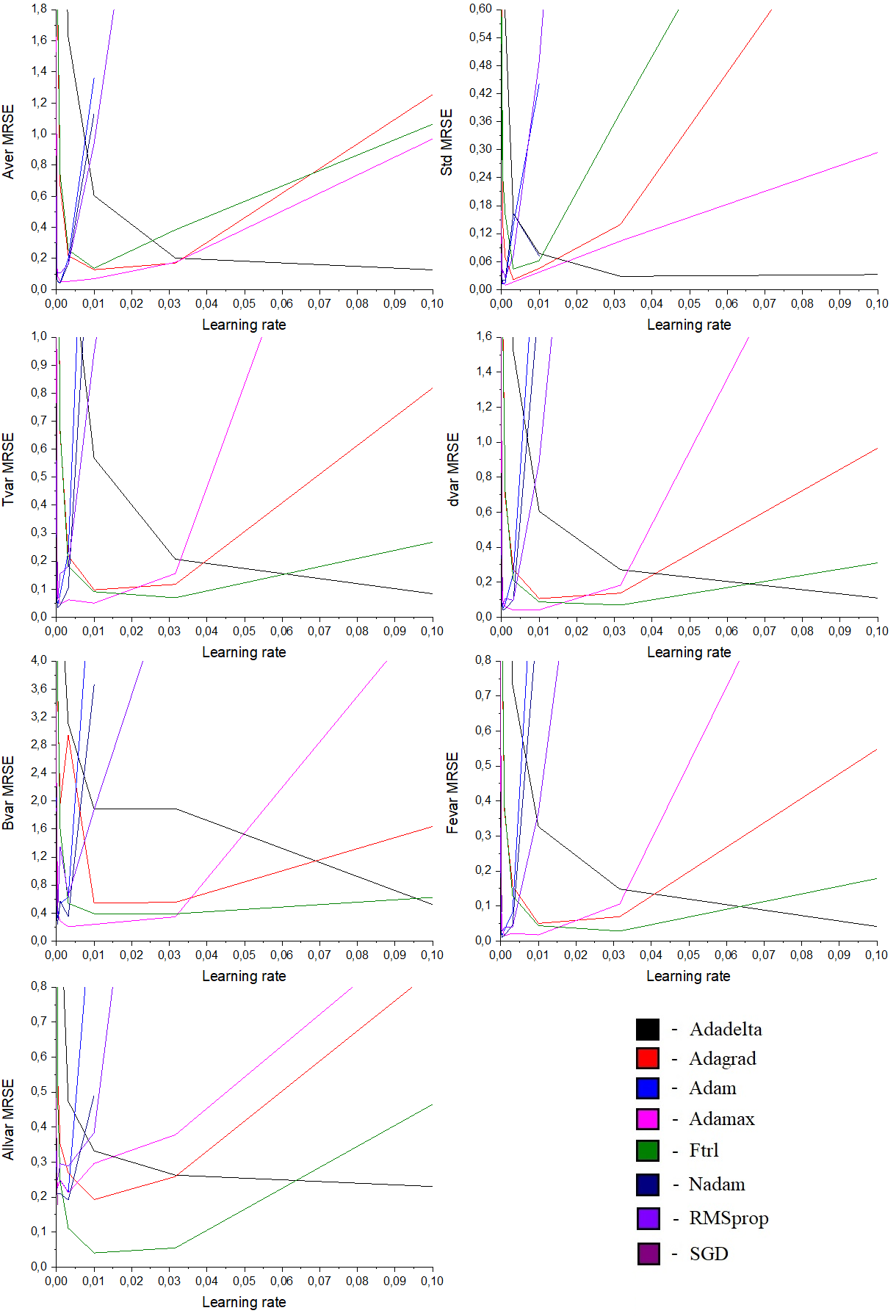


Рис. 16.2 Вплив оптимізаторів та коефіцієнта навчання на моделі (б)

**2.7 Дослідження впливу регуляризації для визначення концентрації домішкового заліза.**

Регуляризатори L1, L2, Dropout використовуються тоді, коли в вашій моделі існує таке явище як перенавчання, тобто модель показує гарні результати на тренуванні та тестуванні ваших наборів, але коли діло доходить до іншої тестової бази даних, то гарні результати раптово зникають. Саме регуляризатори дозволяються вийти з цієї проблеми шляхом «дегродування» моделі, як вже говорилось. Якщо дивитися на наш випадок, то зрозуміло, що з графіку 17.1 випливає тільки те, що зі збільшенням коефіцієнту регуляризації ефективність мережі стрімко спадає як у випадку моделі (а) так і у випадку моделі (б). З рисунку 17.2 видно, що регуляризатор Dropout, який ми розглядали в огляді літератури трошки раніше, теж не дає результатів і тільки погіршує ефективність. В дослідженні, графіки якого на рисунку 17.2, ми додали ще 2 моделі, в яких збільшили кількість нейронів у вузлах, щоб подивитись чи доб’ємося ми перенавчання і чи «включиться у гру» регуляризатор Dropout, але все безрезультатно. Як висновок, регуляризатори нам не потрібні, тоді остаточні моделі:

1 модель

|  |  |
| --- | --- |
| Кількість епох – 300  Розмір міні-вибірки – 32  Оптимізатор – Adamax  Коефіцієнт навчання – 1е-3 | Функція активації – elu  Ініціалізатор – XN  Регуляризатор – немає  Архітектура – 5-30-30-30-30-1 |

Похибка моделі 1: 1,05 %

2 модель

|  |  |
| --- | --- |
| Кількість епох – 300  Розмір міні-вибірки – 32  Оптимізатор – Adamax  Коефіцієнт навчання – 3е-4 | Функції активації –tanh  Ініціалізатор –HU  Регуляризатор – немає  Архітектура – 5-40-40-40-40-40-40-1 |

Похибка моделі 2: 0,92 %

****

Рис. 17.1 Дослідження впливу L1 та L2 регуляризації на моделях (а) та (б)

****

Рис. 17.2 Дослідження впливу Dropout регуляризації на моделях (а) та (б)

**Висновки**

Проведені дослідження показали, що найбільш ефективними є наступні моделі:

1 модель

|  |  |
| --- | --- |
| Кількість епох – 300  Розмір міні-вибірки – 32  Оптимізатор – Adamax  Коефіцієнт навчання – 1е-3 | Функція активації – elu  Ініціалізатор – Xavier Normal  Регуляризатор – немає  Архітектура – 5-30-30-30-30-1 |

Похибка моделі 1: 1,05 %

2 модель

|  |  |
| --- | --- |
| Кількість епох – 300  Розмір міні-вибірки – 32  Оптимізатор – Adamax  Коефіцієнт навчання – 3е-4 | Функції активації –tanh  Ініціалізатор –He\_Uniform  Регуляризатор – немає  Архітектура – 5-40-40-40-40-40-40-1 |

Похибка моделі 2: 0,92 %

**СПИСОК ВИКОРИСТАННОЇ ЛІТЕРАТУРИ**

1. Умберто Микелуччи / Прикладное глубокое обучение. Подход к пониманию глубоких нейронных сетей на основе метода кейсов // "БХВ-Петербург". - 2020 г.

2. https://github.com/BindiChen/machine-learning/blob/master/tensorflow2/010-popular-activation-functions/

3. https://atcold.github.io/pytorch-Deep-Learning/en/week11/11-1/

4. https://towardsdatascience.com/activation-functions-neural-networks-

1cbd9f8d91d6

5. www.tensorflow.org/api\_docs/python/tf/keras/initializers/

6. https://habr.com/ru/company/wunderfund/blog/330814/

7. John Duchi, Elad Hazan, Yoram Singer /Adaptive Subgradient Methods for Online Learning and Stochastic Optimization // Journal of Machine Learning Research. - Vol. 12. - 2011. – Pp. 2121-2159

8. Matthew D. Zeiler / ADADELTA: An Adaptive Learning Rate Method // ArXiv ID:1212.5701, https://arxiv.org/abs/1212.5701

9. https://www.geeksforgeeks.org/ml-stochastic-gradient-descent-sgd/

10. <https://www.kaggle.com/residentmario/keras-optimizers>

11. https://scikit-learn.org/stable/modules/cross\_validation.html

12. O. Ya. Olikh, O. V. Zavhorodnii / Modeling of ideality factor value in structure // Journal of Physical Studies, - Oct. – 22. – 2020. – Vol. 24. – No. 4.