УДК 681.5

*А.Ю. Дорошенко, П.А. Іваненко, О.С. Новак*

# ОПТИМІЗАЦІЯ АВТОТЬЮНІНГУ ПРОГРАМ З ВИКОРИСТАННЯМ НЕЙРОМЕРЕЖ

Автотьюнінг програм – це метод самоналаштування внутрішніх параметрів програми, що мають вплив на її швидкодію, з метою досягнення найвищих показників продуктивності, проте він може вимагати багато часу на випробування. В роботі запропоновано вдосконалення методу автотьюнінгу програм з використанням нейромережевих алгоритмів та статистичного моделювання. Автоматичне навчання моделі програми на результатах "традиційних" циклів тьюнінгу з подальшою підміною частини запусків автотьюнера оцінкою з апроксимаційної моделі дозволяє значно прискорити пошук оптимального варіанта програми.

Ключові слова: автотьюнінг, статистичне моделювання, автоматизація розробки програмного забезпечення, нейромережі.

## Вступ

У попередній роботі авторів [1] запропоновано гібридизацію відомого методу самоналаштування (автотьюнінгу) програм [2], що поєднує реальне випробування програми з використанням статистичного моделювання для звуження простору пошуку оптимального варіанту програми. Головною метою методу при цьому залишається підвищення ефективності використання наявних обчислювальних ресурсів. Питання оптимального використання таких ресурсів є одним із основних для будь-якого програмного забезпечення. Воно є важливим як для невеликих застосунків, так і для складних високонавантажених програмних комплексів. Така проблема оптимізації програмних систем нині вирішується, зокрема, за допомогою методу автотьюнінгу [2], який може успішно використовуватись незалежно від особливостей обчислювального середовища. Характеристичною рисою автотьюнінгу є емпіричне оцінювання якості оптимізованої програми (під якістю зазвичай розуміють швидкодію й точність результату). Тим самим, розробник програми може оцінити результат без моделювання усіх етапів виконання програми та отримати досить точні дані щодо цього виконання у конкретному обчислювальному середовищі.

Проте, застосування автотьюнінгу для складних і нетривіальних програмних систем зазвичай вимагає багато часу. В даній роботі розглядається вдосконалення вищезгаданого гібридного підходу до автотьюнінгу з використанням техніки нейромереж і машинного навчання. В результаті запропонований підхід дозволяє автоматичне навчання моделі на результатах "традиційних" циклів тьюнінгу й подальшу підміну частини запусків автотьюнера оцінкою з апроксимаційної моделі, що значно пришвидшує пошук оптимального варіанта програми.

## Метод автотьюнінгу

Максимальна ефективність програми досягається шляхом її налаштування (тьюнінгу) під обчислювальне середовище (ОС), у якому вона буде виконуватися. Як зазначено [2], метод дозволяє автоматизувати це налаштування. Основою для автотьюнінгу є ідея емпіричної оцінки декількох варіантів програми й вибору найкращого. Програма-тюнер може бути як частиною оптимізовуваної програми, так і окремим додатком. Для повноти переліку часто ще вказують тюнер, інтегрований у операційну систему. Проте, досі не було створено універсальної імплементації такого варіанта.

Хоча методологія автотьюнінгу вже виправдала свою універсальність й ефективність, у процесі її використання доводиться враховувати значний час роботи автотьюнера й необхідність його написання.

Роботу програми-тюнера можна назвати досить шаблонною: обрати/створити

нову версію програми й отримати емпіричну оцінку її швидкодії. Потреба написання коду самого автотьюнера усувається автоматичним генеруванням його коду з вихідного коду оптимізованої програми (наприклад, як це робиться в програмній системі автотьюнінгу TuningGenie [3–4]). Завдяки цьому вихідна програма займається тільки розв'язанням її основної задачі й абстрагована від кількісних характеристик ОС, що значно спрощує її розробку. Така абстракція також дозволяє створювати програмні додатки, що будуть автоматично оптимізуватися для виконання на апаратних засобах наступних поколінь (принаймні доки не зміниться багатоядерна архітектура обчислювачів).

Далі в цій публікації під тьюнером будемо розуміти окрему програму, а точніше універсальну систему генерації автотьюнерів TuningGenie.

Отже, головним недоліком автотьюнінгу залишаються істотні часові і обчислювальні витрати, зумовлені необхідністю емпіричного оцінювання у цільовому середовищі великої множини варіантів вихідної програми. Для скорочення цих витрат гібридний підхід [1] застосовує машинне навчання. А саме – емпіричне оцінювання частини варіацій підміняється оцінкою з апроксимаційної моделі програми.

## Використання статистичного моделювання та нейромереж

Усі статистичні алгоритми (у тому числі й алгоритми машинного навчання) потребують значної кількості статистичних даних для аналізу та побудови моделі [5]. В контексті задач автотьюнінгу збір великої кількості статистичних даних може бути доволі тривалим процесом. Тому досить гостро стає проблема алгоритмів, які дозволять максимально звузити область пошуку при мінімальній кількості реальних запусків автотюнера. Щоб частково вирішити цю проблему, в даній роботі запропоновано використовувати нейромережу для екстраполяції даних. В такому випадку необхідна буде відносно невелика кількість реальних запусків для побудови наближеної моделі, після чого нейромережева модель може бути використана іншими алгоритмами за принципом «чорної скрині».

Як правило, задачі машинного навчання (в тому числі побудови нейромереж) в залежності від природи «сигналу» поділяють на такі категорії [6]:

*навчання з учителем.* Системі надаються приклади входів та їхніх бажаних виходів, задані «вчителем», і метою є навчання загального правила, яке відображає входи на виходи;

*навчання без учителя*. Алгоритму навчання не надається жодних міток, залишаючи його самому знаходити структуру на своєму вході. Навчання без учителя може бути метою саме по собі (виявлення прихованих закономірностей у даних), або засобом досягнення мети (навчання ознакам);

*навчання з підкріпленням*. Комп'ютерна програма взаємодіє з динамічним середовищем, у якому вона має виконувати певну мету (наприклад, таку як водіння авто) без учителя, який явно казав би їй, чи підійшла вона близько до мети. Іншим прикладом є навчання грі через гру із суперником.

Загалом, можна описати загальний підхід до вирішення такого типу задач [7]:

- *аналіз записів і підготовка «сирих» даних до завантаження*. Оскільки дані можуть приходити з різних джерел і в різних форматах, то потрібно все привести до єдиного вигляду;
- *підготовка даних*. На цьому етапі вирішуються проблеми неповноти даних, шумів, суперечності в даних і т. п. Основні задачі цього етапу – очищення (заповнення відсутніх значень, видалення спотворених даних та випадкових викидів), перетворення (нормалізація для зниження спотворень), ущільнення (створення вибірок даних для окремих атрибутів/груп атрибутів та дискретизація (перетворення неперервних атрибутів у категоріальні);
- *аналіз даних та побудова моделі;*
- *перевірка моделі на тестовій вибірці даних.*

Для перевірки точності моделі можна використовувати декілька методів, один з них – матриця похибок (confussion matrix) з різними відтінками позитивного чи негативного прогнозу [8] (див. таблицю).

Таблиця

| | Позитивний прогноз | Негативний прогноз |
|---|---|---|
| Позитивне значення | Позитивний результат | Псевдонегативний результат (статистична похибка 2-го роду) |
| Негативне значення | Псевдопозитивний результат (статистична похибка 1-го роду) | Негативний результат |

Тут позитивний чи негативний результат – це ситуації, коли прогноз моделі співпадає чи, відповідно, не співпадає з реальним результатом. Статистична похибка 1-го роду – ситуація коли модель дає позитивний прогноз для негативного результату. А статистична похибка 2-го роду – дзеркальна ситуація (негативний прогноз до позитивного результату).

В цьому випадку, для оцінки якості моделі можна використовувати точність

$$Acc = \frac{TP + TN}{P + N},$$

де $TP$ – кількість позитивних результатів, $TN$ – кількість негативних результатів, $P$ – дзеркальна кількість позитивних значень, $N$ – загальна кількість негативних значень.

## Практичний експеримент

Для практичного експерименту в якості об'єкта тьюнінгу було використано розширену версію алгоритму з попередньої роботи авторів [1] (алгоритм сортування злиттям чи вставками, в залежності від розміру блоку числового масиву даних).

Сам алгоритм був реалізований на мові Java.

Первинний аналіз даних виконувався на мові Python з використанням бібліотеки Sckikit learn [9]. Подальший аналіз виконувався за допомогою мови R.

Експеримент складався з декількох стандартних етапів: підготовка та завантаження записів роботи автотьюнера, підготовка даних (в тому числі нормалізація), побудова нейромережевої моделі на навчальній виборці даних та перевірка моделі на контрольній виборці даних.

Сам процес, за яким проходив аналіз даних, показано на рис. 1. Спочатку автотьюнер проводить $N$ експериментів та зберігає дані для нейромережі в окремий файл. Ці дані використовувались нейромережею для навчання. Після навчання, нейромережа екстраполювала дані генеруючи новий набір даних (в окремий файл). І в кінці, обидва набори даних аналізувались і порівнювались вже людиною.

В якості нейромережі обрано багатошаровий перцептрон з трьома вхідними нейронами, трьома прихованими шарами (по 20-10-5 нейронів відповідно) та одним вихідним. Як активатор у нейронів використовувалась випрямлена лінійна функція ( $f(x) = \max(0, x)$ ). Як метод навчання використовувався метод зворотного поширення помилки (та алгоритм Бройдена–Флетчера–Гольдфарба–Шанно для оптимізації вагових коефіцієнтів).

Приклад вхідних даних показано на рис. 2, де (V1 – номер запуску автотьюнера, вилучено з рисунка), V2,V3,V4 – вхідні параметри, V5 – результат, V6 – якісна оцінка результату.

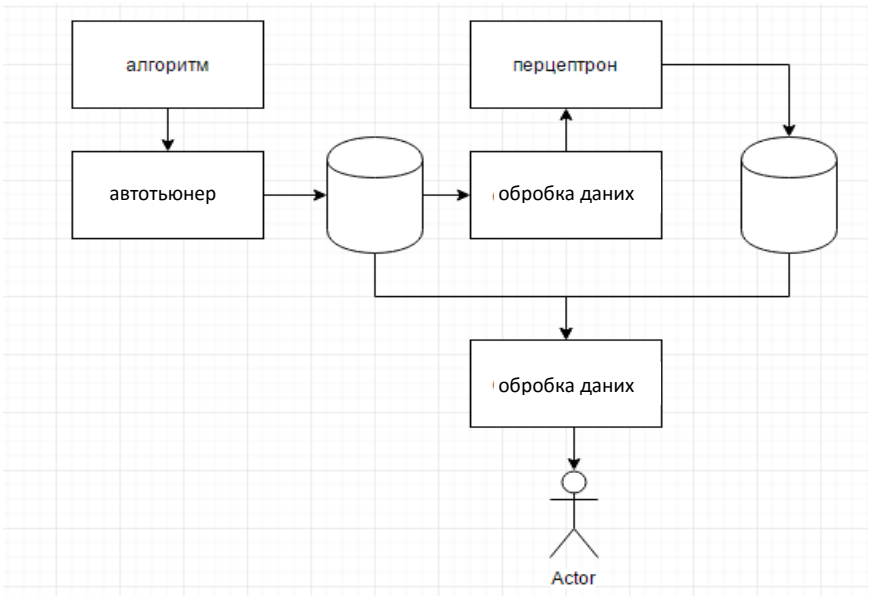


Рис. 1. Процес аналізу

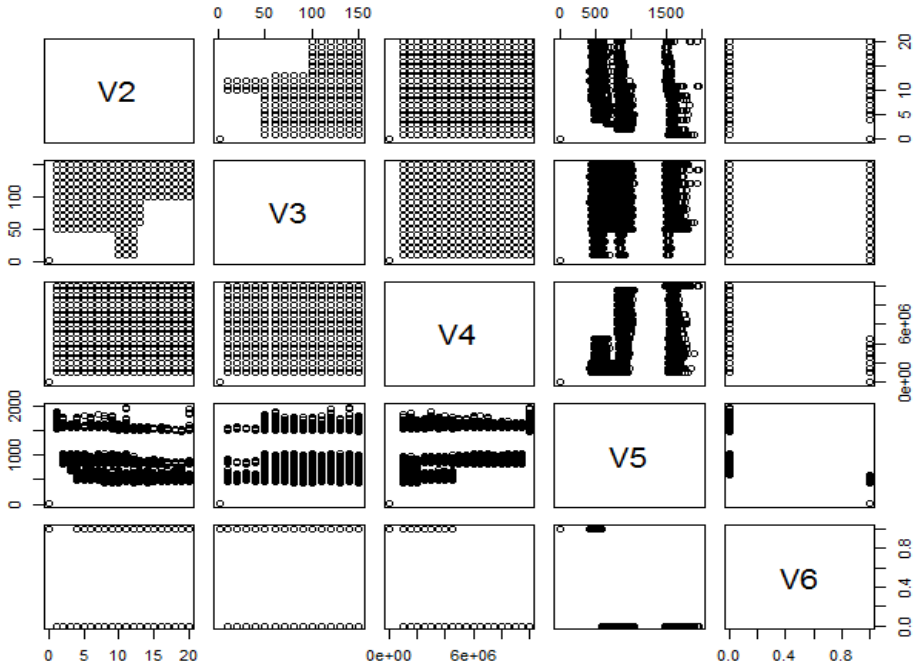


Рис. 2. Вхідні дані

На основі результатів декількох тисяч запусків (3300) побудовано нейромережу, яку було використано для подальшої генерації даних (рис. 3, 4). Крім того, як буде видно з рис. 5 – це не мінімальна кількість даних які потрібні для побудови моделі, і кількість замірів можливо буде зменшити при незначній втраті в точності. Варто ще зазначити, що дані первісних замірів виявились не зовсім рівномірно розподіленими, що пояснює білі області для залежності V2 ~ V3 на рис. 2 і далі).

З рис. 4 можна побачити, що комбінація параметрів X0 та X2 мають значний вплив на результат роботи алгоритму, що збігається з результатами ручного аналізу даних у минулій публікації [1]. Тут біла область – комбінація параметрів при якій відсутні «неякісні» результати.

В контексті даного експерименту використання нейромережі для початкового приближення дозволило звузити область пошуку на 58 % (з $10^6$ до $4.2*10^5$).

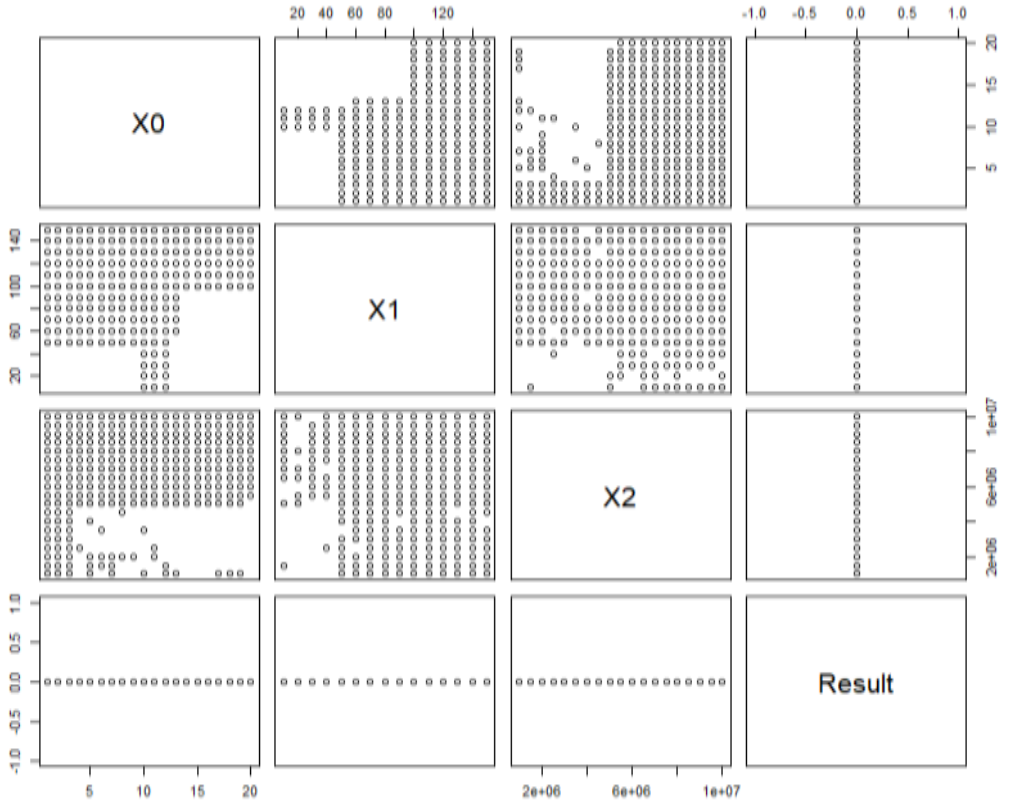


Рис. 3. Вхідні дані (3300 запусків автотьюнера)

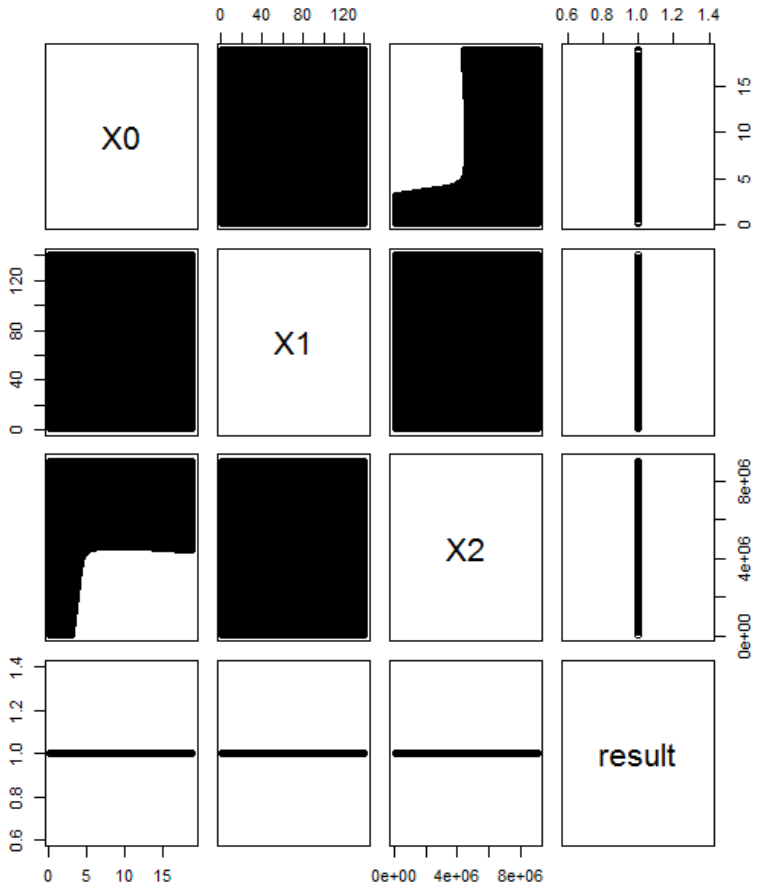


Рис. 4. Результати прогнозування (125 мільйонів емуляцій запуска автотьюнера)

## Аналіз результатів експерименту

Для перевірки якості отриманих результатів було зроблено більше 30 тисяч реальних запусків автотьюнера (рівномірно розподілених по всій множині варіантів) та проаналізовані результати роботи. (рис. 4). Хід роботи тут був аналогічний тому, що був описаний в експериментальній частині окрім кількості реальних запусків автотьюнера. Після чого було порівняно результат прогнозу нейромережі (на основі 3 тисяч запусків) з реальними результатами.

Для оцінки якості моделі використовувалась матриця похибок (confussion matrix). На рис. 6 показано залежність точності моделі від частки даних яку було використано для навчання. Тут по осі Y – кількість відсотків від всіх наявних даних яка використовувалась для навчання моделі, по осі X – середня точність (ACC) від 10 нейромереж що були навчені на цих даних.

У випадку використання 10 % від усіх замірів (тобто, 3300 замірів) матриця мала вигляд,

| | TP | PN |
|---|---|---|
| CP | 19978 | 570 |
| CN | 488 | 9506 |

де TP – позитивний прогноз, PN – негативний прогноз, CP – позитивне значення, CN – негативне значення. Тоді ACC = 96 %.

Оскільки мета цього етапу – відсіяти очевидно неякісні варіанти, то питання точності тут менш критичне (так як ці дані будуть ще раз оброблені на наступному етапі вже іншим алгоритмом). Більш критичним тут є зменшення кількості похибок 2-го роду.

З цього можна зробити висновок, що для звуження області пошуку автотьюнера, за умови збереження високої точності результатів, розглянутій системі достатньо відносно невеликої частини замірів.

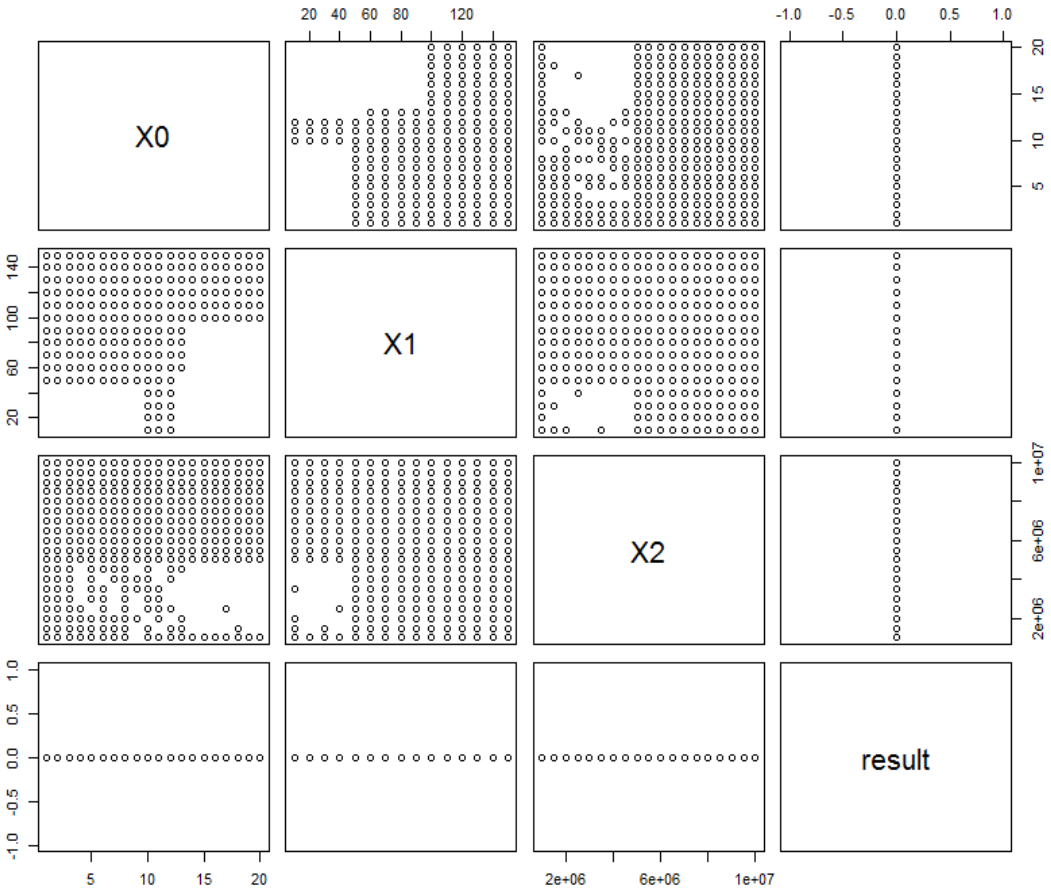


Рис. 5. Вхідні дані (33 тисячі запусків автотьюнера)

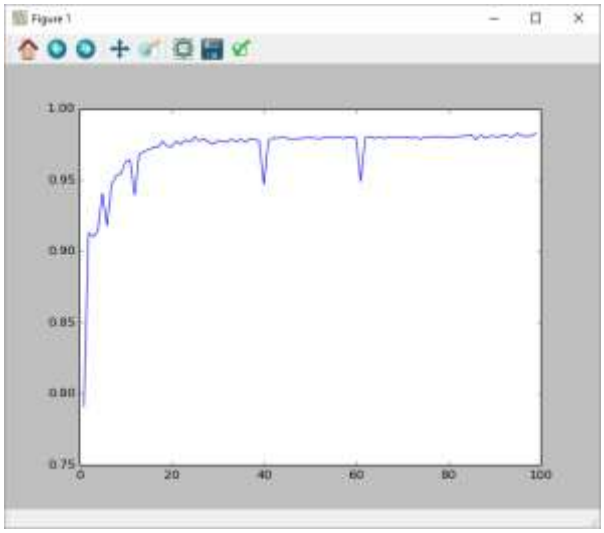


Рис. 6. Залежність точності моделі від кількості даних для навчання

## Висновки

При всіх недоліках, притаманних нейромережам (необхідність у великій кількості даних для навчання, можлива велика обчислювальна складність), їх використання все-ж таки уможливлює помітне звуження області пошуку конфігурацій й підходить як перший етап роботи автотьюнера.

Крім того, завдяки можливостям перцептронів у задачах класифікації, використання такого типу нейромережі як перший фільтр дозволяє легко виділити найбільш вагомі вхідні параметри (тобто ті, що мають найбільший вплив на кінцевий результат).

З негативних сторін слід виділити, що на сьогодні нейромережі – це досить складна прикладна область, яка сама по собі потребує налаштування метапараметрів, що впливають на швидкодію і якість роботи системи, починаючи від кількості нейронів і конфігурації нейромережі та завершуючи різноманітними коефіцієнтами.

1. Дорошенко А.Ю., Іваненко П.А., Новак О.С. Гібридна модель автотьюнінгу з використанням статистичного моделювання. *Проблеми програмування.* 2016. № 4. С. 27–32.
2. Naono K., Teranishi K., Cavazos J., Suda R. Software Automatic Tuning From Concepts to State-of-the-Art Results. Springer; 1st Edition. Edition, 2010.
3. Ivanenko P.A, Doroshenko A.Y., Zhereb K.A. TuningGenie: Auto-Tuning Framework Based on Rewriting Rules. 10th International Conference, ICTERI 2014, Kherson, Ukraine, June 9–12, 2014. Revised Selected Papers. P. 139–158.
4. Ivanenko P.A, Doroshenko A.Y. Method for automated generation of autotuners for parallel applications. *Cybernetics and system analisys*. 2014. N 3. P. 75–83.
5. Tom M. Mitchell. Machine learning. mcGraw-Hill Science/Engineering/Math., 1997.
6. Russell, Stuart, Norvig, Peter. Artificial Intelligence: A Modern Approach (2nd edition). Prentice Hall.
7. Jiawei Han, Micheline Kamber, Jian Pei. Data mining: Cencepts and Techniques, 3rd edition. Morgan Kaufmann, 2011.
8. Tom Fawcett. An indroduction to ROC analysis. Elsevier B.V., 2005.
9. Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, Édouard Duchesnay. Scikit-learn: Machine learning in Python. 2011. JMLR 12. P. 2825–2830.

## References

1. Doroshenko A., Ivanenko P., Novak O. Hybrid auto tuning model with the use of a static modelling. Scientific journal “Problems of programming”. 2016, N 4. P. 27–32.
2. Naono K., Teranishi K., Cavazos J., Suda R. Software Automatic Tuning From Concepts to State-of-the-Art Results. Springer; 1st Edition. Edition, 2010
3. Ivanenko P.A., Doroshenko A.Y., Zhereb K.A. TuningGenie: Auto-Tuning Framework Based on Rewriting Rules. 10th International Conference, ICTERI 2014, Kherson, Ukraine, June 9–12, 2014. Revised Selected Papers. P. 139–158.
4. Ivanenko P.A., Doroshenko A.Y. Method for automated generation of autotuners for parallel applications. *Cybernetics and system analisys.* 2014. N 3. P. 75–83.

5. Tom M. Mitchell. Machine learning. mcGraw-Hill Science/Engineering/Math., 1997.
6. Russell, Stuart, Norvig, Peter. Artificial Intelligence: A Modern Approach (2nd edition). Prentice Hall.
7. Jiawei Han, Micheline Kamber, Jian Pei. Data mining: Cencepts and Techniques, 3rd edition. Morgan Kaufmann, 2011.
8. Tom Fawcett. An indroduction to ROC analysis. Elsevier B.V. 2005.
9. Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, Édouard Duchesnay. Scikit-learn: Machine learning in Python. 2011. JMLR 12, P. 2825–2830.

Одержано 03.05.2017

***Про авторів:***

*Дорошенко Анатолій Юхимович,*
доктор фізико-математичних наук,
професор, завідувач відділу теорії
комп'ютерних обчислень,
професор кафедри автоматики
та управління в технічних системах
НТУ України "КПІ".
Кількість наукових публікацій в
українських виданнях – понад 200.
Кількість наукових публікацій в
зарубіжних виданнях – понад 50.
Індекс Хірша – 5.
http://orcid.org/0000-0002-8435-1451,

*Іваненко Павло Андрійович,*
молодший науковий співробітник.
orcid.org/0000-0001-5437-9763,

*Новак Олександр Сергійович,*
аспірант.
orcid.org/0000-0002-1665-7360.

***Місце роботи авторів:***

Інститут програмних систем
НАН України.
проспект Академіка Глушкова, 40.
03187, Київ,
НТУ України «КПІ».
Тел.: (044) 526 3559.
E-mail: a-y-doroshenko@ukr.net,
paiv@ukr.net,
novak.as@i.ua