масс. Это также является косвенным подтверждением реакций между оксидом магния и соединениями титана.

Практической рекомендацией, которую можно сделать из полученных экспериментальных данных, является то, что загружать увлажненную соль в сухое время года в головной аппарат поточной линии во время заливки очередной порции хлорида магния.

Литература

1. Баранник, И. А. Промышленные исследования влияния примесей титана на электролиз хлористого магния [Текст] / И. А. Баранник, З. В. Ястребова, А. П. Егоров и др. // Цветная металургія. — 1971. — № 8. — С. 40—42.
2. Мужжавлев, К. Д. Влияние влажности воздуха на выход по току при электролизе хлористого магния титанового производства [Текст] / К. Д. Мужжавлев, Н. А. Франтасьев, В. Г. Гопиенко и др. // Цветные металлы. — 1984. — № 8. — С. 62—64.
3. Свалов, Г. Н. Зависимость выхода по току магния от абсолютной влажности воздуха и срока службы электролизера с катодом-рамой при питании возвратным хлористым магнием [Текст] / Г. Н. Свалов, В. Н. Белов, Г. В. Олюнин, В. С. Чистякова // Электролитическое производство магния. — Запорожье. — 1982. — С. 53—63.
4. Яковлева, Г. А. О режимных параметрах электролитического получения магния [Текст] / Г. А. Яковлева, Ж. В. Пилецкая, Р. Г. Минина. — М. : Цветные металлы. — 2010. — № 8. — С. 55—58. — ISSN 0372-2929.
5. Олесов, Ю. Г. Влияние состава электролита на поведение низших хлоридов титана в атмосфере воздуха [Текст] / Ю. Г. Олесов, И. А. Баранник, В. В. Нерубащенко и др. // Вопросы химии и химической технологии. — 1982. — Вып. 67. — С. 33—36.
6. Баранник, И. А. Исследование катодного процесса при электролизе хлористого магния содержащего низшие хлориды титана [Текст] / И. А. Баранник, В. В. Волынский, Л. Н. Антипин // Украинский химический журнал. — Киев. — 1968. — С. 789—794.
7. Криворучко, Н. П. Экспериментальное изучение влияния влажности воздуха на осаждение частиц оксида магния и соосаждение соединений титана в расплавленном хлориде магния [Текст] / Н. П. Криворучко, Д. В. Бачурский, И. Ф. Червоный, В. Н. Михайлин // Металургія. — Запоріжжя, 2008. — Випуск 17. — С. 52—59.
8. Бачурский, Д. В. К вопросу о поведении примесей титана в электролитах магниевых электролизеров [Текст] / Д. В. Бачурский, И. Ф. Червоный, Н. П. Криворучко, Д. М. Хабров, Е. А. Матвеев, Е. П. Щербань // Теория и практика металлургии. — 2013. — № 2. — С. 17—24.
9. Пилипчук М. І. Основи наукових досліджень [Текст] : підручник / М. І. Пилипчук, А. С. Григор'єв, В. В. Шостак. — К.: Знання, 2007. — 270 с. — ISBN 966-346-248-5.
10. Адлер, Ю. П. Планирование эксперимента при поиске оптимальных условий [Текст] / Ю. П. Адлер, Е. В. Маркова, Ю. В. Грановский. — М. : Наука, 1976. — 280 с. — Библиогр. : по главам. — 4200 экз.
11. Спиридонов, А. А. Планирование эксперимента при исследовании технологических процессов [Текст] / А. А. Спиридонов. — М. : Машиностроение, 1981. — 184 с.
12. Налимов, В. В. Теория эксперимента [Текст] / В. В. Налимов. — М. : Наука, 1971. — 208 с.
13. Шенк, Х. Теория инженерного эксперимента [Текст] / Х. Шенк; перев. с англ. Е. Г. Коваленко. — М. : Мир, 1972. — 386 с.
14. Бойко, А. И. Методы Аналитического контроля в цветной металлургии. Том 6. Методы аналитического контроля в производстве титана и магния [Текст] : руководство / под ред. А. И. Бойко, Н. В. Галицкого, Т. А. Пампушко. — Москва. — 1983. — 230 с.

ОСАДЖЕННЯ $TiCl_{2(3)}$ У РОЗПЛАВІ KCl : NaCl : $MgCl_2$ В ЗАЛЕЖНОСТІ ВІД ВОЛОГОСТІ NaCl, ЩО ЗАВАНТАЖУЄТЬСЯ

Викладено результати досліджень процесу осадження сполук титану в розплаві солей, залежно від вологості NaCl, що подається на дзеркало розплаву. Склад розплаву відповідає електроліту, що застосовується при електролітичному виробництві магнію в потокових лініях. Приведено рівняння регресії, яке дозволяє визначити ступінь впливу вологості на процес осадження сполук титану, а також визначити кількість осадженого титану.

Ключові слова: електроліз магнію, потокова лінія, нижчі хлориди титану, факторний експеримент, вологість.

Бачурский Денис Васильевич, *аспирант, кафедра металлургии цветных металлов, Запорожская государственная инженерная академия,* ***e-mail: denis-bacho@yandex.ru.***

Бачурський Денис Васильович, *аспірант, кафедра металургії кольорових металів, Запорізька державна інженерна академія.*

Bachurskyy Denys, *Zaporizhzhya State Engineering Academy,* ***e-mail: denis-bacho@yandex.ru***

УДК 004.052.3

Литвин Т. Р.,
Сердюк П. В.,
Зачковська Х. О.

АВТОМАТИЗОВАНЕ РЕГРЕСІЙНЕ ТЕСТУВАННЯ ОБЧИСЛЮВАЛЬНИХ АЛГОРИТМІВ НА ОСНОВІ НЕЙРОННИХ МЕРЕЖ

Запропоновано метод підтримки автоматизованих тестів обчислювальних алгоритмів на основі нейронних мереж, який надає можливість оцінити правильність результатів обчислювальних алгоритмів після змін у функціональності програмного забезпечення. Описано математичну модель автоматизованого тестування з використанням нейронних мереж, розглянуто приклади застосування алгоритму.

Ключові слова: *автоматизоване тестування, нейронні мережі, алгоритм зворотного поширення помилки, регресійне тестування.*

1. Вступ

З удосконаленням процесів розробки програмного забезпечення та методологій їхнього ведення на різного роду проектах, однією з найактуальніших проблем на стадії розробки є гарантія надійності кінцевого продукту. Впровадження різносторонніх процесів валідації та верифікації результату у процесах тестування програмного забезпечення дає змогу довести кінцевий продукт до еталонного вигляду, забезпечити дотримання усіх необхідних вимог, поставлених перед командою розробників. З розвитком гнучких процесів ведення проектів розробки програмного забезпечення, процеси тестування набули певних властивостей, вимог і шаблонів, потребують окремого підходу до їхньої організації і проведення. На сьогоднішній день існує ціла низка видів тестування, впроваджених у різних сферах розробки програмного забезпечення, які відрізняються своїми властивостями, етапами проведення, методологією та засобами виконання, тощо. Усі процеси тестування потребують регресійного тестування [1, 2] на всіх ітераціях розробки, чи в період між будь-якими змінами у коді та функціоналі програмного забезпечення, тому що існує великий ризик виникнення прихованих помилок при змінах у коді програми [3, 4]. Фахівцям з тестування доводиться постійно налаштовувати системи автоматизованого тестування в умовах жорстких часових обмежень [5], для того, щоб збільшити обсяг перевірок і підвищити якість тестування, забезпечити можливість повторного використання тестових сценаріїв, а також для зменшення рутинної ручної роботи, застосовують засоби автоматизації тестування. У зв'язку з надто частими ітераційними змінами у процесі розробки програмного забезпечення важливим є питання впровадження підходу тестування, в основі якого лежить алгоритм, який приймає рішення про порушення у роботі функціоналу програмного забезпечення, при тому є адаптованим до змін у функціоналі і може використовуватись безпосередньо для регресійного тестування [3].

2. Автоматизація процесу тестування

Кожен процес тестування включає проходження тестових випадків [6], необхідних для покриття тестами потрібної частини функціоналу програмного забезпечення. Інакше кажучи, тестування ПЗ проводиться на відповідність наперед визначеним вимогам з функціональності, продуктивності, безпеки та ін., і з відповідно визначеними тестовими даними. Для заміни ручного тестування використовують підходи автоматизованого тестування програмного забезпечення. Рішення про дефект чи успішність проходження автоматизовані тести приймають на основі тестового оракулу, яким виступають assert подібні функції, які перевіряють відповідність між отриманим та очікуваним результатом, після проходження тесту) [1, 7].

В цілях автоматизації процесів тестування у даній роботі запропоновано адаптивний алгоритм визначення коректності обчислювальних процесів з використанням нейронних мереж функціоналу програмного забезпечення як основи тестового оракулу.

Плюси такого використання нейромережі зумовлені їхньою адаптованістю, можливістю самонавчання, тощо [8]. У роботі [9] розглянутий алгоритм використання нейронних мереж для прогнозування можливих помилок при генерації нових тестових даних для тест-кейсів. Автори запропонували алгоритм, який використовує навчену нейронну мережу в якості предиктора для визначення помилок у генерованих тестових даних для тестових сценаріїв. У роботі [10] розглянутий підхід з використанням нейронних мереж у якості предиктора у процесах тестування, а саме для визначення надійності роботи функціоналу програмного забезпечення при наявності помилок.

Вхідні параметри нейронної мережі, яка симулює поведінку програми, потребують певної абстракції, в залежності від специфіки програми. Так для представлення даних, які використовуються для тестування певного функціоналу програмного забезпечення, у вигляді вхідного вектора для нейронної мережі, — необхідна їхня нормалізація [9, 10, 11]. До цього часу були описані підходи використання нейромереж, як предиктора для аналізу надійності результатів і прогнозування у процесах тестування, проте застосування вище згаданих підходів може бути не дуже зручним, так як вони вимагають відомості про існування помилок в системі наперед [12], таким чином не вирішуючи основної проблеми регресії тестування, а саме виявлення нових помилок при ітераційних змінах у функціоналі ПЗ.

Досі не розглядались алгоритми з використанням засобів штучного інтелекту для виявлення самих помилок у функціоналі ПЗ під час тестування, тобто для перевірки коректності його роботи. Такі підходи можливі для навчання нейромережі в якості предиктора для аналізу правильності роботи функціоналу та локалізації помилок при певній абстракції даних. Вирішення такої задачі у даній статті зводиться до порівняння результатів обчислень навченої мережі з результатами роботи програми при однакових вхідних параметрах. Даний підхід побудови автоматизованих тестів програмного забезпечення може застосовуватись у регресійних процесах тестування при постійних змінах у функціоналі на різного роду проектах. Описаний у даній статті алгоритм здатний вирішувати задачі тестування, в яких є невідомі закономірності розвитку ситуації і залежності між вхідними та вихідними даними та локалізувати помилки у функціоналі. Для побудови інтелектуального тестового оракулу найефективнішим підходом є використання нейронних мереж [10, 13]. Нейронні мережі мають здатність навчатися на прикладах в тих випадках, коли невідомі закономірності розвитку ситуації, при невизначеній залежності між вхідними та вихідними даними, а також у випадках неповної, не точної і внутрішньо суперечливої вхідної інформації [14]. Завдяки цій можливості нейромережу можна навчити імітувати функціонал ПЗ, що тестується, використовуючи нормалізовані дані з тест-кейсів, які покривають процес тестування цього функціоналу. Таким чином тестовий оракул буде використовувати особливості достовірної робочої версії функціоналу у вигляді нейронної мережі для класифікацій результатів [13].

Загалом для імітації процесу навчання мережі та впровадження описаного процесу тестування зручним буде використання програмної реалізації алгоритму. Розробка спеціальної програми для симуляції тестового оракулу на основі нейромережі дасть можливість гарантувати правильність роботи тестового функціоналу та підвищити загальну якість процесу тестування складного програмного забезпечення.

Для реалізації процесу навчання нейронної мережі, який лежить в основі алгоритму, використовується метод зворотного поширення помилки [14, 15]. Його перевага в тому, що він може навчити всі прошарки нейронної мережі і його легко прорахувати локально. Використовуючи певні формули нормалізації даних, — дані з звичайних тест-кейсів можна абстрагувати та використовувати для навчання. Програмна реалізація алгоритму дасть можливість зробити необхідні розрахунки алгоритму при великих множинах навчання за відносно невеликі часові проміжки.

В кінцевому результаті тестовий оракул на основі нейронних мереж, насамперед, допоможе зробити процес тестування продуктивнішим та якіснішим для складного функціоналу ПЗ, яке потребує постійної регресії та контролю, затратних обчислень, тощо.

3. Структурна модель тестового оракулу

Наявність множини тест кейсів, які покривають функціонал ПЗ призвела до появи класифікаторів тестових результатів: «False positive», «False negative», «True positive», «True negative» [13]. Ця інтерпретація зумовлена наявністю прихованих помилок у програмному продукті. Для їх локалізації використовують тестовий оракул, який лежить в основі описаного у даній роботі алгоритму оцінки коректності виконання тестів в процесі тестування ПЗ.

Тестові оракули можуть бути сформульовані виходячи з [13]:

— Специфікацій та документації (визначаємо очікувані результати на підставі специфікацій);

— Аналогічних продуктів (визначаємо правильну поведінку системи в порівнянні з аналогічними продуктами конкурентів);

— Статистичної оцінки (визначаємо очікуваний результат, на підставі статистики ґрунтуючись на статистиці по раніше випущеним продуктам);

— Евристичної оцінки (визначаємо очікуваний результат, на підставі можливості його виникнення);

— Повторної перевірки (визначаємо очікуваний результат, на підставі проведення одного і того ж тесту кілька разів, звіряючи отримані результати);

— Раціонального мислення (визначаємо очікуваний результат, використовуючи наше розуміння раціональності і правильності результату проведення тесту, що дуже схоже на adhoc тестування);

— Досвіду інженера, що проводить тестування.

Повний оракул повинен мати три особливості: забезпечення прогнозованих, або очікуваних результатів для кожного тесту, компаратор для порівняння передвіщених і отриманих результатів, визначати, чи є отримані результати такі, щоб вважати тест пройденим успішно [16].

Відповідно до специфікації кількість тест-кейсів може змінюватись на подальших ітераціях розробки з імплементацією нових змін та модифікацією існуючих зі зміною вимог у функціоналі, що тестується [6, 17]. Формально, тест кейс — це послідовність дій за якою можна перевірити, чи відповідає функціонал, що тестується, певним вимогам [6]. Процес проходження тест кейсу t у процесі тестування ПЗ представимо у вигляді моделі (рис. 1) [6, 17].

Кроки виконання тест кейсу t проводяться у порядку розташування їх у множині кроків S. Етап верифікації i-того кроку для кожного тест кейсу t проводиться після виконання самого кроку у процесі тестування та вибирається з множини етапів верифікації V теж у порядку розташування елементів множини, згідно того, що розмірності множин рівні. $V_i - i$-тий елемент етапу верифікації може теж бути множиною виконання дій та перевірки даних, тобто для верифікації виконання i-того кроку тест кейсу потрібно перевірити, наприклад, правильну роботу чи поведінку декількох речей у функціоналі.

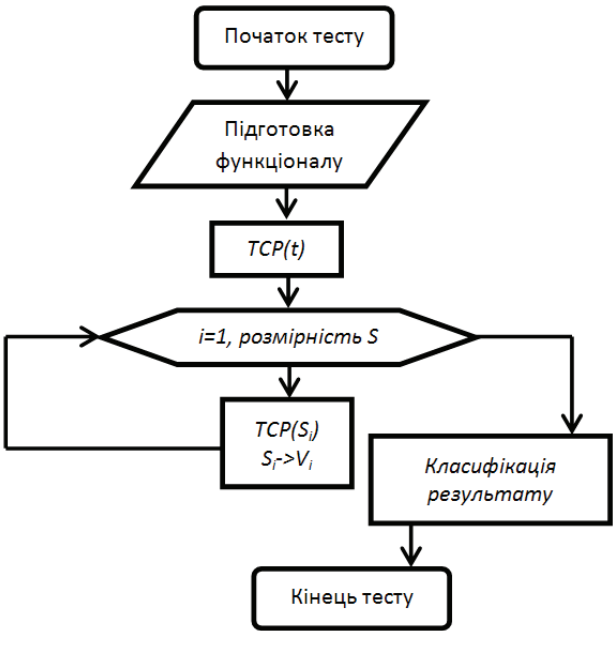

Рис. 1. Модель процесу проходження тест кейсу

Для даної моделі справедливі наступні твердження [17]:

TCE (Test case execution) — виконання тест кейсу, можна подати у вигляді функції (відображення), яка діє на кроки тест кейсу згідно рис. 1, тобто:

$$TCE(t): S_i \to V_i,$$

TCP (Test case precondition) — множина дій, які приводять систему до придатного стану для проведення основної перевірки певним тест кейсом. *TCP* для певного кроку тест кейсу — це множина дій, які приводять систему, що тестується до придатного стану для виконання даного кроку і передбачає виконання усіх попередніх кроків тест кейсу, тобто:

$$TCP(S_i): S_{i-1} \to V_{i-1_i}, ..., S_1 \to V_1, \; i \neq 1.$$

$TCP(t)$ означає, що схема на рис. 1 для тест кейсу t виконується лише в тому разі, якщо встановленні всі необхідні передумови для виконання $TCP(S_1): S_1 \to V_1$.

Використання заздалегідь з'ясованих, очікуваних результатів, є ключовим фактором успіху виконання *TCE*, для того чи іншого тест кейсу [17]. Це твердження впливає на загальні результати тестування і на якість продукту. Тестовий оракул є певним висновком до етапів верифікації, підсумковим критерієм оцінки роботи функціоналу, який тестувався. Сутність тестового оракулу для того тест кейсу t можна подати у такому вигляді:

1. Виконується *TCP* для цілого тест кейсу t (якщо це необхідно).

2. Виконується *TCE* для t (рис. 1).
3. Реалізовується алгоритм для визначення класифікаторів тестового результату.

У новому підході реалізації тестового оракулу для функціоналу, що буде тестуватись, розглянуто використання нейронних мереж. В дійсності, нейронна мережа після процесу навчання стає імітованим симулятором моделі функціоналу, що тестується (емулятором). Коли розробляються нові версії програмного забезпечення і регресійне тестування стає необхідним, в функціонал системи підставляються тестові дані, для отримання вихідних результатів, які порівнюються з результатами роботи мережі. Вважається, що нові версії програмного продукту не впливають на зміну існуючих функцій, що означає, що система здатна видавати однакові вихідні результати при повторному тестуванні на однакових тестових даних, тобто нові модифікації існуючої системи не повинні стосуватись частини функціоналу, який буде тестуватись за допомогою нижче описаного алгоритму. За допомогою методу порівняння, визначається, чи вихідні результати функціоналу, що тестується, є коректними чи ні. Таким чином, виконання *TCE* може здійснюватись нейронною мережею самостійно, після закінчення процесу навчання, а алгоритмом визначення класифікаторів для тестового результату служить — метод порівняння. Загальну концепцію усього процесу зображено на рис. 2.

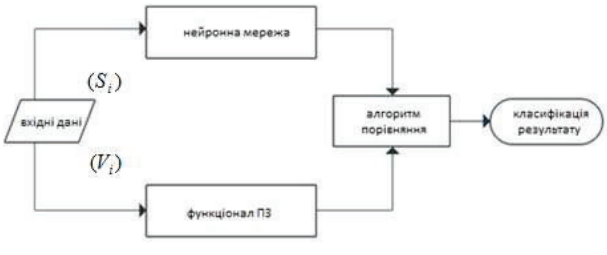

Рис. 2. Концепція тестового оракулу

Використання нейромережевої моделі, яка служить емулятором функціоналу продукту може мати низку переваг. Оригінальна версія програмного продукту не застережена від наявності прихованих дефектів, які були відсутні у старіших версіях, що були випущені на стадіях розробки. Нейронна мережа вимагає додатковий параметр — активаційну функцію, яка пов'язана з вихідним вектором мережі (результатом роботи програми), яка дає змогу оцінити його надійність. Під час етапу навчання, синаптичні ваги мережі змінюються при представленні нових множин даних в алгоритм навчання. Множина навчальних даних складається з векторних пар, які подається на вхід та на вихід мережі і являють собою нормалізовані, певним способом, дані з тест-кейсів. Отже, після завершення процесу навчання мережі — вона працює у вигляді частини функціоналу програмного забезпечення, що тестується та приймає вхідні вектори даних для емуляції, розрахунку вихідного результату. У ролі вхідного вектора використовуються дані з множини S, а у ролі вихідного — відповідні їм дані з верифікаційної множини V, з збереженням усіх сутностей та визначених раніше умов (рис. 3). Достовірність результату роботи частини функціоналу програми безпосереднього визначається з порівняння результатів роботи цього функціоналу ПЗ з результатами роботи попередньо навченої мережі, яка його симулює.

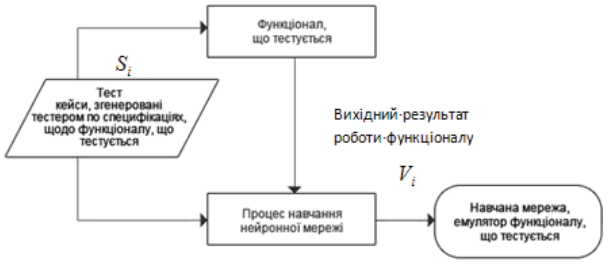

Рис. 3. Концепція фази навчання

Класифікація результату. Реалізований процес тестування є тестуванням за стратегією «чорного ящика», тобто без доступу до вихідного програмного коду функціоналу [18].

Оцінка результату роботи функціоналу, що тестується, відбувається після проходження відповідного тест-кейсу. Дані з тест-кейсу представляються у вигляді вхідного вектора у функціонал, що тестується, нової версії програмного забезпечення та для навченої нейронної мережі. Для кожного вектора, розраховується абсолютна різниця між вихідним нейроном переможцем та відповідним вихідним результатом програми. Дана різниця передається у метод порівняння для класифікації результату. Всі вихідні результати — це числа, що лежать у проміжку [0, 1]. Розрахована різниця поміщається в один з трьох інтервалів, що визначаються двома граничними значеннями на проміжку [0, 1]. Якщо прогноз нейронної мережі і результат виконання тест-кейсу програмою дають практично ідентичний результат, а розрахована різниця потрапляє у проміжок між 0 та низьким граничним значенням — це означає, що результат роботи мережі відповідає результату роботи функціоналу ПЗ і обидва вихідні результати вважаються правильними. У цьому випадку метод порівняння класифікує правильний результат роботи тієї частини програми, що тестувалась та тест кейс вважається пройденим успішно. З іншого боку, якщо результат роботи мережі відрізняється від результату роботи програми і розрахована різниця знаходиться у проміжку між великим граничним значенням та 1, метод порівняння класифікує роботу функціоналу, як помилку і тест кейс вважається не пройденим. У двох випадках, результат роботи мережі є коректним, що забезпечує певну надійність в оцінюванні вірності результату. Проте, якщо значення різниці, що розраховується методом порівняння, поміщається у інтервал між низьким та високим граничним значенням, то вихідний результат нейронної мережі вважається ненадійним. У цьому випадку, у функціоналі, що тестується, буде помилка, тільки тоді, якщо результат мережі співпадає з вихідним результатом програми [10, 11].

Метод порівняння використовується у якості незалежного методу, який порівнює результат нейронної мережі, яка імітує функціонал, описаний тест-кейсами з результатами роботи цього функціоналу у самій програмі. Різниця між цими результатами визначається, як різниця по модулю між результатом нейрона переможця і відповідним результатом роботи програми. Так, як для забезпечення вихідного результату мережі використовується сигмоїдальна активаційна функція,

то активаційне значення нейрона переможця лежить у проміжку між 0 та 1. Відповідне значення результату роботи функціоналу програми рівне 1, якщо прогнозований та отриманий результати рівні, у іншому випадку він рівний 0. Таким чином різниця покриває проміжок від 0 та 1 і ми використовуємо її значення для визначення чи програма згенерувала коректний результат чи помилковий (1).

$$\begin{aligned} &0 \le |y_{winner} - p| \le 1 \\ &0 \le y_{winner} \le 1 \\ &p = \begin{cases} 0, & predicted \ne actual \\ 1, & predicted = actual, \end{cases} \end{aligned} \quad (1)$$

— y_{winner} — значення нейрона переможця;

— p — модифіковане вихідне значення програми.

Табл. 1 демонструє чотири можливі категорії класифікації підсумкового результату тестового оракулу. Результат оцінки можна помістити в один з приведених класифікаторів для визначення його сутності. Так, як нейронна мережа — це лише імітація актуальної системи, — деякі з її вихідних результатів можуть бути хибними. Якщо вихідний результат нейронної мережі є коректним і при цьому відповідний результат роботи функціоналу програми є некоректним, метод порівняння класифікує результат, як True negative, що означає, що у програмі помилка. Аналогічно три інші значення з таблиці представляють інші види класифікаторів. Кожен вихідний результат, отриманий з нейронної мережі та з програми оцінюється відповідно.

Таблиця 1

Класифікація підсумкового результату

	Результат системи, що тестується	
Результат нейронної мережі	Коректний	Некоректний
Коректний	1 True Positive	2 True Negative
Некоректний	4 False Negative	3 False Positive

Два типи вихідних результатів обробляються через метод порівняння по різному, так як для бінарних вихідних результатів є чотири можливих класифікатори результату, а для неперервних — п'ять (табл. 2).

Таблиця 2

Можливі варіанти результату

	Порівняння результатів	
Тип вихідного результату	Однакові	Різні
Бінарний	Обидва коректні Обидва некоректні	Вихідний результат нейронної мережі коректний Вихідний результат програми коректний
Неперервний	Обидва коректні Обидва некоректні	Вихідний результат нейронної мережі коректний Вихідний результат програми коректний Обидва некоректні

Опис тестової методології. Алгоритм впровадження описаної тестової методології фактично можна розділити на чотири етапи:

1. Генерування тест-кейсів для функціоналу, що тестується тест-інженером (тестером, QA) [17];
2. Виконання *TCE* для генерованих тест-кейсів у самій програмі тест-інженером [17, 18];
3. Конструювання та навчання нейронної мережі за допомогою даних з генерованих тест-кейсів [14, 15];
4. Проведення регресивного тестування (виконання кроку 2 через програму та нейронну мережу) — класифікація результату через метод порівняння [2, 9, 10, 11].

(S_i) — вхідні атрибути, кроки тест-кейсів генеруються згідно з специфікацій програми, (V_i) — етапи верифікації, можуть бути визначенні як у процесі аналізу специфікацій, так і розраховані в процесі тестування.

4. Практичні результати автоматизації тестування на основі нейромережі

Даний алгоритм був застосований для проведення процесу регресивного тестування для однієї з частин функціоналу Win-desktop-ної програми для розрахунку виплат (Pay Calculator).

Перший тест кейс (табл. 3) демонструє випадок, коли єдиному кроку з множини кроків тест-кейсу відповідає етап верифікації, який включає цілу множину перевірок і якщо один з елементів множини не пройде перевірку при тестуванні — тест-кейс вважається не пройденим і отримує статус fail. може приймати лише два значення істинності (Pay Calculator відкритий та невідкритий), а буде істиною при перевірці лише, коли всі перевірки з приймають істинне значення, тобто теж покривається двома варіантами істинності у даному випадку. Застосовувати нейронну мережу тут не актуально, так як при виконанні лише один варіант виконання (= true, = true) робить тест-кейс пройденим успішно на будь-якому етапі регресійного тестування.

Щодо t_2, то тут кількість тестових випадків може бути суттєво великою. Незалежно від того, що було введено у перших трьох кроках при *TCE*, — четвертий етап визначення коректності відображення Gross Pay прямо залежить від попередніх трьох кроків *TCE*, тому виконання четвертого етапу — V_4 залежить від $TCP(S_4)$. Як і у попередньому тест-кейсі успішність $TCP(S_4)$ для другого тест-кейсу гарантує лише варіант, коли для всіх пар $S_i, V_i, i = \overline{1,3}$ виконується (S_i = true, V_i = true) в плані виконання операцій описаних у кроках. Так як у даному тест-кейсі потрібно перевірити відображення на формі інтерфейсу елементів при різних маніпуляціях, то актуальність введення нейронної мережі є, проте навчання і генерування тестових випадків у даному сенсі, через бінарні значення елементів V_i та S_i, буде зайвою. Тому розглянемо нейронну мережу для алгоритму тестового оракулу для визначення верифікації етапу «Gross Pay повинен бути порахований правильно».

Відповідно перевірка на недопустимі значення вводу відбувається окремо, що являє собою проходження негативних тест кейсів.

Для правильного функціонування нейронної мережі, вхідні дані, взяті з кроків тест кейсу і відповідні вихідні значення з етапів верифікації, потрібно нормалізувати до значень між 0 та 1. Попередня обробка даних здійснюється за формулою [10, 15]:

$$Normalized\{P(S_i)\} = \frac{Original\{P(S_i)\} - \min\{P(S_i)\}}{\max\{P(S_i)\} - \min\{P(S_i)\}}, \quad (2)$$

де $P(S_i)$ — вхідні дані тест-кейсу, взяті для кроку S_i; $\min\{P(S_i)\}$ — мінімальне значення вектора вхідних даних взятих для S_i кроку тест-кейсу; $\max\{P(S_i)\}$ — максимальне значення вектора вхідних даних, взятих для S_i кроку тест-кейсу.

Вихідні дані ж трактуються різним чином. Для неперервних вихідних атрибутів, взятих з етапів верифікації $P(V_i)$ вводять інтервали, кожен з яких вибирають таким чином щоб він містив однакову кількість записів. Бінарні ж вихідні дані відповідно будуть поділятись лише на два інтервали для 0 та 1 відповідно.

Таблиця 3

Тест кейси для тестового функціоналу

ID (№)	Назва Тест Кейсу	Множина кроків S	Множина етапів верифікації V
t_1	Pay Calculator View Layout	1. Відкрити Pay Calculator	1. Pay Calculator повинен запуститись з такими елементами форми: — Надпис PayCalculator повинен бути присутній та кнопка Calculate. — Текст бокси Regular Hours, Age, Rate Of Pay повинні бути присутні. — Поле розрахунку Gross Pay повинно бути присутнє
t_2	Gross Pay View	1. Ввести у поле Regular Hours значення. 2. Ввести у поле Age значення. 3. Ввести у поле Rate of Pay значення. 4. Натиснути елемент управління Calculate	1. Regular Hours повинно відобразитись. 2. Age повинно відобразитись. 3. Rate of Pay повинно відобразитись. 4. Gross Pay повинен бути порахований правильно

Після попередньої обробки дані можна використовувати для процесу навчання, щоб отримати мережу, яка буде здатна «повторювати» поведінку функціоналу програми, що тестується, з достатньою точністю. Після навчання можна робити оцінки правильності обрахунку Gross Pay програмою та визначити чи версія є несправною та чи містить дефекти. Генерована множина вхідних векторів (табл. 4) підставляється у мережу та у необхідний для тестування функціонал нової програмної версії.

Таблиця 4

Приклад вхідних даних перед

поле Age	поле Rate of Pay	поле Regular Hours	Gross Pay розрахунок (вихідний результат програми)
55	57	49	1851,281250
38	37	59	1355,289062
18	27	43	467,883026
57	53	64	2534,063965
44	45	43	1195,700928

Дані в табл. 5 отримані за допомогою формули (2).

Архітектура нейронної мережі також залежна від вхідних параметрів. Відповідно кількість нейронів вхідного шару рівна розмірності множини даних, які беруться з кроків тест-кейсу, що покриває тестовий функціонал; кількість нейронів прихованого шару можна визначити за допомогою методу Арнольда — Колмогорова — Хехт-Нільсена [19]; кількість нейронів вихідного шару (кількість інтервалів) залежить від типів результатів самої програми, розмірності можнини даних, які беруться з множини верифікаційних етапів тест кейсу та від розміру навчальної множини. Юніт-інтервал вихідного шару з найбільшим значенням розглядається як нейрон переможець. Початкові значення синаптичних ваг, випадковим чином заповнюються значеннями –0,5 та 0,5, коефіцієнт навчання рівний 0,5. Із збільшенням навчальної множини середньоквадратична похибка (СКП) мережі під час навчання зменшується (рис. 4).

Таблиця 5

Нормалізовані вхідні дані

поле Age	поле Rate of Pay	поле Regular Hours	Gross Pay розрахунок (вихідний результат програми)
0,897436	1	0,285714	{0,0,0,1,0}
0,512821	0,333333	0,761905	{0,0,1,0,0}
0	0	0	{1,0,0,0,0}
1	0,866667	1	{0,0,0,0,1}
0,666667	0,6	0	{0,1,0,0,0}

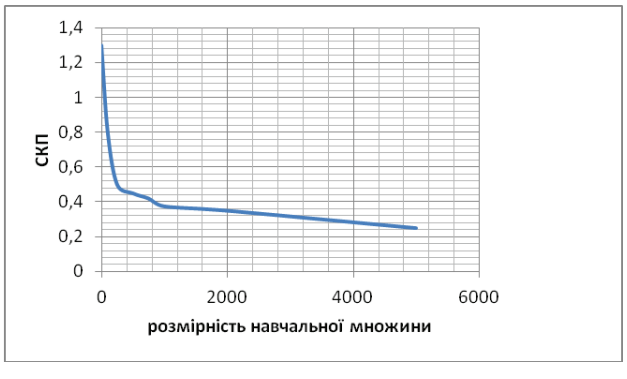

Рис. 4. Середньоквадратична похибка мережі під час навчання

5. Висновок

Даний алгоритм ресурсозатратний, тому для адекватної поведінки мережі, у різних випадках застосування, епоха навчання може досягати проходження кількох тисяч тестових даних (рис. 5). Беручи до уваги ресурсні затрати даного алгоритму, постає питання його програмної реалізації для складних проектів із десятками тисяч автоматизованих тестів. В результаті розвитку інформаційних технологій, сучасні процесори дозволяють достатньо швидко обробляти та розпаралелювати обробку великих об'ємів інформації. Для моделювання описаного алгоритму тестування потрібно врахувати параметри тест-кейсів, які користувач може заносити вручну в графічний інтерфейс програми, або які можуть динамічно зчитуватись з файлів та запрограмувати алгоритм конструювання та навчання мережі.

Розроблене програмне забезпечення забезпечить змогу підвищити якість тестування функціоналу, який затратно тестувати вручну повторно та який вимагає постійного регресійного контролю. Програмний продукт відображатиме візуалізацію процесу конструювання нейронної мережі, її навчання, порівняння результатів,

класифікацію результату, надаватиме можливість виведення графіків середньоквадратичної похибки мережі, статистику локалізації помилок алгоритмом при різних версіях ПЗ, завдяки чому можна буде локалізувати наявність помилок у системі, що тестується.

При тестуванні програми з помилками високого пріоритету, за допомогою запропонованого алгоритму, були отримані наступні результати: з тестової вибірки у 1000 тестових випадків — 93,1 % вихідних результатів були пораховані коректно, з яких невірно класифіковані були лише 1,61 %.

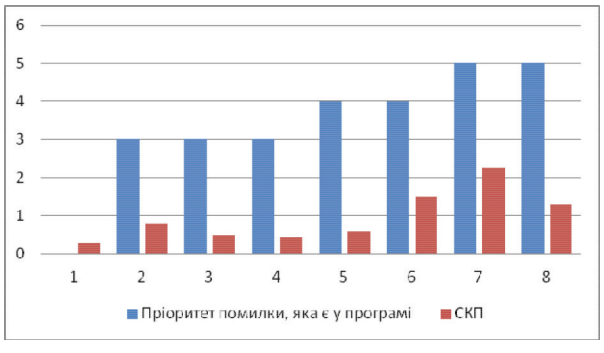

Рис. 5. Середньоквадратична похибка мережі під час тестування

Література

1. Dustin, E. Automated software testing: introduction, management, and performance [Text] / E. Dustin // Addison Wesley. — 1999. — 575 c.
2. Савин, Р. Тестирование «Dot Com» [Текст] / Р. Савин // Изд. дом «Дело». — 2007. — 310 с.
3. Говорущенко, Т. О. Інтелектуальна система визначення необхідності повторного тестування програмного забезпечення [Текст] / Т. О. Говорущенко // Искусств. интеллект. — 2006. — № 4. — С. 706—712.
4. Яковина, В. Критерій достатності процесу тестування програмного забезпечення [Текст] / В. Яковина, М. Сенів, Я. Чабанюк, Д. Федасюк, У. Хімка // Вісн. Нац. ун-ту «Львів. політехніка». Комп'ют. науки та інформ. технології. — 2010. — № 672. — С. 346—358.
5. Буров, Є. В. Інтелектуальна система автоматизованого тестування програмного продукту з використанням алгоритмічних моделей [Текст] / Є. В. Буров // Вісн. Нац. ун-ту «Львів. політехніка». Інформ. системи та мережі. — 2011. — № 699. — С. 21—30.
6. Булат, А. Написание Тестовых случаев [Електронний ресурс] / А. Булат // Блог А. Булата Про Тестинг. — 2008. — Режим доступу: http://alexeybulat.blogspot.com/2008/02/test-case-writing.html.
7. Automated-testing Portal (Портал автоматизации ПО [Електронний ресурс]. — Режим доступу: http://automated-testing.info.
8. Haykin, S. Neural Networks, A Comprehensive Foundation [Text] / S. Haykin // Prentice Hall, Upper Saddle River. — New Jersey 07458. — Second Edition. — 2006. — 1103 c.
9. Anderson, C. On the Use of Neural Networks to Guide Software Testing Activities [Text] / C. Anderson, A. Mayrhauser, R. Mraz // Colorado State University, Computer Science Department. — 1996. — С. 1—10.
10. Vanmali, M. Using a neural Network in the Software Testing Process. [Text] / M. Vanmali, M. Last, A. Kandel // International Journal of Intelligent systems. — Vol. 17. — 2002. — Department of Computer Science and Engeineering, University of South Florida, 4202 E. Fowler Ave., ENB 118, Tampa, FL 33620. — 17 p.
11. Lilan, Wu. Using back-propagation neural networks for functional software testing [Text] / Wu. Lilan // Sch. of Math. & Comput. Sci., Guizhou Normal Univ., Guiyang. — 2008. — Pp. 1—5.
12. Яковлев, И. А. Автоматизация тестирования и контроля качества систем оптического распознавания символов с применением нейронных сетей [Електронний ресурс] / И. А. Яковлев // Московский Государственный Университет Приборостроения и Информатики. — Режим доступу: http://www.sworld.com.ua/konfer22/260.htm.
13. Рыбалко, Г. Псевдопозитивный или псевдонегативный оракул [Електронний ресурс] / Г. Рыбалко // QAConsulting. — 2011. — Режим доступу: http://qaconsulting.ru/2011/01/test-oracle.
14. Копосов, А. И. Использование нейропарадигмы «Back Propagation» для решения практических задач [Текст] / А. И. Копосов, И. Б. Щербаков, Н. А. Кисленко, О. П. Кисленко, Ю. В. Варивода // ВНИИГАЗ. — 1995. — С. 1—7.
15. Chakraborty, R. C. Back Propagation Network [Електронний ресурс] / R. C. Chakraborty // Soft Computing Course Lecture. — 2010. — Режим доступу: http://www.myreaders.info/html/soft_computing.html.
16. Kaner, C. A Course in Black Box Software Testing [Text] / C. Kaner, J. Bach, M. Bolton, R. Fielder, M. Kelly // Florida Institute of Technology. — 2006. — 208 c.
17. Enge, C. Verification Based Test Case Generation [Text] / C. Enge // ITI, Universit at Karlsruh. — 2006. — С. 1—25.
18. Бейзер, Б. Тестирование черного ящика. Технологии функционального тестирования программного обеспечения и систем [Текст] / Б. Бейзер // Питер. — 2004. — 320 с.
19. Ясницкий, Л. Искусственный интеллект — проектирование персептронов [Електронний ресурс] / Л. Ясницкий // Пермская научная школа искусственного интеллекта. — Режим доступу: http://www.LbAI.ru.

АВТОМАТИЗИРОВАННОЕ РЕГРЕССИОННОЕ ТЕСТИРОВАНИЕ ВЫЧИСЛИТЕЛЬНЫХ АЛГОРИТМОВ НА ОСНОВЕ НЕЙРОННЫХ СЕТЕЙ

Предложен метод поддержки автоматизированных тестов вычислительных алгоритмов на основе нейронных сетей, который позволяет оценить правильность результатов вычислительных алгоритмов после изменений в функциональности программного обеспечения. Описана математическая модель автоматизированного тестирования с использованием нейронных сетей, рассмотрены примеры применения алгоритма.

Ключевые слова: автоматизированное тестирование, нейронные сети, алгоритм обратного распространения ошибки, регрессионное тестирование.

Литвин Тарас Романович, *асистент, кафедра дискретного аналізу та інтелектуальних систем, Львівський національний університет ім. І. Франка,* ***e-mail: tlytvyn@lohika.com.***
Сердюк Павло Віталійович, *кандидат технічних наук, доцент, кафедра програмного забезпечення, Національний університет «Львівська політехніка»,* ***e-mail: pavlo.serdyuk@gmail.com.***
Зачковська Христина Орестівна, *аспірант, кафедра програмного забезпечення, Національний університет «Львівська політехніка»,* ***e-mail: lystykhrystyny@gmail.com.***

Литвин Тарас Романович, *ассистент, кафедра дискретного анализа и интеллектуальных систем, Львовский национальный университет им. И. Франко.*
Сердюк Павел Витальевич, *кандидат технических наук, доцент, кафедра программного обеспечения, Национальный университет «Львовская политехника».*
Зачковская Кристина Орестовна, *аспирант, кафедра программного обеспечения, Национальный университет «Львовская политехника».*

Lytvyn Taras, *Ivan Franko Lviv National University,* ***e-mail: tlytvyn@lohika.com.***
Serdyuk Pavlo, *Lviv Polytechnic National University,* ***e-mail: pavlo.serdyuk@gmail.com.***
Zachkovska Khrystyna, *Lviv Polytechnic National University,* ***e-mail: lystykhrystyny@gmail.com***