UDC 004.032.26

RESEARCH AND IMPLEMENTATION OF NEURAL NETWORKS

Samalikov Farabi
master's student, Department of Computer Engineering, Astana IT University, Astana, Kazakhstan

Oralbekova Zhanar,.
PhD in Computer Science, associate professor, the Department of Computational and Data Sciences, Astana IT University, Astana, Kazakhstan

This article explores the application of neural networks in software testing, detailing their theoretical underpinnings, practical implementation, and innovative approaches. It highlights how neural networks can significantly enhance software testing processes by automating functions and improving fault detection capabilities. Empirical validation through a case study on credit card approval software demonstrates the effectiveness and efficiency of neural network methodologies. The article also addresses potential challenges such as data quality, computational resource demands, interpretability, and overfitting, providing a comprehensive overview of neural networks' transformative potential in software engineering.

Keywords: neural networks, software testing, automated oracle, fault detection, generative algorithms, machine learning, empirical validation.

Introduction

Neural networks, sophisticated computational structures inspired by the biological neural systems found within the human brain, have progressively become integral components across various technological domains. Initially making substantial impacts in fields such as healthcare, finance, and transportation, neural networks have more recently emerged as vital tools in software engineering, particularly software testing. Their adaptive learning capabilities, ability to recognize complex patterns, and proficiency in processing vast datasets position them uniquely to transform traditional software testing methodologies.

In the broader context of technological innovation, the integration of advanced artificial intelligence methods, including generative models, continues to reshape diverse sectors, including education and software development. Generative AI technologies—comprising natural language processing, machine learning, and neural networks—have demonstrated their transformative power by offering personalized experiences, enabling adaptive feedback, and identifying complex patterns across large datasets. Their role in software testing similarly opens new horizons for

customizing testing processes, optimizing test coverage, and detecting faults that traditional methods might overlook.

However, with the advancement of AI-driven methodologies, new challenges arise. Ethical considerations, including the transparency and accountability of AI decisions, become critical when AI models influence significant outcomes, such as software validation. Data privacy, algorithmic bias, and equitable access to AI-enhanced testing tools also emerge as crucial topics. Developers and testers must therefore be equipped not only with technical proficiency but also with a deep understanding of digital ethics, data security, and AI reliability.

This article synthesizes comprehensive theoretical foundations alongside practical implementation insights derived from research conducted at Astana IT University. It specifically examines the revolutionary role neural networks can play within software testing, emphasizing their potential in automating oracle functions and enhancing fault detection capabilities. Through detailed exploration and empirical evaluation, this article highlights the innovations neural networks bring to software testing processes, ultimately aiming to enhance accuracy, efficiency, and adaptability in testing practices.

Theoretical Foundations of Neural Networks

Neural networks comprise interconnected computational units or neurons designed to emulate biological neurons' synaptic connections. Key architectural components include input, hidden, and output layers, employing activation functions such as sigmoid, ReLU, or Softmax, which determine neuron activation states based on inputs.

The backpropagation algorithm stands out as the cornerstone for training multilayer feedforward networks. By iteratively adjusting synaptic weights, neural networks minimize the error between predicted and actual outputs. Challenges such as local minima in error surfaces are mitigated through techniques like cross-validation, hyperparameter tuning, and network retraining with varied initializations.

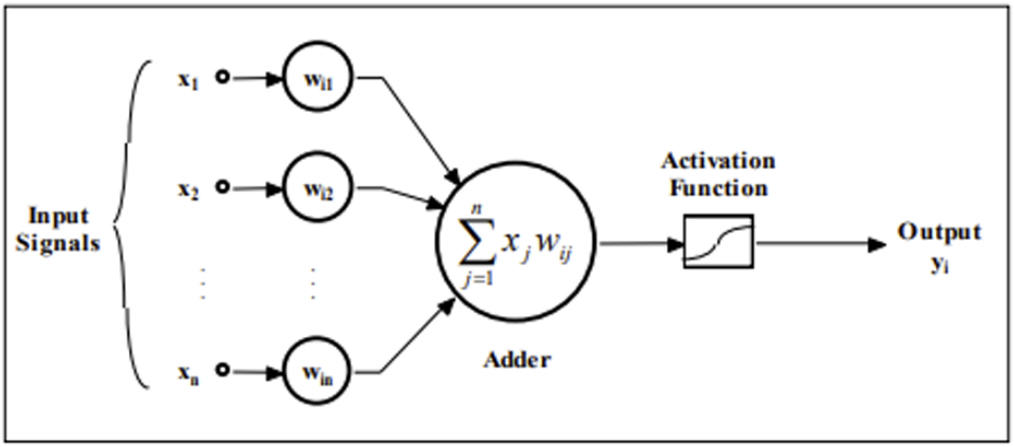

Fig.1

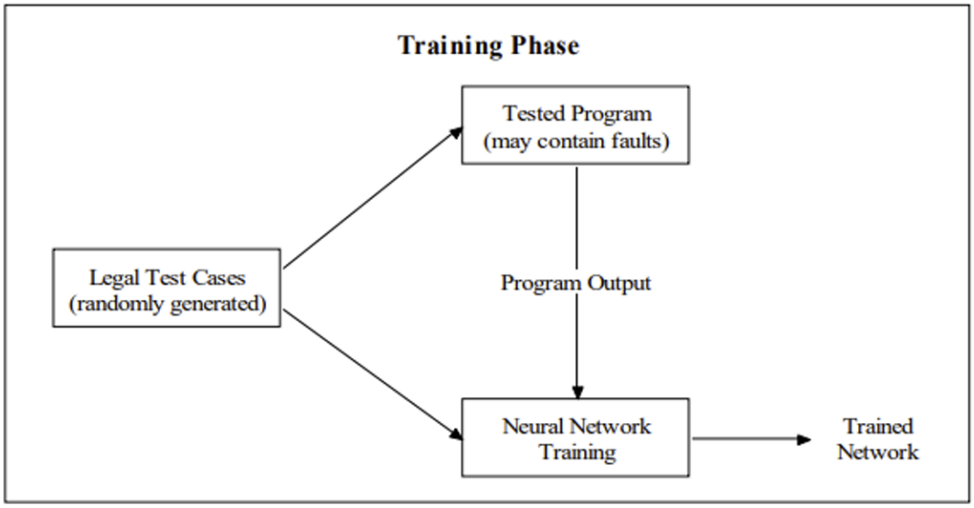

Fig.2

Software Implementation and Methodology

Utilizing frameworks like TensorFlow and PyTorch, neural network models are developed, optimized, and evaluated against diverse datasets. Effective data preprocessing, including normalization and augmentation, ensures high-quality inputs, enhancing model accuracy and generalization capabilities.

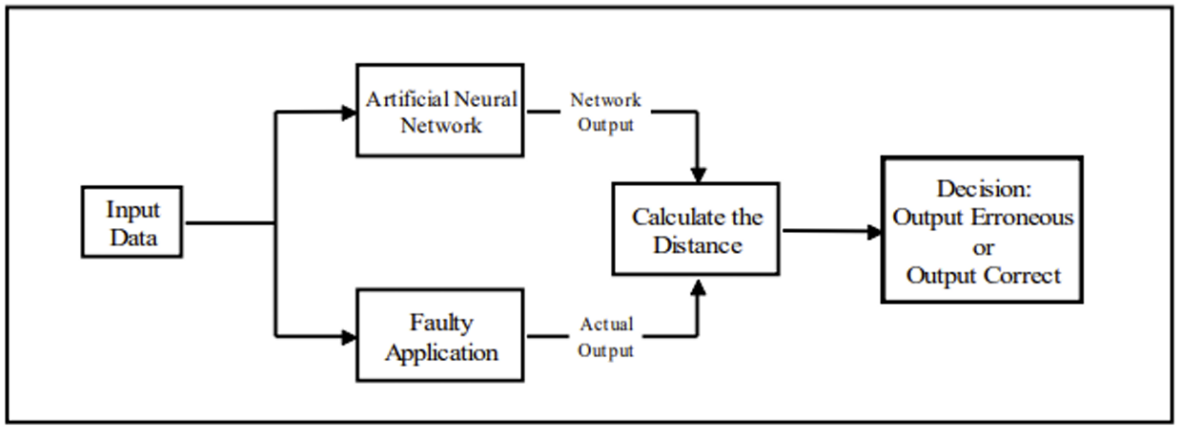

Fig.3

Innovative Approach: Neural Networks in Software Testing The uploaded paper outlines an innovative methodology employing neural networks as automated oracles within software testing. Traditionally, test oracles verify the correctness of software outputs against expected results. However, manual oracles are labor-intensive and error-prone, while automated oracles often lack adaptability to new software versions.

This research introduces neural networks trained on original software versions' outputs using random, specification-conforming test data. By comparing outputs of new software versions against neural network predictions, the methodology

effectively identifies discrepancies indicative of faults. This approach significantly reduces testing costs and improves accuracy, especially during regression testing.

Integration of Generative Algorithms

Further enhancing the robustness and generalization of neural models, generative algorithms like GANs (Generative Adversarial Networks) and VAEs (Variational Autoencoders) were proposed for data augmentation. These algorithms produce synthetic yet realistic data samples, addressing dataset limitations and improving neural network performance.

Fig.4

Results and Insights

During experimental validation, a variational autoencoder (VAE) model was trained on the processed credit approval dataset. The VAE architecture was optimized using the Adam optimizer, and training proceeded over 30 epochs with a batch size of 128. As illustrated in the training log, the loss function demonstrated consistent convergence, decreasing steadily from an initial value of 176.88 to 156.79 by the thirteenth epoch. This indicates the model's ability to effectively learn the underlying data distribution and reconstruct the input space with reduced error.

Additionally, time series forecasting was employed to predict trends based on fault detection data using various smoothing models such as the Brown Model, Holt-Winters Model, and Trigg-Lich Model. Forecasting results, visualized through a plotted graph, revealed an upward trend in model performance and fault detection accuracy over time. The Holt-Winters model, incorporating both trend and seasonal components, proved particularly effective in adapting to complex periodic data patterns, offering valuable predictive insights for continuous monitoring of software quality.

These results underscore the potential of combining neural network models with time series analysis to enhance fault prediction, optimize maintenance schedules, and ensure sustained reliability in evolving software systems.

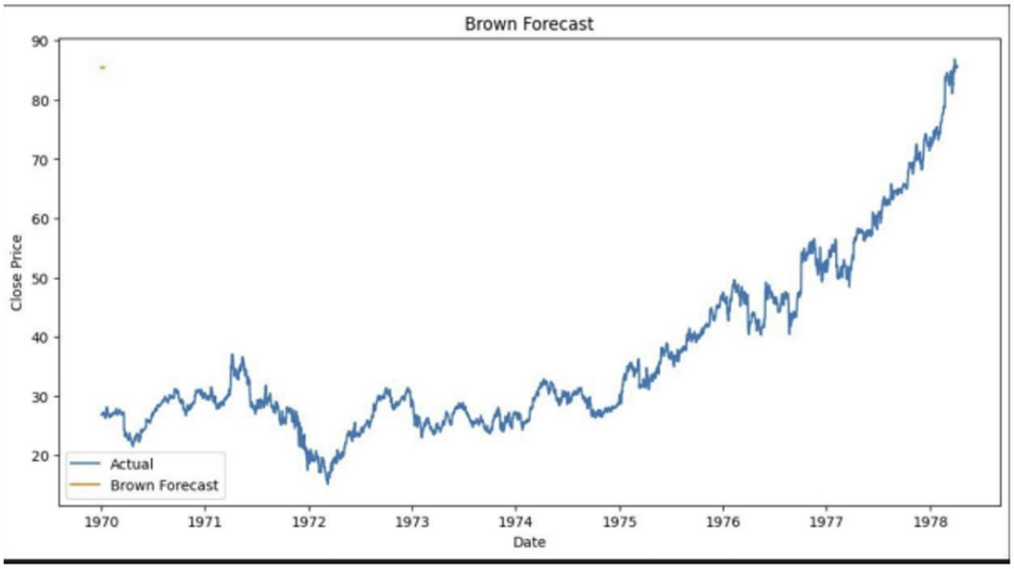

Fig.5

```python
latent_dim =2
vae=VAE(latent_dim)
optimizer = tf.keras.optimizers.Adam()
@tf.function
def train_step(x):
    with tf.GradientTape() as tape:
        reconstructed, mu, log_var = vae(x)
        loss = vae.compute_loss(x, reconstructed, mu, log_var)
    gradients = tape.gradient(loss, vae.trainable_variables)
    optimizer.apply_gradients(zip(gradients, vae.trainable_variables))
    return loss

# Обучение
epochs = 30
batch_size = 128

for epoch in range(epochs):
    for i in range(0, len(x_train), batch_size):
        x_batch = x_train[i:i + batch_size]
        loss = train_step(x_batch)
    print(f'Epoch {epoch + 1}, Loss: {loss.numpy()}')
```

```
To enable them in other operations,
Epoch 1, Loss: 176.88998413085938
Epoch 2, Loss: 173.79403686523438
Epoch 3, Loss: 169.65579223632812
Epoch 4, Loss: 166.28977966308594
Epoch 5, Loss: 164.37161254882812
Epoch 6, Loss: 163.59764099121094
Epoch 7, Loss: 162.5578155517578
Epoch 8, Loss: 161.3791046142578
Epoch 9, Loss: 160.0150604248047
Epoch 10, Loss: 158.47474670410156
Epoch 11, Loss: 158.0936737060547
Epoch 12, Loss: 157.31736755371094
Epoch 13, Loss: 156.79994201660156
```

Fig.6

Potential Challenges

Data Quality and Availability: Datasets often contain a large number of features, many of which may be redundant or irrelevant. This leads to overfitting and reduces model performance. Effective feature selection and dimensionality reduction techniques are critical to mitigate this risk.

Class Imbalance: In many software fault prediction datasets, non-faulty instances vastly outnumber faulty ones. This imbalance causes models to be biased towards the majority class, lowering fault detection rates for critical faulty modules.

Noisy and Redundant Data: Datasets may contain erroneous, duplicated, or irrelevant entries, which can mislead the learning process and degrade the predictive capability of neural networks.

Missing Values: Incomplete data entries can result in inaccurate model training, necessitating preprocessing techniques such as imputation or discarding corrupted samples.

Class Overlapping: In some cases, characteristics of faulty and non-faulty classes are very similar, making it challenging for models to distinguish between them. Advanced classification techniques or additional feature engineering may be required to address this issue.

Fig.7

Computational Resources: Neural network training requires substantial computational power and time, particularly when employing deep learning architectures or extensive hyperparameter tuning, potentially limiting practical applicability.

Interpretability and Transparency: Neural networks often act as "black boxes," complicating efforts to interpret their decision-making processes clearly. This lack of transparency can hinder trust and acceptance, especially in critical testing scenarios.

Overfitting and Generalization: Achieving optimal generalization without overfitting training data can be complex, necessitating rigorous validation techniques and careful model design.

Conclusion

In conclusion, neural networks present a transformative potential in software testing by fundamentally reshaping traditional methodologies through automation and enhanced fault detection capabilities. By utilizing multilayered neural network architectures, trained via rigorous backpropagation algorithms, the proposed methodology effectively minimizes the manual effort required in verifying software correctness and significantly improves fault detection accuracy. The empirical study demonstrated through the credit card approval software clearly illustrates the neural network's proficiency in identifying both intentional and unintentional faults efficiently. Moreover, integrating generative algorithms such as GANs and VAEs significantly enriches the training datasets, enhancing the neural network's robustness, resilience, and adaptability to evolving software environments. Ultimately, this approach not only streamlines testing processes, reducing costs and resource expenditures, but also establishes new benchmarks for reliability and performance, driving further innovation and progress in software engineering practices.

REFERENCES

1. Michael Nielsen. Neural Networks and Deep Learning. 2015.
2. Chris M. Bishop. Neural Networks and Their Applications. 1994.
3. D.H. Nguyen, B. Widrow. Neural Networks for Self-Learning Control Systems. 1990.
4. Koushal Kumar, G.S.M. Thakur. Advanced Applications of Neural Networks and Artificial Intelligence: A Review. 2012.
5. Manavendra Misra, Brad Warner. Understanding Neural Networks as Statistical Tools. 1996.
6. J.A. Anderson. Two Models for Memory Organization. Mathematical Biosciences, 1970.
7. G. Cybenko. Approximation by Superpositions of a Sigmoidal Function. Mathematics of Control, Signals, and Systems, 1989.
8. Z. Khabazi. Generative Algorithms. 2011.
9. Stanford University. Generative Learning Algorithms — CS229 Lecture Notes. 2008.
10. Inês Caetano. Computational Design in Architecture: Defining Parametric, Generative, and Algorithmic Design. Frontiers of Architectural Research, 2020.
11. Ali Bou Nassif, Mohammad Azzeh, Luiz F. Capretz, Danny Ho. Neural Network Models in Software Engineering Applications. 2021.

НЕЙРОНДЫҚ ЖЕЛІЛЕРДІ ЗЕРТТЕУ ЖӘНЕ БАҒДАРЛАМАЛЫҚ ТҮРДЕ ЖҮЗЕГЕ АСЫРУ

Самаликов Фараби, Оралбекова Жанар

Бұл мақалада нейрондық желілерді бағдарламалық тестілеуде қолданылуы, олардың теориялық негіздері, практикалық іске асырылуы және инновациялық тәсілдері егжей тегжейлі қарастырылады. Онда нейрондық желілер функцияларын автоматтандыру және ақауларды анықтау мүмкіндіктерін жақсарту арқылы бағдарламалық тестілеу процестерін айтарлықтай жақсарта алатыны көрсетілген. Несиелік карталарды мақұлдау бағдарламалық жасақтамасын кейс-стади арқылы эмпирикалық тексеру нейрондық желі әдістемелерінің тиімділігі мен тиімділігін көрсетеді. Мақалада сонымен қатар деректер сапасы, есептеу ресурстарына қойылатын талаптар, интерпретациялану және шамадан тыс сәйкестік сияқты ықтимал мәселелер қарастырылады, бұл бағдарламалық қамтамасыз етуді әзірлеудегі нейрондық желілердің трансформациялық әлеуетіне жан-жақты шолу жасайды.

Кілт сөздері: нейрондық желілер, бағдарламалық жасақтаманы тестілеу, автоматтандырылған oracle, ақауларды анықтау, генеративті алгоритмдер, машиналық оқыту, эмпирикалық тексеру.

ИССЛЕДОВАНИЕ И РЕАЛИЗАЦИЯ НЕЙРОННЫХ СЕТЕЙ

Самаликов Фараби, Оралбекова Жанар

Статья рассматривает применение нейронных сетей в процессе тестирования программного обеспечения, раскрывая их теоретические основы, практическую реализацию и инновационные подходы. Подчёркивается, что нейронные сети могут значительно улучшить процессы тестирования программного обеспечения за счёт автоматизации функций и повышения точности обнаружения ошибок. Эмпирическая проверка на примере программного обеспечения для одобрения кредитных карт демонстрирует эффективность и результативность предложенных методик. Также обсуждаются потенциальные трудности, такие как качество данных, потребности в вычислительных ресурсах, интерпретируемость и проблема переобучения, обеспечивая комплексный обзор трансформирующего потенциала нейронных сетей в области программной инженерии.

Ключевые слова: нейронные сети, тестирование программного обеспечения, автоматизированный оракул, обнаружение ошибок, генеративные алгоритмы, машинное обучение, эмпирическая проверка