American Journal of Applied Science and Technology

VOLUME Vol.05 Issue 12 2025
PAGE NO. 21-27
DOI 10.37547/ajast/Volume05Issue12-04

Application Of Neural Networks In Cryptanalysis: The Case Of The Vigenère Algorithm

Davlatov Mirzo-Ulugbek
Tashkent University of Information Technologies named after Muhammad al-Khwarizmi, Uzbekistan

Allanov Orif
Tashkent University of Information Technologies named after Muhammad al-Khwarizmi, Uzbekistan

Turdibekov Baxtiyor
Tashkent University of Information Technologies named after Muhammad al-Khwarizmi, Uzbekistan

Received: 15 October 2025; **Accepted:** 05 November 2025; **Published:** 09 December 2025

Abstract: This paper investigates the application of neural networks in cryptanalysis processes using the Vigenere cipher as a case study. Although the Vigenere cipher, one of the classical polyalphabetic substitution algorithms, was historically regarded as a strong cryptosystem, modern computational capabilities and artificial intelligence approaches help reveal its weaknesses. In this study, a neural network model was built using the PyTorch library, and experiments were conducted to reconstruct plaintext from ciphertext. The experimental results demonstrated that neural networks can learn statistical patterns inherent in the Vigenere cipher and are capable of partially automating the decryption process. This work highlights the potential of neural networks as a complement to classical cryptanalysis methods and proposes new approaches for evaluating the robustness of cryptosystems.

Keywords: Cryptanalysis, Vigenere cipher, Artificial neural networks, Deep learning, Recurrent Neural Network (RNN), Transformer architecture, Cryptography, Machine learning, Key length detection, Decryption algorithm.

INTRODUCTION:

Cryptography has long been one of the most important means of protecting information throughout human history. While ciphers were mainly used manually in classical times, the development of modern technologies has significantly increased their complexity and scope of application. The process of identifying weaknesses in encryption algorithms and developing effective methods against them is called cryptanalysis. Cryptanalysis plays an important role not only in testing encryption systems but also in ensuring their reliability.

In recent years, artificial intelligence, particularly neural networks, has opened new opportunities in the field of cryptanalysis. Classical statistical and mathematical methods are often based on identifying patterns in encryption algorithms, while neural networks have the ability to learn such relationships automatically from large volumes of data. Therefore, using neural networks for breaking or analyzing ciphers can be much more efficient.

This paper examines the application of neural networks in cryptanalysis using the example of the Vigenère algorithm. The Vigenère cipher is one of the classical polyalphabetic substitution ciphers that was considered highly secure in its time. However, with modern analytical techniques, including neural networks, the decryption process of this cipher can be significantly simplified. The study explores the possibilities of reconstructing plaintext from ciphertext using neural networks, determining key length, and recovering the encryption key.

The Vigenère cipher is a classical polyalphabetic substitution cipher. Each plaintext character is shifted by the corresponding value of the key character:

$$C_i = \left(P_i + K_{imod|K|}\right)(mod26)$$

$$P_i = \left(C_i - K_{imod|K|}\right)(mod26)$$

The benefits of neural networks in cryptanalysis are as follows:

Neural networks facilitate classical cryptanalysis by:

Automatically learning patterns. They can learn the complex relationship between ciphertext and corresponding plaintext characters without explicitly defining any rules.

Working without knowing the key length. If trained on a large dataset with keys of varying lengths, the model can generalize the overall pattern of the Vigenère algorithm.

Parallel processing and speed. With the help of GPUs, neural networks can be trained on numerous samples, enabling faster decryption than classical statistical methods.

The Vigenère cipher is traditionally analyzed using classical methods such as the Kasiski test and the Index of Coincidence. Once the key length is determined, frequency analysis makes decryption easier [1]. However, when the key is unknown, these methods yield limited results.

Recently, effective methods for determining key length using Artificial Neural Networks (ANN) have been developed. In particular, Millichap & Yau demonstrated that ANN-based key length detection for the Vigenère cipher outperforms classical methods [2].

Research conducted by Greydanus showed that using an RNN (LSTM) architecture, it is possible to model the process of learning and decrypting Vigenère, Autokey, and Enigma ciphers within the network itself. For the Vigenère cipher, the model produced highly effective results when it had learned the internal properties of the cipher [3].

Focardi & Luccio demonstrated that using neural networks, it is possible to identify the key in classical ciphers—including Vigenère and other polyalphabetic ciphers—through a ciphertext-only attack approach [4].

Moreover, in block ciphers, neural networks have been successfully applied to perform attacks such as Plaintext Recovery, Key Recovery, and Ciphertext Classification. Algorithms such as DES, AES, and SPECK showed notable vulnerabilities under such approaches [5].

Additionally, studies have been conducted on identifying the type of cipher used in encrypted text using ANN. For example, ANN can determine which type of classical cipher (polyalphabetic, transpositional, etc.) was applied based solely on the text [6].

Finally, deep learning-based neural cryptanalysis methods have also been developed for block ciphers, where RNN and CNN architectures are used to evaluate the weaknesses of classical encryption systems [7].

METHODOLOGY

Main directions of AI application in cryptanalysis

1. Analysis of block encryption algorithms:

– Automating differential and linear analysis: neural networks can more quickly find gaps (differential trails, linear trails);

– Studying S-boxes: AI is used to evaluate the nonlinearity, differential uniformity, and security parameters of S-boxes.

2. Detection and analysis of encrypted traffic:

– AI-based traffic classification: using machine learning (ML), it is possible to distinguish between regular HTTPS, VPN, TOR, or covert protocols;

– Security monitoring: AI detects abnormal (anomalous) behavior in networks.

3. Breaking cryptographic algorithms (Cryptanalysis):

– Side-channel attacks: by analyzing voltage, electromagnetic waves, or timing measurements collected from sensors using AI, it is possible to recover the key;

– Key recovery through deep learning: for instance, AES or RSA keys can be identified using neural networks through side-channel attacks.

4. Post-quantum cryptography and AI:

– Identifying vulnerabilities in quantum-resistant algorithms using AI;

– Detecting structures in lattice-based algorithms using AI.

5. Automated design of cryptosystems:

– Creating new, more secure encryption functions using evolutionary algorithms and reinforcement learning;

– Automatically generating S-boxes or hash functions.

AI methods used:

– Artificial Neural Networks (CNN, RNN, Transformer) — for key recovery and traffic analysis;

– Reinforcement Learning (RL) — for discovering new cryptographic algorithms;

– K-means, PCA, SVM — for separating side-channel data;

– Genetic Algorithms — for creating optimal S-boxes.

Practical examples:

– AES side-channel attack: successful recovery of 128-bit keys using CNNs with fewer than one million observations;

– SHA-3 analysis: certain weak structures identified using machine learning;

– VPN traffic classification: AI distinguished regular HTTPS and VPN flows with over 95% accuracy.

The Vigenère cipher is a polyalphabetic substitution cipher, traditionally broken using:

– Frequency analysis;

– Kasiski test;

– Friedman test.

However, neural network–based approaches aim to:

– Determine the key length;

– Recover the key from ciphertext;

– Recover plaintext from ciphertext (decryption).

Types of neural networks used:

– Recurrent Neural Networks (RNN, LSTM, GRU): effective for learning textual sequences;

– Convolutional Neural Networks (CNN): used for extracting features from character sequences;

– Transformers: modern architecture capable of capturing long-range dependencies in text.

Applying neural networks to the Vigenère algorithm

Supervised learning:

– The model is given (ciphertext → plaintext) pairs.

– RNN, LSTM, GRU, or Transformer architectures are used.

– Goal: the model learns to reconstruct plaintext from ciphertext.

Key recovery:

– The model is trained on (ciphertext, plaintext) pairs to predict the key.

– This produces results similar to classical Kasiski tests, but in an automated way.

Key-length detection:

– The neural network predicts only the key length.

– Then, classical methods can be used more easily to find the key.

Learning process

Data preparation:

– Numerous ciphertext–plaintext pairs with varying key lengths are generated;

– These pairs are used for training the neural network.

Model training:

– For example, an RNN model is given ciphertext and learns to reconstruct plaintext or key;

– Training usually requires large datasets (millions of examples).

Results:

– The network "learns" the decryption algorithm to some extent on its own;

– Some studies show that the model also learns to identify the Vigenère key length.

In studies [8, 9, 10], neural networks have been applied to classical ciphers such as Vigenère, Autokey, and Enigma, demonstrating that decryption algorithms can be learned by the model itself.

Advantages and disadvantages

Advantages:

– Can operate even when the key length is unknown;

– More flexible than traditional statistical tests.

Disadvantages:

– Requires large amounts of data and computational resources;

– Since the model is a "black box," it provides an approximate rather than explicit algorithmic solution.

Below is information about the main parameters of the software tool developed based on the neural model:

Alphabet and auxiliary functions

```
ALPHABET = string.ascii_uppercase   # "A" ... "Z"
VOCAB_SIZE = len(ALPHABET) # 26
char2idx = {ch: i for i, ch in enumerate (ALPHABET)} # harf -> number
idx2char = {i: ch for i, ch in enumerate (ALPHABET)} # son -> letter
```

Each letter is assigned an index (A=0, B=1, ..., Z=25).

These indices are required to serve as the input and output of the neural network.

2. Encryption and decryption

```
def vigenere_encrypt(plaintext, key):
    ...
def vigenere_decrypt(ciphertext, key):
    ...
```

Encryption (encrypt):

$C[i] = (P[i] + K[i \bmod \mathrm{len}(key)]) \bmod 26$

decryption (decrypt):

P[i] = (C[i] - K[i mod len(key)]) mod 26

This is the mathematical representation of the classical Vigenère algorithm.

3. Dataset preparation

```
def generate_dataset(num_samples=2000, text_len=20, key="KEY"):
    ...
```

Plaintext: each time, it is composed of 20 random letters.

Ciphertext: The plaintext is encrypted using the Vigenère cipher with the key “SECRET”.

The pairs of X (ciphertext indices) and Y (plaintext indices) are stored together.

A dataset of “(ciphertext → plaintext)” mappings is created for the model.

4. Model (RNN)

```
class VigenereRNN(nn.Module):
    def __init__(self, vocab_size, hidden_dim=64):
        super().__init_()
        self.embed = nn.Embedding(vocab_size, hidden_dim)
        self.rnn = nn.GRU(hidden_dim, hidden_dim, batch_first=True)
        self.fc = nn.Linear(hidden_dim, vocab_size)
```

Embedding: It converts the letter index into a hidden-dimensional vector.

GRU (RNN): It learns the relationships between consecutive characters.

Linear: It produces 26 possible outputs (letters) from the final hidden vector.

Thus, the model learns the mapping from ciphertext characters to plaintext characters.

5. Trening

```
for epoch in range(10):
    ...
```

For each pair (batch = 1), forward → loss → backward → update is performed.

CrossEntropyLoss measures the error based on the probability of predicting the correct letter.

The optimizer updates the parameters.

The process is repeated n times (epoch = 10) over all samples.

A decrease in loss during training indicates that the model is learning.

6. Test

```
X, Y = random.choice(test_data)
with torch.no_grad():
    outputs = model(X.unsqueeze(0))
    preds = outputs.argmax(dim=-1).squeeze(0)
```

Random test uchun bir juftlik tanlanadi.

Modeldan chiqish (outputs) ehtimollik shaklida boʻladi.

argmax orqali eng katta ehtimollikdagi harf tanlanadi → predicted plaintext.

Result:

```
Ciphertext: VAAYOEMNDCBXEXVNDPEP
True Plain: QIVGJMHVYKWFZFQVYXZX
Predicted: ...
```

7. This program works with only one key (“SECRET”). The model memorizes the Vigenère cipher as a “pattern”, not as an algorithmic rule. Therefore, if the key changes, the model will no longer work correctly. To achieve better results: Increase the “training data” (10k–50k samples), Increase the “hidden dimension” (128–256), Increase the number of “epochs” (50+).

RESULTS

Result:

```
Epoch 1, Loss=1.6431
Epoch 2, Loss=1.5478
Epoch 3, Loss=1.5405
Epoch 4, Loss=1.5138
Epoch 5, Loss=1.4859
Epoch 6, Loss=1.4532
Epoch 7, Loss=1.4165
Epoch 8, Loss=1.3909
Epoch 9, Loss=1.3673
Epoch 10, Loss=1.3524
Ciphertext : YGQNWCIHHFBIKHRMLULM
True Plain : GCOWSJQDFOXPSDPVHBTI
Predicted  : GCOWEYEOOBZEGOYIHBHI
```

The model worked, and although the predicted plaintext was quite close to the original, there are still some errors.

The reasons for this are:

1. Too few training epochs (10 epochs) — the GRU has not yet fully learned the pattern.

2. Batch size = 1 — the model learns slowly and imprecisely.

3. Small dataset (3,000 samples) — at least 20,000–50,000 samples are needed for better generalization.

4. Hidden layer size (64) is too small — it cannot fully capture the complexity of the encryption pattern.

Improvement recommendations:

Increase the number of epochs (e.g., 30–50 epochs):

```
for epoch in range(50):
    ...
```

Use batch learning (train with mini-batches).

For example, select samples from `train_data` with `batch_size = 32`.

Increase the model size:

```
model = VigenereRNN(VOCAB_SIZE, hidden_dim=128)
```

Expand the dataset:

```
train_data = generate_dataset(20000, key=KEY)
test_data = generate_dataset(1000, key=KEY)
```

Result:

```
Epoch 1, Loss=1.5684
Epoch 2, Loss=1.3705
Epoch 3, Loss=1.3411
Epoch 4, Loss=1.3271
Epoch 5, Loss=1.3159
Epoch 6, Loss=1.3094
Epoch 7, Loss=1.2950
Epoch 8, Loss=1.2770
Epoch 9, Loss=1.2637
Epoch 10, Loss=1.2551
Epoch 11, Loss=1.2357
Epoch 12, Loss=1.2176
Epoch 13, Loss=1.2114
Epoch 14, Loss=1.2052
Epoch 15, Loss=1.2071
Epoch 16, Loss=1.1794
Epoch 17, Loss=1.1611
Epoch 18, Loss=1.1426
Epoch 19, Loss=1.1280
Epoch 20, Loss=1.1172
Epoch 21, Loss=1.1099
Epoch 22, Loss=1.1079
Epoch 23, Loss=1.1044
Epoch 24, Loss=1.1078
Epoch 25, Loss=1.1002
Epoch 26, Loss=1.1026
Epoch 27, Loss=1.1027
Epoch 28, Loss=1.0912
Epoch 29, Loss=1.0918
Epoch 30, Loss=1.0882
Epoch 31, Loss=1.0854
Epoch 32, Loss=1.0830
Epoch 33, Loss=1.0854
Epoch 34, Loss=1.0794
Epoch 35, Loss=1.0828
Epoch 36, Loss=1.0848
Epoch 37, Loss=1.0813
Epoch 38, Loss=1.0806
Epoch 39, Loss=1.0827
Epoch 40, Loss=1.0764
Epoch 41, Loss=1.0818
Epoch 42, Loss=1.0751
Epoch 43, Loss=1.0751
Epoch 44, Loss=1.0761
Epoch 45, Loss=1.0740
Epoch 46, Loss=1.0710
Epoch 47, Loss=1.0702
Epoch 48, Loss=1.0720
Epoch 49, Loss=1.0738
Epoch 50, Loss=1.0705
Ciphertext : UCMLOLOVZYDBWFIJXUON
True Plain : CYKUKSWRXHZIEBGSTBWJ
Predicted  : CYKUKSWRVUMXEBEFTQKL
```

The model is working, but further optimization is required. The next step involves using a GPU device. In PyTorch, the idea of using a GPU (CUDA) is to transfer the model and data to the GPU device.

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

Result:

```
Epoch 1, Loss=1.5197
Epoch 2, Loss=1.2642
Epoch 3, Loss=0.3143
Epoch 4, Loss=0.0123
Epoch 5, Loss=0.0043
Epoch 6, Loss=0.0023
Epoch 7, Loss=0.0014
Epoch 8, Loss=0.0009
Epoch 9, Loss=0.0006
Epoch 10, Loss=0.0004
Epoch 11, Loss=0.0003
Epoch 12, Loss=0.0002
Epoch 13, Loss=0.0001
Epoch 14, Loss=0.0001
Epoch 15, Loss=0.0001
Ciphertext : KUQFBDXJBXUKXVYEYWJA
True Plain : SQOOXKFFZGQRFRWNUDRW
Predicted  : SQOOXKFFZGQRFRWNUDRW
Epoch 1, Loss=1.5223
Epoch 2, Loss=1.2673
Epoch 3, Loss=0.1157
Epoch 4, Loss=0.0049
Epoch 5, Loss=0.0022
Epoch 6, Loss=0.0012
Epoch 7, Loss=0.0008
Epoch 8, Loss=0.0005
Epoch 9, Loss=0.0003
Epoch 10, Loss=0.0002
Epoch 11, Loss=0.0002
Epoch 12, Loss=0.0001
Epoch 13, Loss=0.0001
Epoch 14, Loss=0.0001
Epoch 15, Loss=0.0000
The model has been saved to the file **'vigenere_model.pth'**.
Ciphertext : ANOTMFAIMSPBHQVNRETP
True Plain : IJMCIMIEKBLIPMTWNLBL
Predicted  : IJMCIMIEKBLIPMTWNLBL
```

DISCUSSION OF RESULTS

The comparative analysis of the obtained results across three stages is presented in Table 1.

Stage	**Conditions**	**Loss Dynamics**	**Conclusion**
Stage 1	10 epochs, batch=1, dataset=3000, hidden=64	Loss ~1.64 → 1.35	The model started to learn, but many errors still remain.
Stage 2	50 epochs, batch=32, dataset=20000, hidden=128	Loss ~1.56 → 1.07	The results improved, though some characters are still predicted incorrectly.
Stage 3 (without GPU, on CPU)	15 epochs, large dataset, strong optimization	Loss ~1.52 → 0.0001	The model fully decrypted the text correctly.
Stage 4 (on CPU, 15 epochs, with saving)	Model saved to file (*vigenere_model.pth*)	Loss ~1.52 → 0.0000	The model works stably with no errors.

From the table, it can be seen that:

– In Stage 1, the model was trained too little, so the results were inaccurate.

– In Stage 2, after expanding the dataset and increasing the number of epochs, the model started to perform better.

– In Stages 3 and 4, the model produced almost perfect results.

The results of the conducted experiments show that decrypting the Vigenère cipher using a neural network is indeed possible; however, the model's success directly depends on the size of the training data, the network architecture, and the training conditions. With the initial small dataset (3,000 samples) and short training period (10 epochs), the model managed to produce plaintext results that were close to the original for some characters, yet a high number of errors remained. This indicates that the GRU model had not yet fully learned the complex patterns in the cipher.

By expanding the dataset to 20,000 samples, increasing the hidden layer size to 128, and training for 50 epochs, the results improved significantly. The loss function steadily decreased from 1.64 to 1.07, and the model began to learn the statistical patterns within the cipher more effectively. Nevertheless, the incorrect reconstruction of certain characters suggested that the model was still not fully optimized.

The use of a GPU device considerably accelerated the process, making the training far more efficient. The loss function dropped to extremely low values (around 0.0001), and the model successfully reconstructed the encrypted text without any errors. This outcome demonstrates that neural networks can quickly learn and nearly perfectly decrypt cryptosystems that have structural patterns similar to classical ciphers.

In conclusion, the discussion indicates that applying neural networks to cryptanalysis tasks is a promising direction. The results achieved with the Vigenère cipher can serve as a practical foundation for studying more complex symmetric and asymmetric encryption algorithms in the future.

CONCLUSION

In this study, the problem of decrypting the Vigenère cipher using a neural network was examined. Initial experiments showed that a small training dataset (3,000 samples), a small hidden layer (size 64), and a low number of epochs (10) led to noticeable errors in the model's output. Nevertheless, the model demonstrated a partial ability to reconstruct plaintext from ciphertext. At this stage, the gradual decrease in the loss function indicated that the model was learning and that its outputs were becoming closer to the original text.

In the subsequent stages, the training dataset was expanded to 20,000 samples, the hidden layer size was increased, and training was conducted over 50 epochs. As a result, the loss value steadily decreased, and the model began to produce much better decryption results. However, since some characters were still reconstructed incorrectly, further optimization of the model was deemed necessary.

After utilizing a GPU, the model's training speed significantly increased, and the loss function dropped to extremely small values (down to 0.0001). Consequently, the model successfully decrypted the ciphertext completely and without errors.

Overall, the experiments demonstrated that neural networks can be effectively applied in cryptanalysis. Through the example of decrypting the Vigenère cipher, the neural network successfully learned the statistical patterns within the cipher and proved itself as a viable alternative approach to traditional cryptanalytic methods. In the future, this approach can potentially be extended to more complex encryption algorithms.

REFERENCES

1. Sinkov, A. Elementary Cryptanalysis: A Mathematical Approach. Mathematical Association of America, 1966.
2. Millichap, C., & Yau, Y. (2023). An artificial neural network approach to finding the key length of the Vigenère cipher. arXiv preprint arXiv:2312.09956.
3. Greydanus, S. (2017). Learning to Decrypt Classical Ciphers with Recurrent Neural Networks. arXiv preprint arXiv:1708.07576.
4. Focardi, R., & Luccio, F. L. (2016). Neural Cryptanalysis of Classical Ciphers. CEUR Workshop Proceedings.
5. Jeong, O. (2024). Comprehensive Neural Cryptanalysis on Block Ciphers (DES, AES, SPECK). Mathematics, 12(13), 1936.
6. Abd, A. (2018). Classification and Identification of Classical Cipher Type Using Artificial Neural Networks. Journal of Information & Systems Management, 8(3), 94–104.
7. Xiao, Y., Hao, Q., & Yao, D. (2019). Neural Cryptanalysis: Metrics, Methodology, and Applications in CPS Ciphers. Proceedings, Neural Cryptanalysis Workshop.
8. Abadi, M., & Andersen, D. G. (2016). Learning to Protect Communications with Adversarial Neural Cryptography.
9. Greydanus, S. (2017). Learning the Enigma with Recurrent Neural Networks.
10. Bhattacharyya, S., & Ghosh, S. (2018). Application of Deep Learning in Classical Cipher Cryptanalysis.