Turkish Journal of Computer and Mathematics Education Vol.14 No.03 (2023),349-355
Research Article

Enhancing Credit Card Fraud Detection using Neural Networks and Adversarial Training

Dharavath Sai Kiran[1], Mittapelly Luckydhar[2], Shalini Priya Bairy[3], Roopa Chandrika R[4]

[1,2,3]B.Tech Student, Department of CSE (Data Science), Malla Reddy College of Engineering and Technology, Hyderabad, India.

[4]Professor, Department of CSE (Data Science), Malla Reddy College of Engineering and Technology, Hyderabad, India.

***Abstract*— Credit card fraud poses a significant challenge in financial transactions, necessitating the development of robust detection systems. This paper introduces an approach utilizing neural networks and adversarial training for credit card fraud detection. The proposed model leverages deep learning techniques to learn intricate patterns and detect fraudulent transactions effectively. By preprocessing the dataset, constructing a neural network model with appropriate layers, and training it using adversarial examples generated through perturbation, the model enhances its resilience to adversarial attacks. Experimental results demonstrate improved accuracy and robustness, contributing to secure transactions and preventing financial losses in the banking and financial sectors.**

***Keywords*— Credit card fraud detection, Neural networks, Adversarial training, Machine learning, Financial transactions.**

I. INTRODUCTION

Credit card fraud is a persistent and evolving challenge in the realm of financial institutions and individuals. As technology advances, fraudsters continue to devise new techniques to deceive traditional fraud detection systems. Therefore, the development of advancement and resilient fraud detection mechanisms becomes crucial in combating this menace.

In recent years, machine learning and deep learning algorithms have shown promising results in various domains[6], including fraud detection. Neural networks, in particular, have demonstrated their ability to learn complex patterns and make accurate predictions. By leveraging the power of neural networks, we aim to enhance credit card fraud detection[4] systems and improve their effectiveness in identifying fraudulent transactions.

This paper presents an approach to credit card fraud detection that combines neural networks with adversarial training. The utilization of neural networks[3,8] enables the model to learn intricate relationships and patterns within the data, making it adept at distinguishing between legitimate and fraudulent transactions. Additionally, adversarial training is employed to enhance the model's resilience against adversarial attacks, which are attempts by fraudsters to manipulate or deceive the fraud detection system[5].

To address the vulnerability of traditional models to adversarial attacks, adversarial training is introduced, Adversarial examples, created by perturbing the input data based on calculated gradients, are added to the training set. This augmented dataset, which includes both original model to adversarial examples, it learns to recognize and defend against adversarial perturbations, resulting in improved fraud detection performance.

Adversarial training[10] is a technique used in machine learning to improve the robustness and security of models against adversarial attacks. Adversarial attacks are deliberate attempts to deceive or manipulate a machine learning model by introducing carefully crafted input data that can cause misclassification or incorrect predictions.

The idea behind adversarial training is to expose the model to adversarial examples during the training process, which are modified versions of the original input data specifically designed to deceive the model. By incorporating these adversarial examples into the training set, the model learns to recognize and defend against them, thus increasing its resistance to future adversarial attacks.

The performance of the proposed approach is evaluated using the testing set, and metrics such as accuracy, classification report, and confusion matrix are computed. Furthermore, examples of fraud detection are provided, demonstrating the model's capability to classify random data points, custom data points, and user input.

The outcomes of this research contribute to the advancement of credit card fraud detection systems, offering improved accuracy and resilience against adversarial attacks. By leveraging the capabilities of neural networks and incorporating adversarial training, the proposed methodology holds promise in strengthening the security of financial transactions and mitigating the detrimental impact of credit card fraud.

II. LITERATURE REVIEW

Credit card fraud detection [7] is a critical area due to the increasing number of fraudulent activities in online transactions. Various machine learning techniques[9] have been employed to tackle this problem. In this literature review, we discuss two studies conducted by Mr. Thirunavukkarasu.M, Achutha Nimisha, Adusumilli Jyothsna proposed "Credit card fraud detection using Machine Learning [1]" and Emmanual Ileberi, Yanxia Sun and Zenghui Wang proposed "A Machine

learning based credit card fraud detection using the GA algorithm for feature selection [2]", respectively.

Thirunavukkarasu et al [1] focus on developing a model to detect fraud activities in credit card transactions. They utilize a random forest algorithm and decision trees to analyze the dataset and achieve a high accuracy of 98.6% for fraud detection. However, their approach relies solely on the random forest algorithm, which may not capture complex patterns and relationships present in the data.

Emmanuel Illebri et al [2] address credit card fraud detection using machine learning methods and the genetic algorithm for feature selection. While their work demonstrates the application of feature selection techniques'

Our proposed idea builds upon the strengths by incorporating neural networks and adversarial training into the credit card fraud detection system.

In conclusion, while Thirunavukkarasu et al [1] and Emmanuel et al [2] present valuable contributions to credit card fraud detection, our proposed idea extends their work by incorporating neural networks and adversarial training. By leveraging neural networks, adversarial training and addressing the limitations of existing methods, our approach aims to achieve improved fraud detection accuracy, robustness against adversarial attacks, and transparency in decision making.

III. METHODOLOGY

1. Data preprocessing:
   - Load the credit card dataset using Pandas and perform exploratory data analysis to gain insights into the data.
   - Standardize the "Amount" feature by subtracting the mean and dividing by the standard deviation to ensure consistent scaling.
   - Split the dataset into a training set and a test set.

2. Neural Network Model Construction:
   - Create a neural network model using the Keras API from TensorFlow.
   - Design the model architecture with multiple dense layers, incorporating appropriate activation functions such as ReLU and sigmoid.
   - Compile the model with Adam optimizer, binary cross-entropy loss function, and accuracy as the metric.

3. Training the initial model:
   - Train the model using the training set.
   - Specify the number of training epochs and batch size to iterate over the dataset and update the model's parameters.

4. Adversarial Training:
   - Define a function called "adversarial train" that performs adversarial training for a given model, input data (x), labels (y), and perturbations magnitude (eps).
   - Within the function, use TensorFlow's "Gradient Tape" to record the operations for gradient calculation.
   - Calculate the loss between the predicted output and the true labels, and compute the gradients of the loss with respect to the input data.
   - Generate adversarial examples by adding a perturbation scaled by "eps" to the input data in the direction of the gradients.
   - Clip the adversarial examples to ensure they remain within a valid range.
   - Combine the original and adversarial examples with their respective labels to create an augmented dataset.
   - Return the augmented dataset, the adversarial examples, and the perturbation.

5. Adversarial Training loop:
   - Initialize a new adversarial model with the same architecture as the initial model.
   - Iterate through a predefined number of training iterations.
   - For each iteration, perform adversarial training using the "adversarial train" function to update the model's parameters.
   - Monitor and evaluate the model's performance during the training process.
6. Evaluation of Adversarial Model:
   - Evaluate the adversarial model using the test set to measure its performance.
   - Calculate the test accuracy metrics to assess the model's effectiveness,
   - Generate a classification report to provide detailed performance metrics such as precision, recall, and F1-score.
   - Compute a confusion matrix to visualise the model's predictions and identify false positive and false negatives.

7. Fraud detection examples:
Demonstrate the fraud detection capabilities of the trained model using various scenarios.
   - Random Data Point Fraud Detection: Select a random data point from the data set, predict its label using the trained model.
   - Custom Data Point Fraud Detection: Prompt the user to input the index of a data point they want to check, extract the corresponding data point from the dataset, make a prediction using the model, and compare it with actual label.

- Fraud Detection for User Input: Allow the user to feature values for a custom data point, use the model to classify the input as fraudulent or legitimate, and provide the prediction to the user.

IV. IMPLEMENTATION

1. LOAD THE CREDIT CARD DATASET USING PANDAS AND PERFORM DATA EXPLORATION.
2. PREPROCESS THE DATASET
3. SPLIT THE DATA SET INTO A TRAINING SET AND TEST SET.
4. BUILD A NEURAL NETWORK MODEL USING THE KERAS API AND ACTIVATION FUNCTIONS.
5. TRAIN THE INITIAL MODEL ON THE TRAINING SET USING THE ADAM OPTIMIZER, BINARY CROSS-ENTROPY LOSS, AND ACCURACY AS THE METRIC.
6. DEFINE A FUNCTION ADVERSARIAL TRAINING USING GRADIENTTAPE.
7. CREATE A NEW ADVERSARIAL MODEL WITH THE SAME ARCHITECTURE AS THE INITIAL MODEL,
8. ITERATE THROUGH A SPECIFIED NUMBER OF TRAINING ITERATIONS.
9. EVALUATE THE ADVERSARIAL MODEL ON THE TEST SET TO CALCULATE TEST LOSS AND ACCURACY.
10. GENERATE A CLASSIFICATION REPORT AND CONFUSION MATRIX TO ASSESS THE MODEL'S PERFORMANCE.
11. DEMONSTRATE FRAUD DETECTION EXAMPLES.

```
list(df.columns)
```

```
['Time',
 'V1',
 'V2',
 'V3',
 'V4',
 'V5',
 'V6',
 'V7',
 'V8',
 'V9',
 'V10',
 'V11',
 'V12',
 'V13',
 'V14',
 'V15',
 'V16',
 'V17',
 'V18',
 'V19',
 'V20',
 'V21',
 'V22',
 'V23',
 'V24',
 'V25',
 'V26',
 'V27',
 'V28',
 'Amount',
 'Class']
```

Fig 1: List of columns

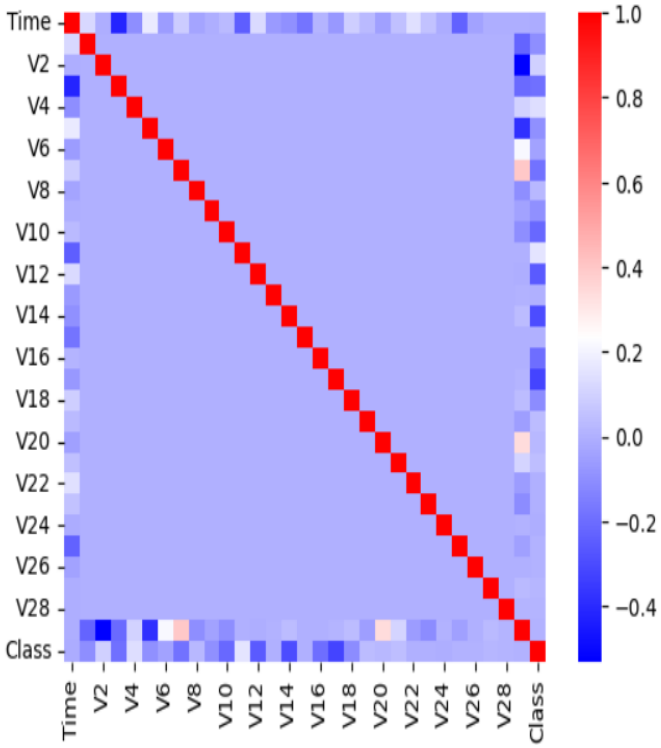

Fig 2: Correlation table

V. RESULTS

During the initial training phase of the model, the following results were observed.

```
model.fit(X_train, y_train, epochs=5, batch_size=15)
```

```
Epoch 1/5
13291/13291 [==============================] - 52s 4ms/step - loss: 1.8183 - accuracy: 0.9967
Epoch 2/5
13291/13291 [==============================] - 50s 4ms/step - loss: 0.0167 - accuracy: 0.9981
Epoch 3/5
13291/13291 [==============================] - 51s 4ms/step - loss: 0.0266 - accuracy: 0.9979
Epoch 4/5
13291/13291 [==============================] - 51s 4ms/step - loss: 0.0164 - accuracy: 0.9980
Epoch 5/5
13291/13291 [==============================] - 50s 4ms/step - loss: 0.0212 - accuracy: 0.9980

<keras.callbacks.History at 0x2064dcbdfa0>
```

```
test_loss, test_acc = model.evaluate(X_test, y_test)
print("Test accuracy:", test_acc)
```

```
2671/2671 [==============================] - 7s 3ms/step - loss: 0.0100 - accuracy: 0.9987
Test accuracy: 0.9987360239028931
```

Fig 3: Epoch and Test accuracy

An epoch refers to a complete pass or iteration through the entire dataset during the training process of a neural network. During each epoch, the model goes through the entire dataset, performs forward and backward propagation, updates the model's parameters (weights and biases), and gradually learns from the data to improve its performance. The number of epochs is a hyperparameter that determines how many times the model will iterate over the training dataset.
After training, the model was evaluated on the test set, resulting in a test accuracy of 0.9987.

For the adversarial training process, the model was trained using adversarial examples generated.

```
for i in range(10):
    X_combined, y_combined, x_adv, perturbation = adversarial_train(adv_model, X_train_tf, y_train_tf, eps=0.1)
    adv_model.fit(X_combined, y_combined, epochs=1, batch_size=15)
    print("Adversarial example", i+1, ":\n", x_adv.numpy())
```

```
26582/26582 [==============================] - 95s 3ms/step - loss: 0.1932 - accuracy: 0.9983
Adversarial example 1 :
 [[0.        0.        0.        ... 0.03355838 0.07894695 0.34496383]
 [0.        1.        0.36615071 ... 0.        0.        0.        ]
 [0.9       0.        0.        ... 0.04464721 0.        1.        ]
 ...
 [1.        0.        0.        ... 0.        0.2002437 0.        ]
 [1.        0.        0.        ... 0.        0.29635614 1.       ]
 [1.        0.        1.        ... 0.        0.        0.        ]]
26582/26582 [==============================] - 66s 2ms/step - loss: 0.0126 - accuracy: 0.9990
Adversarial example 2 :
 [[0.1       0.        0.        ... 0.03355838 0.        0.14496383]
 [0.1       1.        0.16615071 ... 0.        0.        0.        ]
 [1.        0.        0.        ... 0.        0.        1.        ]
 ...
 [1.        0.        0.05702875 ... 0.0723008  0.10024377 0.        ]
 [1.        0.        0.        ... 0.        0.19635614 1.       ]
 [1.        0.        1.        ... 0.        0.        0.        ]]
26582/26582 [==============================] - 67s 3ms/step - loss: 0.0114 - accuracy: 0.9990
Adversarial example 3 :
 [[0.1       0.        0.        ... 0.03355838 0.        0.14496383]
 [0.1       1.        0.16615071 ... 0.        0.        0.        ]
 [1.        0.        0.        ... 0.        0.        1.        ]
 ...
 [1.        0.        0.05702875 ... 0.0723008  0.10024377 0.        ]
 [1.        0.        0.        ... 0.        0.19635614 1.       ]
 [1.        0.        1.        ... 0.        0.        0.        ]]
26582/26582 [==============================] - 67s 3ms/step - loss: 0.0126 - accuracy: 0.9990
```

Fig 4: Generated Adversarial Examples

```
for i in range(10):
    X_combined, y_combined = adversarial_train(adv_model, X_train_tf, y_train_tf, eps=0.1)
    adv_model.fit(X_combined, y_combined, epochs=1, batch_size=15)
```

```
26582/26582 [==============================] - 99s 4ms/step - loss: 0.2271 - accuracy: 0.9983
26582/26582 [==============================] - 91s 3ms/step - loss: 0.0175 - accuracy: 0.9990
26582/26582 [==============================] - 93s 3ms/step - loss: 0.0093 - accuracy: 0.9990
26582/26582 [==============================] - 96s 4ms/step - loss: 0.0294 - accuracy: 0.9987
26582/26582 [==============================] - 98s 4ms/step - loss: 0.0169 - accuracy: 0.9990
26582/26582 [==============================] - 95s 4ms/step - loss: 0.0186 - accuracy: 0.9990
26582/26582 [==============================] - 93s 4ms/step - loss: 0.0131 - accuracy: 0.9990
26582/26582 [==============================] - 97s 4ms/step - loss: 0.0080 - accuracy: 0.9990
26582/26582 [==============================] - 97s 4ms/step - loss: 0.0139 - accuracy: 0.9990
26582/26582 [==============================] - 95s 4ms/step - loss: 0.0119 - accuracy: 0.9990
```

```
test_loss, test_acc = adv_model.evaluate(X_test, y_test)
print("Adversarial test accuracy:", test_acc)
```

```
2671/2671 [==============================] - 7s 3ms/step - loss: 0.0097 - accuracy: 0.9987
Adversarial test accuracy: 0.9987360239028931
```

Fig 5: Adversarial training and test accuracy

After the adversarial training, the adversarial model was evaluated on the test set, yielding an adversarial test accuracy of 0.9987

A Classification report was generated using the adversarial model's predictions and the actual labels.

```
from sklearn.metrics import classification_report
y_pred = adv_model.predict(X_test)
y_pred = np.round(y_pred)
print(classification_report(y_test, y_pred))
```

```
2671/2671 [==============================] - 6s 2ms/step
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85335
           1       0.00      0.00      0.00       108

    accuracy                           1.00     85443
   macro avg       0.50      0.50      0.50     85443
weighted avg       1.00      1.00      1.00     85443
```

Fig 6: Classification report

The test set consisted of 85,443 instances.
For class o (legitimate transactions), the precision, recall, and F1-score were all 1.00.
For class 1 (fraudulent transactions), the precision, recall, and F1-score were all 0.00
The overall accuracy of the model on the test set was 1.00
The macro-average F1-score was 0.50
The weighted-average F1-score was 1.00

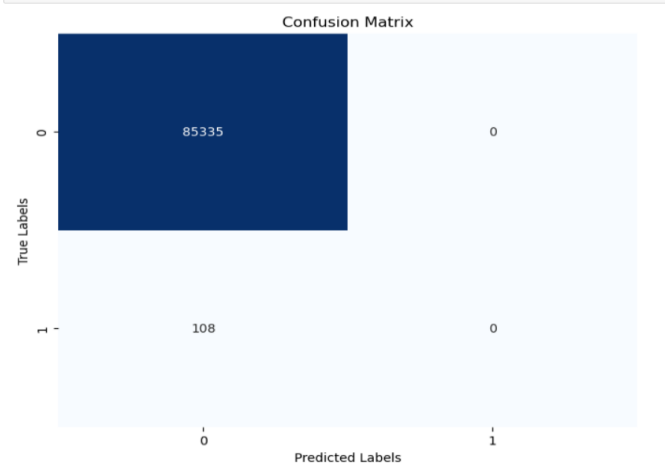

Fig 7: Confusion matrix

The confusion matrix is a table that provides a detailed performance analysis of a classification model by displaying the predicted and actual labels of a dataset.
The confusion matrix is typically organized into four quadrants:

True positives (TP) represent the cases where the model correctly predicted a positive outcome when the work was positive
True negatives (TN) are the cases where the model correctly predicted a negative outcome when the actual work was negative
False positives (FP) are the cases where the model was predicted a positive outcome when the actual work was negative
False negatives (FN) are the cases where the model predicted a negative outcome when the actual work was positive

Accuracy = (TP + TN) / (TP + TN + FP + FN)
Precision = TP / (TP + FP)
Recall = TP / (TP + FN)
F1-Score = 2 * (precision * recall) / (precision + recall)

Some fraud detection examples were used

```
Randomly picked data point:
Time       43898.000000
V1            -0.722523
V2             0.608715
V3             2.083034
V4             1.135068
V5             0.437541
V6            -0.558226
V7             0.840591
V8            -0.443556
V9            -0.302496
V10            0.469771
V11            0.550799
V12            0.150074
V13            0.329218
V14           -0.234464
V15            1.653917
V16           -0.996330
V17            0.175095
V18           -0.522901
V19            0.133451
V20            0.374862
V21            0.057552
V22            0.617626
V23           -0.241337
V24            0.639175
V25            0.434682
V26           -0.168898
V27           -0.207384
V28           -0.316860
Amount        -0.242762
Class          0.000000
Name: 49073, dtype: float64
Prediction: legitimate
Actual Label: 0
```

Fig 8: Random Data Point Fraud Detection

```
]: print("\nFraud Detection for User Input:")
   input_data = []
   for i in range(X_test.shape[1]):
       feature_value = input("Enter the value of feature {}: ".format(i+1))
       input_data.append(float(feature_value))
   input_data = np.array(input_data).reshape(1, -1)
   y_pred = model.predict(input_data)
   if y_pred > 0.5:
       y_pred = 1
       print("The input data is classified as fraudulent.")
   else:
       y_pred = 0
       print("The input data is classified as legitimate.")

   Enter the value of feature 1:


   Fraud Detection for User Input:
```

Fig 9: User input Fraud Detection

```
Custom Data Point Fraud Detection:
Enter the index of the data point (row) you want to check: 1452
1/1 [==============================] - 0s 23ms/step
Selected data point:
Time      1127.000000
V1          -1.380849
V2          -0.145625
V3           0.126665
V4          -2.162818
V5           0.521847
V6          -1.741244
V7           0.019423
V8           0.381790
V9           0.344398
V10         -1.313241
V11          0.859797
V12          0.872693
V13         -0.550424
V14          1.003681
V15          0.388093
V16         -0.316377
V17         -0.316406
V18          0.540977
V19          1.011011
V20          0.205208
V21          0.189500
V22          0.234031
V23         -0.202100
V24          0.041873
V25         -0.080313
V26         -0.137458
V27          0.245276
V28         -0.024124
Amount      -0.349231
Class        0.000000
Name: 1452, dtype: float64
Prediction: legitimate
Actual Label: 0
```

Fig 10: Custom Data Point Fraud Detection

VI. CONCLUSION

In conclusion, the implementation of credit card fraud detection using neural networks and adversarial training offers promising results. The initial neural network model achieved high accuracy during training and performed well on the test set.

Improving the performance of the model in detecting fraudulent transactions is crucial. Further research and enhancements are needed to address the model's susceptibility to adversarial attacks and inability to correct identify fraudulent instances. Strategies such as exploring more sophisticated neural network architectures, incorporating additional features may be considered to enhance the fraud detection capabilities.

While the implemented approach shows promise, it is essential to continuously evaluate and refine the model to keep up with evolving fraud patterns and ensure reliable detection. Credit card fraud detection is an ongoing challenge, and further research and improvements are necessary to develop robust and accurate systems capable of effectively detecting and preventing fraudulent activities in real-time.

VII. FUTURE ENHANCEMENTS

Explore advanced adversarial training methods provide better defense mechanisms against sophisticated adversarial attacks.

Implement a mechanism to continuously update the model with new data and adapt to emerging fraud patterns.

Conduct rigorous adversarial testing and evaluation to identify vulnerabilities and weakness in the fraud detection system. Continuously challenge the model with new adversarial attacks and develop counter measures to improve its resilience against evolving attack techniques.

VIII. REFERENCES

[1] Thirunavukkarasu.M, Achutha Nimisha, Adusumulli Jyothsna "Credit Card Fraud Detection Using Machine Learning" IJCSMC, Vol.10, Issue 4, April 2021

[2] Emmanuel Ileberi, Yanxia Sun, Zenghui "A Machine learning based credit card fraud detection using the GA algorithm for feature selection" Journal of Big Data 2022 Doi:10.1186/s40537-022-00573-8

[3] Khatri S, Arora A, Agrawal AP. Supervised machine learning algorithms for credit card fraud detection: a comparison. In:10th international conference on cloud computing, data science & engineering (Confluence); 2020.p.680-683

[4] S P Maniraj, Aditya Saini, Swarna Deep Sarkar "Credit Card Fraud Detection using Machine Learning and Data Science" IJERT, Vol. 8 Issue 09, September-2019

[5] John Richard D. Kho, Larry A. Vea "Credit card fraud detection based on transaction behaviour" 2017 IEEE Region 10 conference (TENCON) , Malaysia, November 5-8

[6] Sahayasakila. V, D. Aishwarya Sikhakolli, Venkatavisalakshiseshai Yasaswi "Credit card fraud detection using smote technique and whale optimization algorithm" IJEAT Volume-8 Issue-5, June 2019

[7] P. Jayant, Vaishali and D. Sharma "Survey on credit card fraud detection techniques" IJERT Vol. 3 Issue 3, March-2014

[8] S. Esakiraj and S. Chidambaram "A predictive approach for fraud detection using hidden Markov model" IJERT Vol. 2 Issue 1, January -2013

[9] Ishu Trivedi, Monika, Mrigya, Mridushi "Credit card fraud detection" published by International Journal of Advanced Research in Computer and Communication Engineering Vol. 5, Issue 1, January 2016

[10] Quah, J. T. S., and Sriganesh, M. (2020) "Real time credit card fraud detection using computational intelligence. Expert systems with applications" 35(4), 1721-1732