Recurrent Neural Networks for Credit Card Fraud Detection

by

© **Ruben Chevez-Guardado**

A research paper submitted to VERAFIN. This work was supported by MITACS through the MITACS Accelerate program.

May 29, 2020

St. John's, Newfoundland and Labrador, Canada

Abstract

Fraudulent activity is hard to detect due to the low amounts of incidents among the millions of credit card transactions, but it can cost millions of dollars in monetary losses and legal costs to financial institutions. The lack of open-source fraud labeled data and the unbalanced nature of such, make it hard for engineers and computer/data scientists to design solutions to detect financial fraud. Recurrent Neural Networks (RNNs) are a promising machine learning approach for fraud detection due to their ability to learn patterns from time series. In this study, the use of RNNs for fraud detection was explored. To do this, first we gathered the BankSim synthetic and publicly available credit card transaction dataset. Next, RNNs variations were design and tried. Finally, a performance assessment of RNNs for fraud was made.

Table of contents

List of tables

List of figures

Chapter 1

Introduction

1.1 Problem Definition

In the digital age or fourth industrial revolution, fraud has become a significant problem for financial institutions [12]. Fraudsters continue to find new ways to exploit financial systems' vulnerabilities, leaving previous detection methods obsolete.

The focus of this study is the identification of credit card fraud transactions using RNN variants on publicly available datasets. There is a lack of fraud labeled datasets due to concerns of disclosing the financial institutions' client personal information. Luckily, there was one synthetically generated dataset based on real credit card transactions called BankSim [27]. This dataset has time-ordered transactions, which is a perfect environment to test RNNs [33].

1.2 Financial Fraud

1.2.1 Fraudsters Categories

There are multiple ways to commit fraud, and one or many individuals can be involved to accomplish such a goal. The following are categories used to identify individuals or groups committing fraud.

1. **First Party** - First party fraudsters are the users of the financial institution or

service trying to trick such services. A simple example is a bank user trying to submit a check multiple times [34].

2. **Second Party** - All fraud actions made inside the financial information are considered as second party fraud. For example, a bank employee creating false accounts and moving big amounts of money between these accounts [32].

3. **Third Party** - When a person not related to a financial institution or its users is involved in the theft of sensitive information to act using a false identity to make fraud transactions, it is considered as third party frauds. For example, when card information is stolen, or cloned and used to do withdrawals money or do online transactions [32].

1.2.2 Types of Fraud

The following are type of frauds and information used to identify them [**?**].

1. **Card Fraud** - Card frauds include credit and debit card transactions. Indicators of card frauds are unpaid balance, late payment days, consumption days, number of daily transactions, maximum and minimum ranges of transactions in set periods of time, and frequency of use.

2. **Money Laundering** - Money laundering detection can be based on the transaction amount, geographic location of the sender and recipient, source, date, and time of the transaction.

3. **Mortgage Fraud** - When a customer submits false information or overestimates value regarding a mortgage to access new credit. This prevents the bank from retrieving the full or partial repayment of the credit. The detection of mortgage fraud is based on the customer personal, professional, and mortgage information.

4. **Home Equity Line of Credit Fraud (HELOC)** - HELOC is a type of credit in which a customer borrows against the amount of home equity in a mortgage. If the payment is not done, the borrower can lose the house or property. Frauds can be done by a third party impersonating the homeowner with stolen information and rerouting the funds to other accounts. The detection of HELOC fraud is based on the customer personal and professional information.

5. **Check Fraud** - Check frauds include multiple deposits or withdrawals of the same check. Methods of identifying them are based on the check's information and identifier.

6. **Wire Transactions Fraud** Wire transactions are similar to Automated Clearing House(ACH) but with larger amounts and cannot be reversed. Fraudsters can obtain money using this method based on false representations or identity theft.

7. **ACH Fraud** - ACH and wire transfer methods transfer funds from one account to another but wire transfers are sent directly between account, while ACH transfers use the automated clearinghouse as an intermediary. ACH transfers can be reversed. Fraudsters can obtain money using this method based on false representations or identity theft. Canadian Payments Association(CPA) is the ACH alternative in Canada.

1.3 Fraud Detection Methods

The following are anomaly detection techniques and their categories [22].

1. **Statistical Based**

   (a) **Univariate** - Outliers found on a single space dimension or feature are called univariate outliers.

   (b) **Multivariate** - Outliers found on a multiple space dimensions or features are called multivariate outliers and can be hard for humans to understand when the number of dimensions are greater than a three-dimensional plane.

   (c) **Time series** - A sudden spike or dip in a series of data points ordered by time is consider an outlier or anomalous behavior.

2. **Knowledge Based**

   (a) **Rule-based Systems (Expert Systems)** - Set of human-crafted, fixed and inflexible decision-making rules, usually simple rules like "if-then" statements, to do an educated guess or prediction.

3. **Machine Learning**

   (a) **Neural Networks** - Algorithms inspired by the human brain neural architecture, and designed to recognize patterns. They are used together with an optimization algorithm, such as gradient descent, to minimize a loss function.

   (b) **Genetic Algorithms** - Algorithms that imitate the natural selection process where only the strongest individuals pass their information to the next generation.

   (c) **Reinforcement Learning** - Algorithms exposed to an environment, that learns from past experience, trial and error, and try to capture the best possible knowledge to make an accurate decisions.

   (d) **Tree Based Algorithms** - A model based on a series of decisions to determine the class label.

1.3.1 Public Data Sets

PaySim: A financial mobile money simulator for fraud detection (2016)

PaySim is a synthetic mobile payment transactions simulator [26]. It is based on a Multi Agent Based Simulation (MABS) and generates synthetic datasets based on a sample of real transactions from an anonymous mobile money service from an African country [20]. The goal of PaySim is solve the lack of publicly available fraud labeled payment transaction datasets. This problem is in big part due to the private nature of the transactions and the user information linked to them. One of the advantage of MABS is that allows to add behaviors on its simulation agents to accomplish a goal. For example, the fraudster agents can be tasked to do different goals such as the theft of sensitive information from the customer users. The simulation scope is to reach similar statistical properties as the original dataset. The simulator creates five types of transaction: "CASH-IN" (increasing the balance of an account by paying in cash to a merchant "CASHOUT" (withdraw cash from a merchant), "DEBIT" (sending the money from the mobile money service to a bank account), "PAYMENT" (paying for goods or services to merchants) and "TRANSFER" (sending money to another user).

A Paysim generated dataset is available in Kaggle and the source code is publicly available in Github [29, 25]. One thing to note is that the samples of data available in the Github source code is not the real African mobile money service data, but a synthetically generated from the real one. The author mentioned that he did not want to disclose the real data due to privacy concerns. This means that if anyone else want to run Paysim from the source code, it will be generating synthetic data from another synthetic dataset.

BankSim: A bank payment simulator for fraud detection research

BankSim is very similar to Paysim and was made by the same author, but it contains additional features [27]. It is also based in MABS but uses a sample of credit card transactions data between November 20112 and April 2013 from an anonymous bank in Spain to generate new data. The simulator output data is ordered by time and is based on three agent interactions from Fraudsters, Customers, and Merchants. Due to these attributes, the dataset is used in this study to test the proposed algorithms.

The goal of BankSim is to solve the lack of publicly available fraud-labeled financial transactions datasets. The output dataset contains information such as zip code location of origin/source, sixteen merchant categories, customer gender and age. The data was generated using population statistical information from the real sample. The sample contains averages, minimum and maximum scores for each feature. The fraud scenarios available are theft, cloned card/skimming, and internet purchases/Carding. These scenarios are made by playing with the fraudsters agents' parameters such as "customers proximity". The actions available in the simulator are "Find Merchant", "Offer Service", "Buy/Sell", "Steal Card/Information", "Abuse Purchasing", and "Report Block". BankSim uses the Multi-Agent Based Simulation toolkit MASON [6]. The BankSim generated dataset available in Kaggle was generated by running the simulator for 180 steps (approx. six months), producing in total 594643 records. Where 7200 are fraudulent transactions, and 587443 are normal transactions.

Anonymized credit card transactions

The dataset was developed during a research collaboration on big data and fraud detection between the Machine Learning Group of the Université Libre de Bruxelles

(ULB) and Worldline. The dataset is made of anonymous credit cards transactions from September 2013 in Europe and the features are the result of a Principal Component Analysis (PCA) transformation. All the features are named with the values V1 to V28 follow by the labels column. The real name of each feature was not disclosed due to confidentiality issues. The only features that were not transformed are "Time" and "Amount". The total number of transactions is 284,807 from which 492 or 0.172% are labeled as fraud [19, 17, 18, 16, 14, 15, 23].

1.4 Recurrent Neural Networks

Recurrent Neural Networks are neural networks designed to handle sequence data by using gated units that selectively propagate forward information from previous time steps. This is something traditional models are not able to do. Due to this feature, RNNs and its variations are very popular for time-series prediction, machine translation, speech synthesis, music composition, text summarizing, document and report generation, chat-bots, and more [24].

1.4.1 Advantages and Disadvantages

Recurrent neural networks have traditionally been difficult to train, as they contain millions of parameters, and have a vanishing and exploding gradients problems. Exploding gradients are the accumulation of large update values to the neural network model weights during training. Similar to exploding gradients, vanishing gradients are the opposite. This means that update values less than one are used during back propagation causing the model weights to shrink exponentially. Advances in network architectures have addressed most of these problems. The main advantage of recurrent neural networks over traditional models is that RNNs can model time series so that each data point can be assumed to be dependent on previous ones and learn patterns over time.

1.4.2 Types of Recurrent Neural Networks

1. **Long short-term memory (LSTM) -** A single LSTM cell have a forget, input, and output gates that control the cell states. The gates traditionally use sigmoid and hyperbolic tangent activation functions. The vanishing and exploding gradient are solved using the forget gate to control what information will remain in the cell's state.

2. **Gated Recurrent Unit (GRU) -** The GRU cell is simpler in architecture compared to LSTM. GRU cells have an update and forget gates that control the cell's state. Both LSTM and GRU have the ability to retain memory from previous time-steps.

3. **SimpleRNN -** SimpleRNN is a RNN variation found in the Keras library [1]. It does not contain any gates compared to the other variants and only contains a tangent activation function. A more detailed comparison is available in Section 3.4.

Chapter 2

Previous Work

Wiese and Omlin [35] tested three algorithms with real credit card transactions. The algorithms used were feed forward neural networks (FFNN), support vector machine (SVM), and long short-term memory recurrent neural network (LSTM). The dataset and features were not disclosed due to confidentiality issues. The goal of the study was to test whether LSTM outperformed the older techniques in fraud detection.

During the feature selection, the authors pointed at an interesting fact, that credit card transactions data contains multiple symbolic or categorical features. An example of this can be the identifiers or account numbers that connect the transaction between the user and merchant, addresses, and extra information that can be used by the financial system to create database table relationships. The researcher could be tempted to delete this features but there could be hidden patterns in them. For example, a fraudster can make multiple small withdrawals from different locations, and thus, avoiding detection from the financial systems. A pattern that can be found by using the geo-location features.

The first experiment was to compare FFNN vs SVM using a dataset containing a total of 30876 transactions over 12 months. The second experiment uses the same data but ordered in time together with the LSTM. The three models obtained excellent results but LSTM outperformed the other methods on the time-ordered dataset. LSTM had a bigger training time compared to SVM followed by the FFNN.

Malhotra et al. [31] generated a model made of multiple stacked recurrent hidden layers trained on four datasets to determine the efficacy of LSTM for anomaly

detection. The datasets used were an electrocardiogram, space shuttle valve, power demand, and a multi-sensor engine datasets. The authors chose LSTM due to its ability to handle time-series and overcome the vanishing gradient problem found in traditional RNN models. The model's architecture was made up of one input layer for each dimension or dataset feature followed by two fully connected recurrent layers and an output layer. The authors decided to give higher importance to precision over recall. Malhotra et al. concluded that "for the 'ECG' and 'engine' datasets, which do not have any long-term temporal dependence, both LSTM-AD and RNN-AD perform equally well. On the other hand, for 'space shuttle' and 'power demand' datasets which have long-term temporal dependencies along with short-term dependencies, LSTM-AD shows significant improvement of 18% and 30% respectively over RNN-AD in terms of F0.1-score."

Ando, Yoshihiro et al. [13] applied RNN to detect fraudulent behavior on web access logs. The dataset consisted of web server logs from users who attempted to do illegal purchases. This data was provided by Yahoo JAPAN between April 29, 2016 to May 5, 2016. One pattern found in the data was that the users were intending to steal credit card numbers to buy prepaid cards, such as Amazon gift cards. The data had the features access date, time, a unique cookie ID, and the destination host name ordered by time. To encode this dataset, they used specific target words and encoded binary values based on their absence or not. The results showed that traditional RNN had better performance than SVM but were comparable with LSTM. The only difference between RNN and LSTM results was that LSTM converged to a local minima faster.

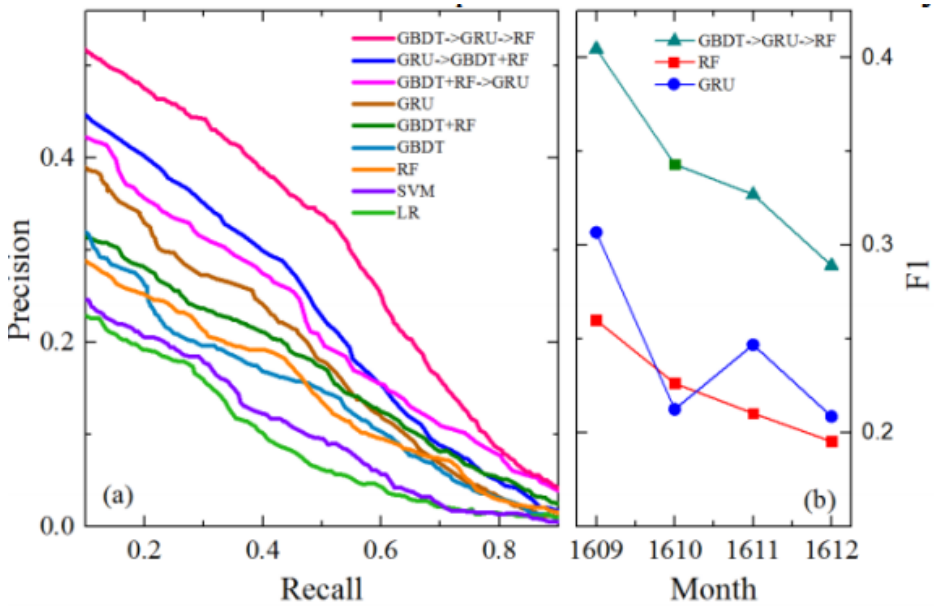
X. Li et al. [30] tested various combinations between RNN and ensemble models on a credit card fraud transaction dataset. The authors proposed an architecture called "within-between-within (WBW)", which consists of a RNN sandwiched between two ensemble models. The main reason of this mix was that RNN have a good performance in sequence analysis but do not have the ability of learning feature relationships like the ensemble models can. RNN cannot do feature learning at the single transaction level.

The data used in this study is from UnionPay within a three month period from June 2016 to August 2016. The explanation in the paper of how the data was processed is a bit confusing but in simple terms, the transactions were grouped by account. Next,

they were sorted by time and a moving sliding window was used to separate them in groups of fixed size. For earlier transactions which had less values before them than the size of the fixed sized batch, padding was introduced to fill the gap. The same method is used to process the BankSim data for our research.

The authors decided to use GRU instead of LSTM because of their simpler inner architecture. This reduced the number of matrix multiplication over a very large dataset, which also reduced the time of convergence. The architectures were made of Random Forest(RF), gradient boost decision tree (GBDT), and extreme gradient boosting (XGBoost). The list of proposed architectures were defined as follows GBDT+GRU+RF (WBW), GBDT+RF+GRU (WWB), and GBDT+RF (BWW) . X. Li et al. concluded that "WBW sequence learning architecture provides a better performance than that of WWB or BWW structures for our business scenario, let alone other simpler structures like WB or BW. In addition, model performance can be further enhanced by using various TS ranges and attention mechanism." Image 2.1 summarizes the models' performances.

Figure 2.1: X. Li et al. [30] demonstrated that WBW or the combination of GBDT-GRU-RF is the best performing architecture for the UnionPay dataset. Graph (a) shows the comparison of Precision-Recall curves. Graph (b) shows the comparison of variation trends for F1 scores as time elapses.



Chapter 3

Methodology

3.1 Data Analysis

3.1.1 Data Sets

Publicly available data sets are rare due to them containing users' sensitive information. For this reason, financial institutions prefer personalized in-house research, protected under non-disclosure agreements, and not available to the research community.

One alternative to this problem is the generation of synthetical data based on patterns found on real datasets. The new data follows patterns from the original but does not contain any sensitive information. The datasets used in the study come from "BankSim: A Bank Payment Simulation for Fraud Detection" [27] and "PaySim: a financial mobile money simulator for fraud detection" [26]. BankSim is the focus of this study, and PaySim's source code is only used to run the simulator and generate a new dataset bigger than BankSim's to test if increasing the data improves the RNN performance. PaySim's source code was used to generate new data since it is publicly available and BankSim is not. BankSim was made based on a multi-agent simulation with transactions that are spread over time and identified by customer id and merchant id. These features allow us to use it with Recurrent Neural Networks for time-series pattern identification.

BankSim and PaySim are described in detail in section 1.3.1. BankSim produced

587,443 regular payments and 7,200 fraudulent transactions. The simulation is randomized and not identical to the original data. The total value of the stolen amount is 4.8 million euros, which corresponds to a high crime rate of 17% of the total amount of payments and an average of 530 euros per fraud. PaySim produced 23 million records during one month of real-time data or 740 steps. The records are divided into the categories "CASH-IN", "CASH-OUT", "TRANSFER", "PAYMENT", and "DEBIT". For our final test where we used PaySim to generate a bigger dataset than BankSim's, the output data length was 10,510,453 compared to BankSim's 587,443 records.

3.1.2 Agents

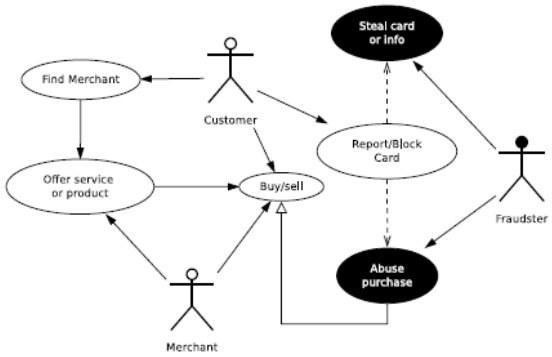
The BankSim simulation is based on three agents: **Merchant, Customer, and Fraudster**. The agents' interactions can be seen in Figure 3.1.

1. **Merchant:** The merchant provides a specific category of service or product to the customer agents. The agent has a geographic distribution by zip code and time.

2. **Customer:** The customers' objective is to request products from merchant and its only payment method is credit card. Only customer-merchant interactions are allowed, the simulator does not allow customer-customer transactions.

3. **Fraudster:** The fraudster agents can achieve one of the following goals: Theft, Cloned Card/Skimming, Merchant Hacking, and Internet Purchases. Based on the author's statement, the above goals can be made by tweaking the parameters: number of theft, zip code, frequency of unusual transaction, customer proximity sensing, and merchants affected.

Figure 3.1: Lopez-Rojas et. al. [27] described the BankSim's Customer vs. Merchant vs. Fraudster agents interactions.



3.1.3 Features

The BankSim dataset contains nine features and a target fraud/non-fraud column. The features columns are the following [28].

1. **Step:** This feature represents the day from the start of simulation. It has 180 steps or approximately 6 months of real-time transactions.
2. **Customer:** This feature represents the customer id.
3. **zipCodeOrigin:** The zip code of origin/source.
4. **Merchant:** This feature represents the merchant's id.
5. **zipMerchant:** The merchant's zip code.
6. **Age:** Table 3.1 summarizes the age categories.

ID	Age
0	Less than or equal to 18
1	19-25
2	26-35
3	36-45
4	46-55
5	56-65
6	Greater than 65
U	Unknown

Table 3.1: Age Ranges

7. **Gender:** Table 3.2 summarizes the gender categories.

ID	Gender
E	Enterprise
F	Female
M	Male
U	Unknown

Table 3.2: Gender

8. **Category:** Table 3.3 summarizes the services offered by the merchant agents.

Category	Percentage (%)	Average (Euro)	Standard Deviation (Euro)
Auto	49.00	224.36	267.52
Bars and restaurants	2.44	31.03	39.19
Books and press	00.14	33.35	45.01
Fashion	0.76	49.73	59.08
Food	7.26	32.49	30.87
Health	1.79	59.39	113.99
Home	0.21	75.48	121.77
Accommodation	0.16	97.41	86.85
Hypermarkets	1.78	33.06	35.78
Leisure	0.01	74.86	22.011
Other services	0.90	52.99	76.65
Sports and toys	0.43	74.80	75.45
Technology	0.31	67.28	108.68
Transport	81.76	24.56	20.77
Travel	0.04	577.46	518.42
Wellness and beauty	0.016	44.21	55.29

Table 3.3: Categories [28]

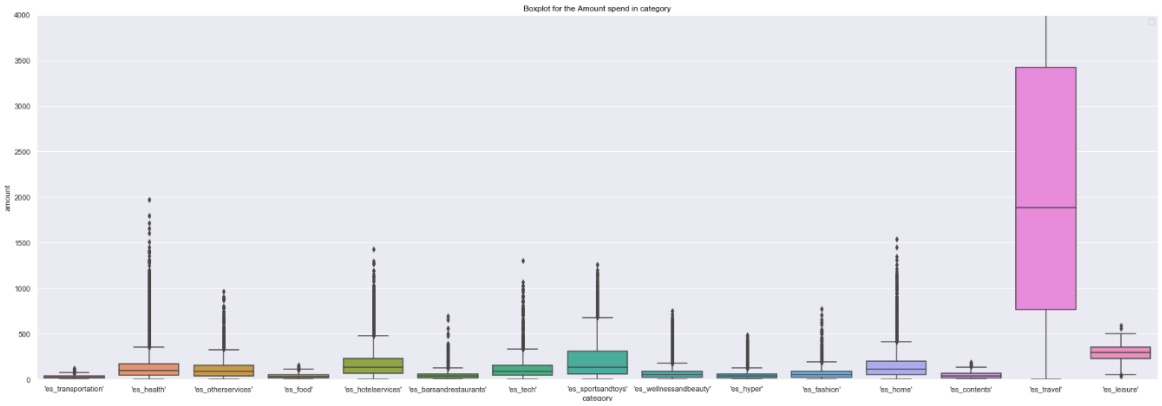
9. **Amount:** Purchase amount in the range of 0 to 8,330 and a mean of 37.9 euros.
10. **Fraud:** Target variable which shows whether the transaction is fraudulent (1) or benign (0).

3.2 Insights

During the analysis of the BankSim dataset, the following patterns were found:

1. Image 3.2 shows that “Travel” is the category with the highest spending.

Figure 3.2: BankSim contains 15 different categories. Each category describes a service offered by a merchant agent. Fraudsters' most targeted transaction classes are "Leisure", "Travel", and "Sports and Toys".



2. Table 3.4 shows that "Leisure", "Travel", "Sports and Toys" are the most selected categories for fraudsters.

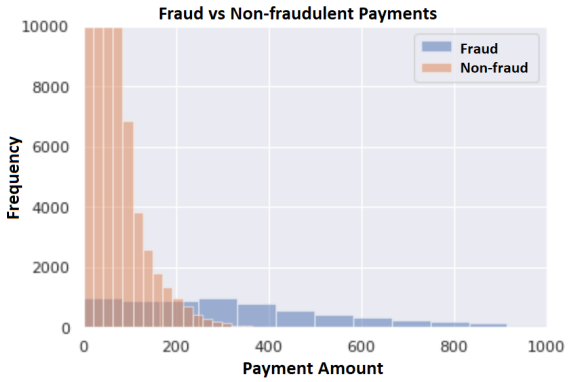
3. Table 3.4 and Image 3.2 show that fraudster agents prefer the categories which people spend more.

Category	Fraud Percent(%)
Transportation	0.0
Food	0.0
Hypermarkets	4.6
Bars and Restaurants	1.9
Contents	0.0
Wellness and Beauty	4.8
Fashion	1.8
Leisure	95.0
Other Services	25.0
Sports and Toys	49.5
Tech	6.7
Health	10.5
Hotel Services	31.4
Home	15.2
Travel	79.4

Table 3.4: Fraud Percent per Category

4. Image 3.3 and Table 3.5 show that fraudulent transactions are less frequent but have much higher amounts than benign transactions.

Figure 3.3: The comparison of fraud transactions vs non-fraud transaction amounts shows a significant difference in both frequency and quantity. Frauds are done less often but in higher quantities compared to benign transactions with a higher rate but with lower monetary value.



Amount Range(Euro)	Total (Euro)	Fraud (Euro)	Safe (Euro)	Fraud (%)	Safe (%)
0 - 555	592793.00	5430.00	587363.00	0.92	99.08
555 - 1110	1309.00	1257.00	52.00	96.03	3.97
1110 - 1665	143.00	121.00	22.00	84.62	15.38
1665 - 2221	81.00	75.00	6.00	92.59	7.41
2221 - 2776	65.00	65.00	0.00	100.00	0.00
2776 - 3331	59.00	59.00	0.00	100.00	0.00
3331 - 3887	48.00	48.00	0.00	100.00	0.00
3887 - 4442	48.00	48.00	0.00	100.00	0.00
4442 - 4997	38.00	38.00	0.00	100.00	0.00
4997 - 5553	21.00	21.00	0.00	100.00	0.00
5553 - 6108	14.00	14.00	0.00	100.00	0.00
6108 - 6663	15.00	15.00	0.00	100.00	0.00
6663 - 7219	3.00	3.00	0.00	100.00	0.00
7219 - 7774	5.00	5.00	0.00	100.00	0.00
7774 - 8329	1.00	1.00	0.00	100.00	0.00

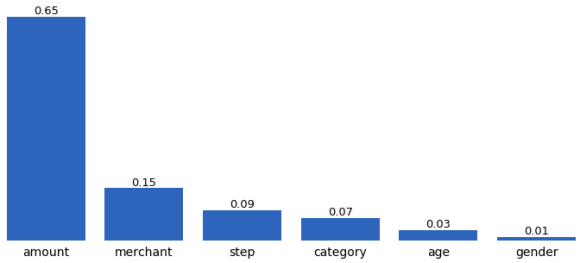
Table 3.5: Fraud Percentage and Values per Amount Range

5. Fraud occurs more in ages equal and below 18 (0th category).

6. The complete dataset contains 98.79% of normal transactions and 1.21% fraudulent transactions.

7. Image 3.4 shows the feature importance ranking generated by the random forest model. The feature "transaction amount" is the most influential feature in detecting (or classifying) a fraud.

Figure 3.4: Random Forest allows us to rank the most important features during training. Before doing any data alteration, the ranking shows that "amount" and "merchant", "step", and "category" are the most important features.



8. The BankSim [28] and PaySim [29] generated datasets are very similar in format, but BankSim's data output contains additional features. For example, it has demographic information such as age and gender.

9. BankSim's source code is private since it was generated using data from an anonymous bank in Spain. PaySim's source code is publicly available but with less features available than BankSim.

10. PaySim's simulator generates many possible scenarios. The creator mentioned that there was no direct way of selecting the best since we do not have access to the real data to compare. For now, we could pick the one with less value on the variable called "totalError".

11. BankSim and PaySim operate based on parameters and probabilistic values obtained from the real data and stored in the folder called paramFiles and "*.properties" file. One thing to note is that PaySim's source code is public but the probabilistic and parameter files included in the source code repository [25] is not from the real data but a synthetic generated approximation, since he did not want to disclose any real data values.

12. Both programs operate based on population statistics but there is no customer by customer-or merchant-specific behavior patterns generated by them. That could be the reason why increasing our batches of customer transactions doesn't make much a difference during training since the customer agents in the simulation do not have personal behavior patterns.

13. The customer and merchant agents in the simulation operate randomly but they try to approximate their spending frequency to the target population statistics.

14. Fraud transactions are generated by including fraudster agents that search for customer agents in the simulation and accomplish one of the following goals: Steal card/info, and abuse purchasing/excessive purchase. The customer agent can report or block the card if they detect abusive behavior. The author mentioned in the paper that by playing with the customer proximity we can change this behavior.

3.3 Data Transformation

3.3.1 Categorical To Numerical

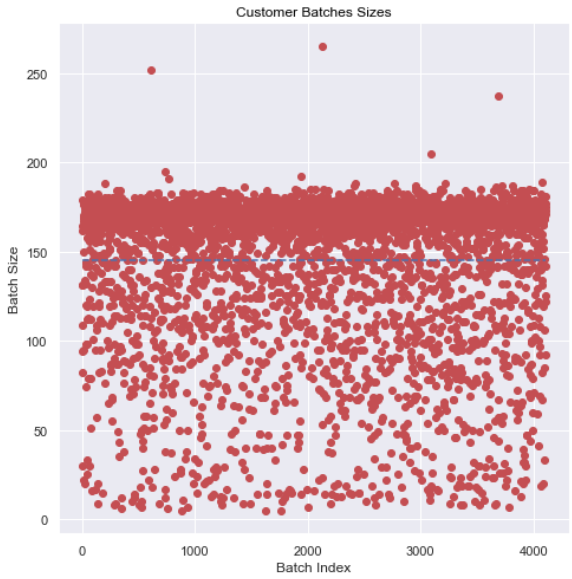
The first step is to convert all categorical features into numerical values. To accomplish this, we use the Pandas library method called "pandas.Series.cat.codes" to obtain a unique identifier for each categorical value, map them in a hash table and store it using the Pickle library in case there is a need to reconstruct the original dataset from the modified one. The modified features are age, gender, category, client, and merchant id. Both id columns are not discarded from the beginning because they will help group the transaction in batches further in the pipeline. The client and merchant zip code are discarded because they were considered irrelevant to the study, but it would be interesting to study further if the fraudulent agents have a geospatial type of movement pattern of targeting the customer agents in the simulation.

3.3.2 2D to 3D Dataset

All recurrent neural network models expect a three-dimensional input. We have two identifier columns, one for the customers and one for merchants. We also have two zip codes columns if we desire to group them by geospatial zones, but in this study, we will focus on grouping them by customers, so the rest of the identifier columns are discarded.

The BankSim dataset has 4,112 customers, so that we will end up with 4,112 batches, but the problem is that not all customers have the same number of transactions. We considered two options, creating fix sized batches with the maximum number of transactions and fill with a padding with a value of **-1** to those batches whose customer has fewer transactions than the maximum value or create batches with the mean value of transactions. Using the mean value of transactions per customer will reduce the amount of padding, but it will eliminate many transactions of those customers who have more transactions than the average. The average number of transactions is 144.6, and the maximum is 265 as shown in Figure 3.5. We decided to utilize batches with a size of 265, and even though it will have more padding, we avoided deleting data points. The final data shape is (4112, 265, 9).

Figure 3.5: By plotting the number of transactions per customer, it is clear that the majority concentrates on between 100 and 200 transactions. A few customers exceed above 200 transactions.



The original dataset had a 1.21% of fraud transactions, after grouping the transactions by customer, there was a 0.00009% probability of finding a fraud per customer batch. This was due to the amount of padding in the batches that were filled with a -1 value and categorized as non fraud. The value of -1 is changed at the end of the pipeline to zero and is only left with this value to be able to identify the padding values further in the pipeline.

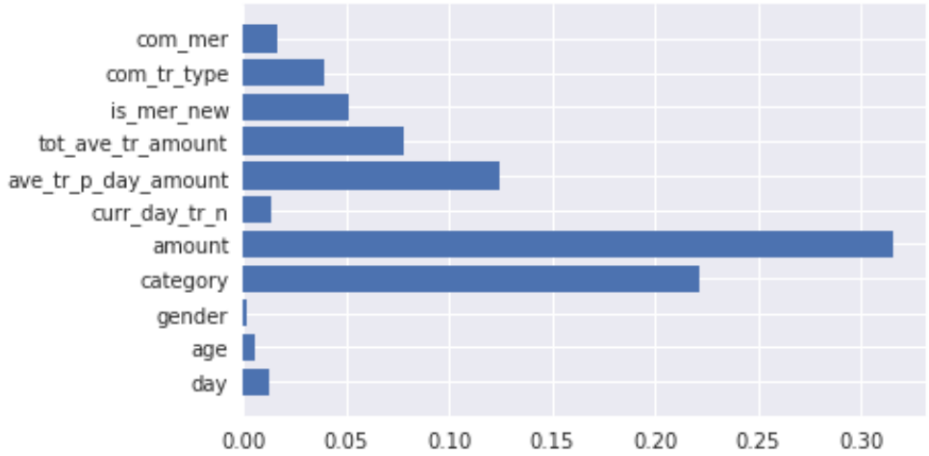
3.3.3 New Feature Generation

The most complex and time consuming part of the data pre-processing pipeline is to generate new features from the exiting data. The data generated is calculated inside each batch grouped by customer ID. The first loop looks at each customer batch, the second loop serves as a sliding window which looks at each transaction of the current batch and calculates the following condition using the data from the start to the current transaction. Figure 3.6 shows the feature importance ranking generated by the Random Forest model including the new features.

1. **curr_day_tr_n** - The number of transactions for the current day.

2. **ave_tr_p_day_amount** - The average transaction amount per day.
3. **tot_ave_tr_amount** - Total average transaction amount from the beginning to current time.
4. **is_mer_new** - Has this merchant been visited before?
5. **com_tr_type** - What is the most common transaction type?
6. **com_mer** - What is the most common merchant visited by the customer?

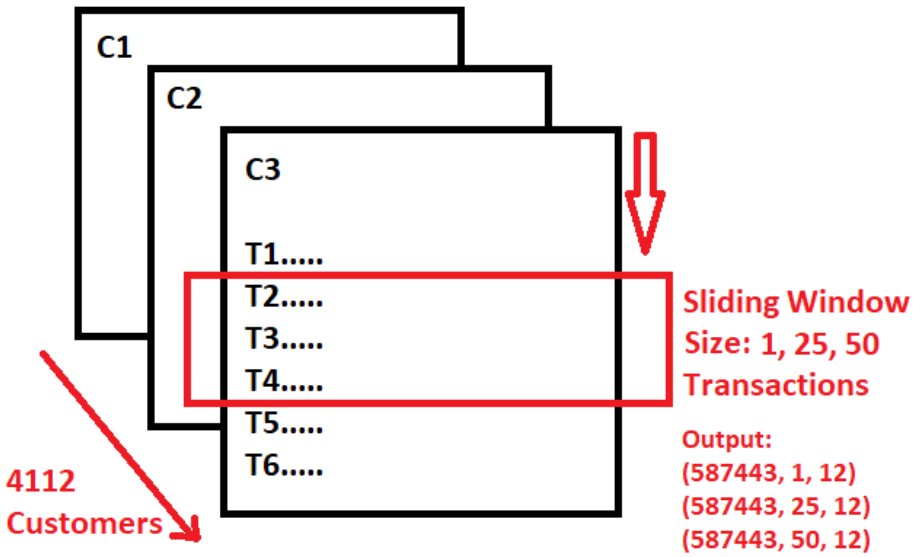
Figure 3.6: Generating each new feature consists of simple sums, averages, and Boolean conditions, but require multiple nested loops that iterate between batches, transactions, and features. Using Random Forest we can see that "amount", "category, and "average transactions per day" are the most important features.



Most of the calculation are simple sums, averages, or Boolean conditions but they require multiple nested loops inside each of the customer batches, which increases the computational time. After generating the new features in each customer batch, a sliding window is used to make smaller batches. This was done to avoid a python "Memory Error" caused by processing large number of batches and each batch containing a large amount of transactions. The size of the sliding windows depends on the user's preference of how far back in time the model will be able to look at. We used three scenarios, using bathes of size 1, 25 and 50 transaction. A sliding window of size 1 is not able to look at previous transactions and each batch has only one transaction, the next scenarios create smaller batches of 24 and 49 previous transactions plus the current transaction, The final data shapes including the new features are (587443, 1,

12), (587443, 25, 12) or (587443, 50, 12). The batch division and sliding window is similar to the strategy done by X. Li et al [30]. Figure 3.7 summarizes the process of generating new batches.

Figure 3.7: To create the 3D shape needed to train RNNs, first, we need to divide the transactions by customers, Then, sort by time in descending order. Finally, a sliding window of variable size is used to create batches of 1, 25, and 50 transactions per batch.



3.3.4 Normalization

The method called "sklearn.preprocessing.MinMaxScaler" is used to calculate the minimum and maximum values for each column and then map all the values in a range between 0 and 1. These values are stored in a binary file in case they are needed later to return the data to the original state.

3.3.5 Oversampling with SMOTE

This step is optional and is only used in one of the models to test the efficacy of using SMOTE over-sampling to improve performance. The model was labeled "DD_1B_OVERSAMPLED_RNN(GRU)". SMOTE works by choosing random example from the minority class and then find the k nearest neighbors for that example. A

randomly selected neighbor is chosen and a synthetic example is created at a randomly selected point between the two examples in the feature space [9].

We used the data generated with new features and only one transaction per batch because the Sklearn SMOTE only accepts two-dimensional inputs and not three-dimensional. We split the dataset into train, test, and validation sets and only the training set is over-sampled. The training set with shape (404357, 1, 12) is transformed to a 2D vector (404357, 12) which contains 4838 frauds and 399519 non-frauds and over-sampled to 399519 frauds and 399519 non-frauds. After this step, the dataset is again reshaped to a 3D vector to be able to feed it to the recurrent neural network.

3.3.6 Class Weights

Class weights are used in all models tested during the study to deal with the nature of the unbalanced dataset. Only in one of the models, called "DD_1B_DW_RNN(GRU)", we manually set the weights to check if increasing the fraud target weights could improve performance. To calculate class weights, we utilize the method "sklearn.utils.class_weight.compute_class_weight" and feed the values to the Keras models' fit method.

3.4 Model Architecture

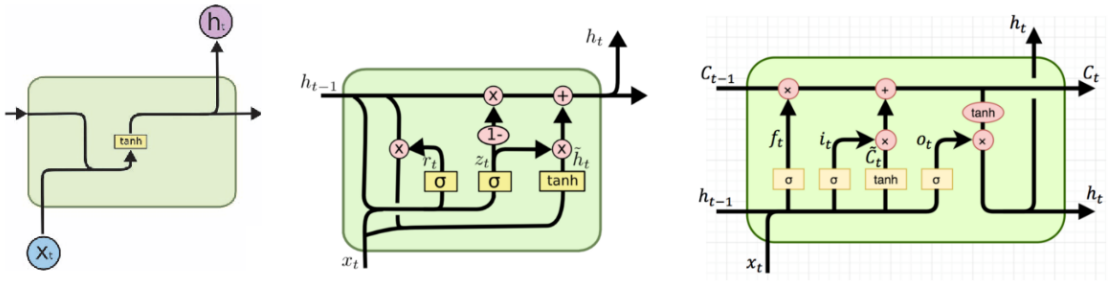
The focus of this study is on recurrent neural networks and its variations. The three recurrent neural layers used on the models are GRU, LSTM, and SimpleRNN.

3.4.1 Recurrent Neural Layers

As discussed in previous chapters, traditional neural networks are unable to learn from previous events because it lacks the ability to retain memory of those events. Image 3.8 shows the internal structure of each RNN cell variation.

Figure 3.8: The difference between SimpleRNN, GRU, and LSTM is the internal cell circuitry. GRU contains two gates to control the cell's state compared to LSTM's three gates. SimpleRNN does not have any gates, only an activation function that receives both the input and the previous time step output [8]

.



GRU

GRU is simpler in structure compared to LSTM. It has two gates instead of three. GRU use less training parameters and therefore use less memory, are better for much bigger datasets and shorter sequences when less resources are available. They execute and have training times faster than LSTM.

LSTM

LSTM are more accurate on dataset using longer sequence. These have widely been used for speech recognition, language modeling, sentiment analysis and text prediction.

SimpleRNN

Inside the SimpleRNN there is simple multiplication of the input and previous output. The result passes through an activation function. This layer does not contain any Forget or Update gates.

3.4.2 Fully Connected Layers

Dropout

The dropout layer randomly sets input units to 0 with a frequency of rate at each step during training time, which helps prevent over-fitting. Dropout is only applied during training [3].

L1 Regularization

Regularization, similar to dropout, is another method to avoid over-fitting. The two regularization methods are L1 and L2. Regularization apply penalties to layer parameters during optimization. The regularization is applied to each Keras layer [4].

3.4.3 Callbacks

Callbacks are custom methods called during training, evaluation, or inference. The callbacks are summarized in Table 3.6.

Callback	Description
EarlyStopping	This method is activated if the model is not able to minimize the loss function anymore. It requires a metric to track during training. The metric used in this study is Recall because we are interested in identifying as many correct fraud transactions as possible due to the dataset highly imbalanced nature.
ModelCheckpoint	Similar to the EarlyStopping callback, ModelCheckpoint tracks a metric variable during training and saves a copy of the model in case the training is stopped due to an error or user intervention. The metric used is Recall.
CSVLogger	This callback saves each epoch results to a csv file.
TensorBoard	This callback allows to visualize the training progress and results.
ProgbarLogger	This callback is added automatically during training and prints the results to console.

Table 3.6: Callbacks

3.5 Model Training and Selection

Hyper-parameters are parameters that are not directly learnt within estimators. In scikit-learn they are passed as arguments to the constructor of the estimator classes.

3.5.1 Adam Optimizer

Adam is a popular optimization algorithm due to its high ability to converge in a shorter time compared to other optimization methods and made specifically for neural networks. According to [21], the method is "computationally efficient, has little memory requirement, invariant to diagonal rescaling of gradients, and is well suited for problems that are large in terms of data/parameters".

3.5.2 Binary Cross Entropy Loss Function

The cross-entropy loss is used when there are only two label classes (assumed to be 0 and 1).

3.5.3 Stratified K-Fold Cross Validation

K-fold is a method to split the data in k partitions where one of them is used for testing. The process is repeated k number of times until all partitions are used for testing. Stratification means that each partition will contain the same percentage of each class of transactions. K-Fold is used together with grid-search to select the best performing model [10].

3.5.4 Grid Search

GridSearchCV is a Sklearn method that considers all parameter combinations. It generates candidates from a grid of parameters and selects the best performing during training [2]. The KerasClassifier wrapper is used to be able to combine Keras model inside the Sklearn grid search [5]. Table 3.7 summarizes the combination used to train and test during grid search.

Hyper-parameter	Values
Number of hidden layers	0, 1, 2
Number of RNN Layers	1, 2
Number of neurons on each RNN layer	25, 50, 100
Number of neurons on each hidden layer	50, 100, 200, 300
Loss function	Binary Cross Entropy
Number of epochs	50
Optimizer	Adam
Model types	“LSTM”, “GRU”
Output layer activation	Sigmoid
Hidden layer activation	Sigmoid
Use dropout	True
Dropout rate	0.2

Table 3.7: GridSearchCV Hyper-parameters

The best model found during grid-search contains the following values: Number of hidden layers : 2, Number of RNN layers : 1, Number of neurons in RNN Layer : 50, Number of neurons on each hidden layer : 300, Loss function: Binary Cross Entropy, Number of Epochs: 50, Optimizer: Adam, Model Types : GRU, Output Layer Activation: Sigmoid, Hidden Layer Activation: Sigmoid, Use Dropout: True, Dropout Rate: 0.2. The model’s performance is shown in Image 3.9.

Figure 3.9: The best grid-search model consists of one GRU input layer, two fully connected hidden layers, and an output layer. It achieved a relatively high Precision-Recall AUC performance of 77% in the BankSim’s highly unbalanced data with only 1.2% of fraud transactions.

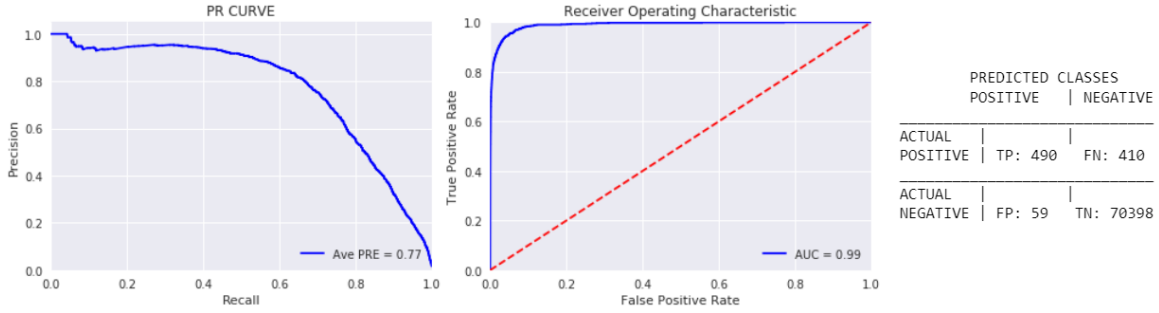


Table 3.8 shows the top 7 models including the ones used to test against the baseline validation models. The best model has the tag "BEST_MODEL(GRU)" and the others follow in descending order.

ID	Hyper-parameters	Train Acc(%)	Train Pre(%)	Train Rec(%)	Train ROC(%)	Test Acc(%)	Test Prec(%)	Test Rec(%)	Test ROC(%)
BEST_MODEL(GRU)	hidden_layers: 2, hidden_layers_neurons: 300, modelType: GRU, rnn_hidden_layers: 1, rnn_hidden_layers_neurons: 50	0.99	0.77	0.66	0.99	0.99	0.77	0.67	0.99
2BEST_RNN(GRU)	hidden_layers: 2, hidden_layers_neurons: 200 modelType: GRU, rnn_hidden_layers: 1, rnn_hidden_layers_neurons: 100,	0.99	0.81	0.63	0.99	0.99	0.81	0.63	0.99
GRU_1	hidden_layers: 2, hidden_layers_neurons: 300, modelType: GRU, rnn_hidden_layers: 1, rnn_hidden_layers_neurons: 100	0.99	0.87	0.59	0.99	0.99	0.87	0.59	0.99
LSTM_1	hidden_layers: 2, hidden_layers_neurons: 200, modelType: LSTM, rnn_hidden_layers: 1, rnn_hidden_layers_neurons: 100	0.99	0.83	0.58	0.99	0.99	0.83	0.58	0.99
GRU_2	hidden_layers: 2, hidden_layers_neurons: 200, modelType: GRU, rnn_hidden_layers: 1, rnn_hidden_layers_neurons: 50	0.99	0.85	0.58	0.99	0.99	0.85	0.58	0.99
LSTM_2	hidden_layers: 2, hidden_layers_neurons: 300, modelType: LSTM, rnn_hidden_layers: 1, rnn_hidden_layers_neurons: 100	0.99	0.82	0.58	0.99	0.99	0.81	0.58	0.99

Table 3.8: Top 7 GridSearchCV Models

3.5.5 Models

Table 3.9 summarizes all the models tested during this study and their performance results in descending order. The changes made to each model are mentioned in the description column.

ID	Description	PR-AUC	ROC-AUC
RANDOM _FOREST	Model used as baseline during inference in the validation set.	**0.91**	0.99
XGBOOST	Model used as baseline during inference in the validation set.	**0.91**	0.99
DD_1B_ OVERSAMPLED_RNN (GRU)	This model has the same hyper-parameters as the best grid-search models but was trained on over-sampled data using SMOTE. The training set was increased to 60% by decreasing the test set and validation set to 20% each. The original split was 50% , 30% , and 30% , respectively. It has the best performance from all the RNN variations.	**0.84**	0.99
DD_2RNN _LYRS_RNN (GRU)	This model was created with almost the same hyper-parameters as the best grid search model but with two GRU layers instead of one. It also has an increased training set.	**0.83**	0.99

DD_1B _RNN (GRU)	This model was created with the same hyper-parameters as the best grid search model but with an increased training set and only 1 transaction per batch. The original batch size is 25 transactions per batch.	**0.81**	0.99
DD_2B_RNN (GRU)	This model was created with the same hyper-parameters as the best grid search model but with an increased training set and 50 transaction per batch or double the original size.	**0.82**	0.99
DD_RNN (GRU)	This model was created with the same hyper-parameters as the best grid search model but with an increased training set. The reason for this change was to test if there could be an improvement by increasing the training set.	**0.81**	0.99
DD_1B _1L_RNN (GRU)	This model was created with almost the same hyper-parameters as the best grid search model but with only one dense layer and trained only with 1 transaction per batch. The reason to reduce the model size was to test if having a simpler model with smaller batches could improve its performance.	**0.80**	0.99
DD_HB_RNN (GRU)	This model was created with the same hyper-parameters as the best grid search model but with an increased training set and 12 transaction per batch or almost half the original size with 25 transactions.	**0.79**	0.99
DD_1B _DW_RNN (GRU)	This model was created with the same hyper-parameters as the best grid search model but with modified weights. This model has double the value automatically computed and assigned to the positive class by the "sklearn.utils.class_weight.compute_class_weight" method.	**0.79**	0.99
BEST _RNN (GRU)	The best model performing model during grid-search. The parameters used can be found in the previous table.	**0.79**	0.99
DD_1B_SRNN _RNN(GRU)	This model was created with almost the same hyper-parameters as the best grid search model but with only one SimpleRNN instead of the GRU layer and trained only with 1 transaction per batch. The reason of using a simpler RNN layer was to test if having a simpler model with smaller batches could improve its performance.	**0.78**	0.99
2BEST _RNN (GRU)	The second best performing model during grid-search. It uses 200 neurons per hidden layer instead of the 300 used by the best model.	**0.59**	0.98
GRU_1	The third best model obtained during grid-search. It uses 100 neurons on the RNN layers instead of the 50 used in the best model.	**0.59**	0.98
GRU_2	Similar to the best grid-search model but only uses 200 neurons in the hidden layers instead of 300.	**0.57**	0.98
LSTM_2	Similar to the best grid-search model but only uses an LSTM layer instead of GRU and 200 neurons in the hidden layers instead of 300.	**0.55**	0.98
LSTM_1	Similar to the best grid-search model but only uses an LSTM layer instead of GRU and 100 neurons in the RNN layers instead of 50.	**0.54**	0.98
DDL1_RNN (GRU)	This model was created with the same hyper-parameters as the best grid search model but with an increased training set and L1 regularization in the fully connected layers.	**0.02**	0.54

Table 3.9: Models

3.6 Model Validation

All the models described above are compared using the Precision-Recall and Receiver Operating Characteristic (ROC) curve versus an XGBoost and Random Forest model. Both models are trained using the default parameters in both BankSim and Paysim data. The results are available in the following chapter.

3.6.1 Baseline Models

XGBoost

XGBoost is a popular gradient boost algorithm. XGBoost models are known due to their fast execution speed and model performance. This fact was proved during training when it took only 20 minutes to train the model on the BankSim dataset compared to 554 minutes it took the best grid-search GRU model [11].

Random Forest

Random forest is an ensemble method that can be used for classification or regression. It works by creating multiple decision tree models and averaging the predictions of all its trees to improve the predictive accuracy and avoid over-fitting. The random forest model is trained using the default sklearn values and prove to have a very high accuracy and a short training time, close to the XGBoost model [7].

3.7 Creating More Data Using PaySim

Our last test was with PaySim. Paysim's source code located publicly in GitHub [25] was compiled and run to test if increasing the amount of data could increase the model performance and it did. Paysim's features are very similar to BankSim, but they lack certain features like age and gender. This missing features were left with zero or the label "Unknown" for the categorical values. We increased the number of steps from 720 to 1440, which is equivalent to 48 months of real-time transactions. The final data output was 10,510,453 compared to BankSim's 587,443 records. Table 3.10 shows the parameters used to run PaySim. The number of steps was increased

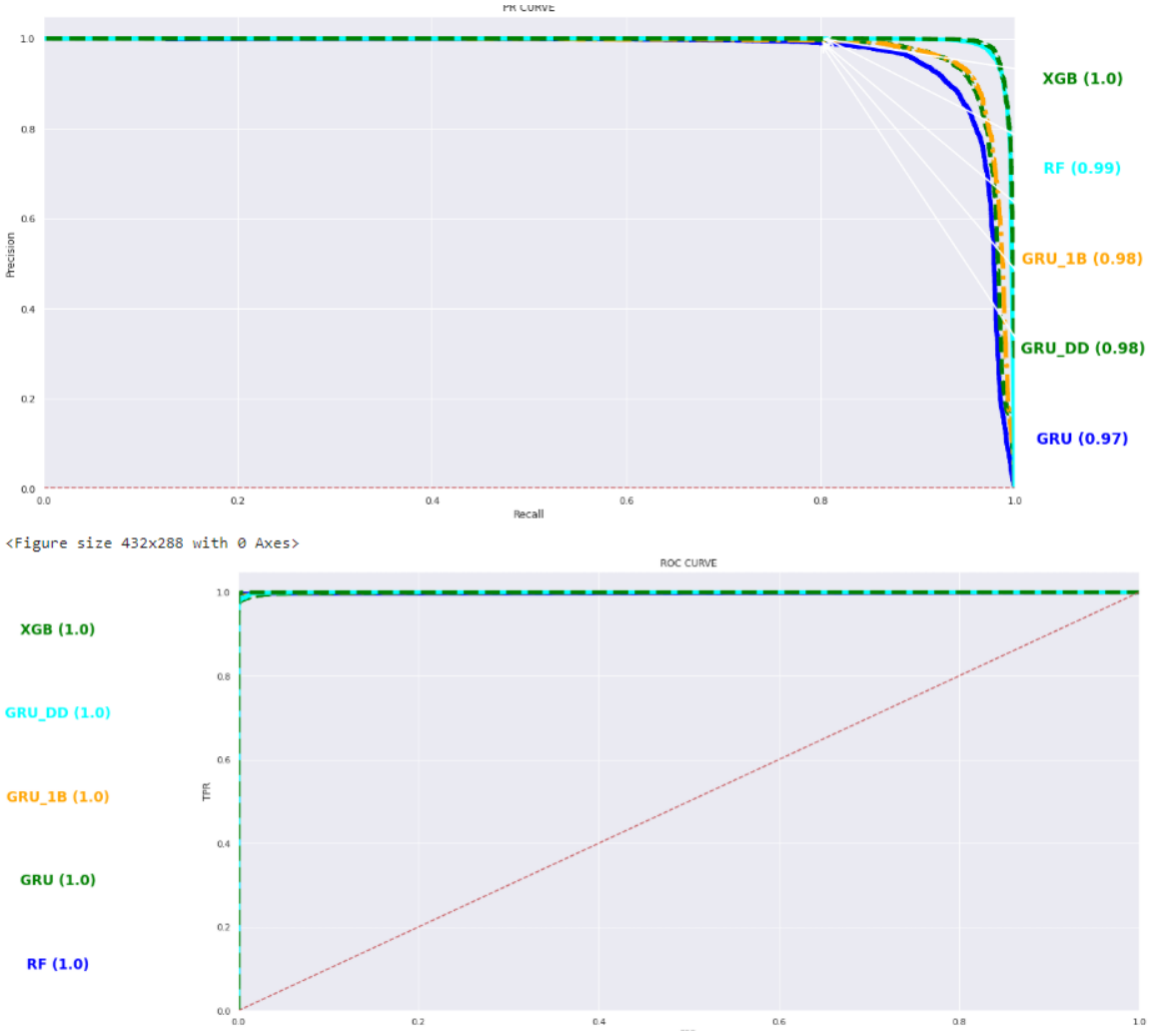
from 720 to 1440 to have a bigger data set. The rest of the parameters were left as default.

Parameter	Value
seed	107842350
nbSteps	1440 instead of the default value, 720
multiplier	1.0
nbFraudsters	1000
nbMerchants	6000
fraudProbability	0.01
transferLimit	2.0E10
transactionsTypes	transactionsTypes.csv
aggregatedTransactions	aggregatedTransactions.csv
clientsProfilesFile	clientsProfiles.csv
initialBalancesDistribution	initialBalancesDistribution.csv
maxOccurrencesPerClient	maxOccurrencesPerClient.csv

Table 3.10: PaySim Parameters

We modified the pipeline to be able to handle Paysim's data and we end up with a data shape of (10,510,453, 25, 12). The best grid-search architecture was used and due to time constrains, there was no time for more optimizations so only a train (8408362, 25, 12) and test (2102091, 25, 12) split was made. This model then was trained and tested versus the XGBoost and Random Forest models. This prove to be a heavy load for our servers which crashed every time we started training our GRU model on such a big dataset. To solve it we decrease the data size and only left one transaction per batch (GRU) (10510453, 1, 12). The model's performance improved significantly but remained below XGBoost and Random Forest. To test if a bigger batch size could improve performance, the number of batches were reduced to the same number obtained with BankSim (GRU 3D) (587,443, 25, 12) and double the data size of BankSim's dataset (GRU 3D DOUBLE DATA) (1174886, 25, 12). Increasing the batch size and the training set increased the performance but remained below the validation methods. Figure 3.10 summarizes the models' results on the PaySim data set.

Figure 3.10: PaySim Models PR & ROC Curve



Chapter 4

Results and Discussion

Evidence of the following statements can be seen in Image 4.1 and Image 4.2.

1. The models with GRU layers performed better than the models with LSTMs during training using the BankSim dataset.This might be due to lack of training data. As LSTM models have more weights to fit they require more training data.

2. Training the model with a SMOTE over-sampled version of the original BankSim, improved the RNN model's performance.

3. Using L1 regularization with a value of 0.01 on the best grid-search architecture's dense layers decrease dramatically the model's performance. This can be seen in the "DDL1_RNN(GRU)" model's Precision-Recall curve. Thus, there was an under-fitting problem rather than over-fitting.

4. Decreasing the batch size improved the model's performance. This can be because the synthetical data does not have customer specific patterns of behavior but instead, it was made with population based parameters.This can be seen in the "DD_1B_RNN(GRU)" model with one transaction per batch and the "DD_2B_RNN(GRU)" model with 12 transaction per batch plotted in the Precision-Recall curve.

5. RNNs had a close performance to Random Forest and XGBoost but remained below them. We can conclude based on the results that BankSim and PaySim data have a lack of long or short term temporal dependence since both simulators

focus on reaching general population based parameters from the real data sample instead of customer specific behavioral patterns.

6. Using two stacked GRU layers instead of one, increase the model's recall performance but remained relatively similar in precision compared to previous models.

7. Using the SimpleRNN layers decreased the model's performance but remained above the LSTM models' performance. This results can be interpreted as SimpleRNN are too simple to learn the complex patterns but LSTM layers under-fit in this specific dataset. GRU are the middle ground and the best performing in this dataset. SimpleRNN do not have Forget and Update gates, compared to LSTM and GRU. This makes it closer to transitional neural network units than RNNs.

8. Modifying the classes' weights did not show any improvement.

9. During grid-search, the architectures with greater number of layers and neurons per layers had the better performance. This shows that there could still be room for optimization because, due to time constrains, there was no chance to test deeper and wider architectures.

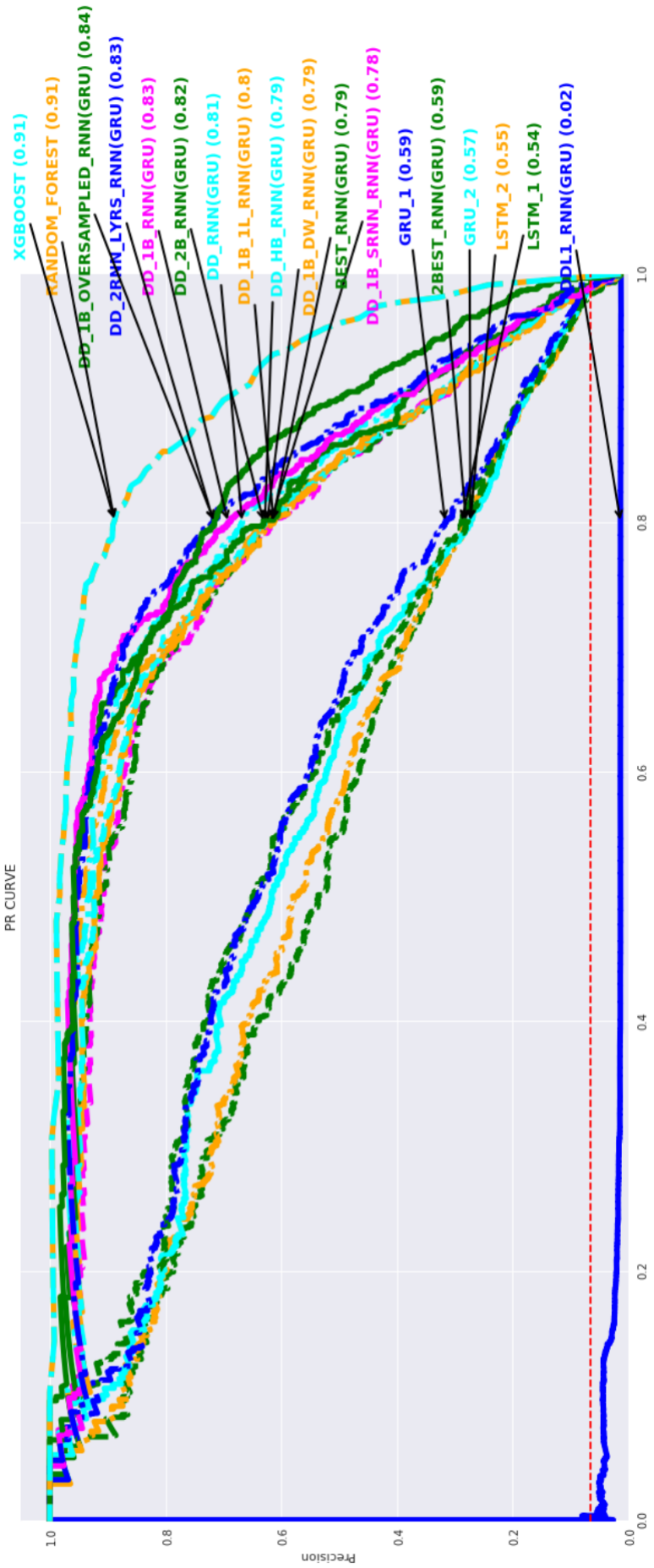


Figure 4.1: PR Curve on BankSim's Dataset

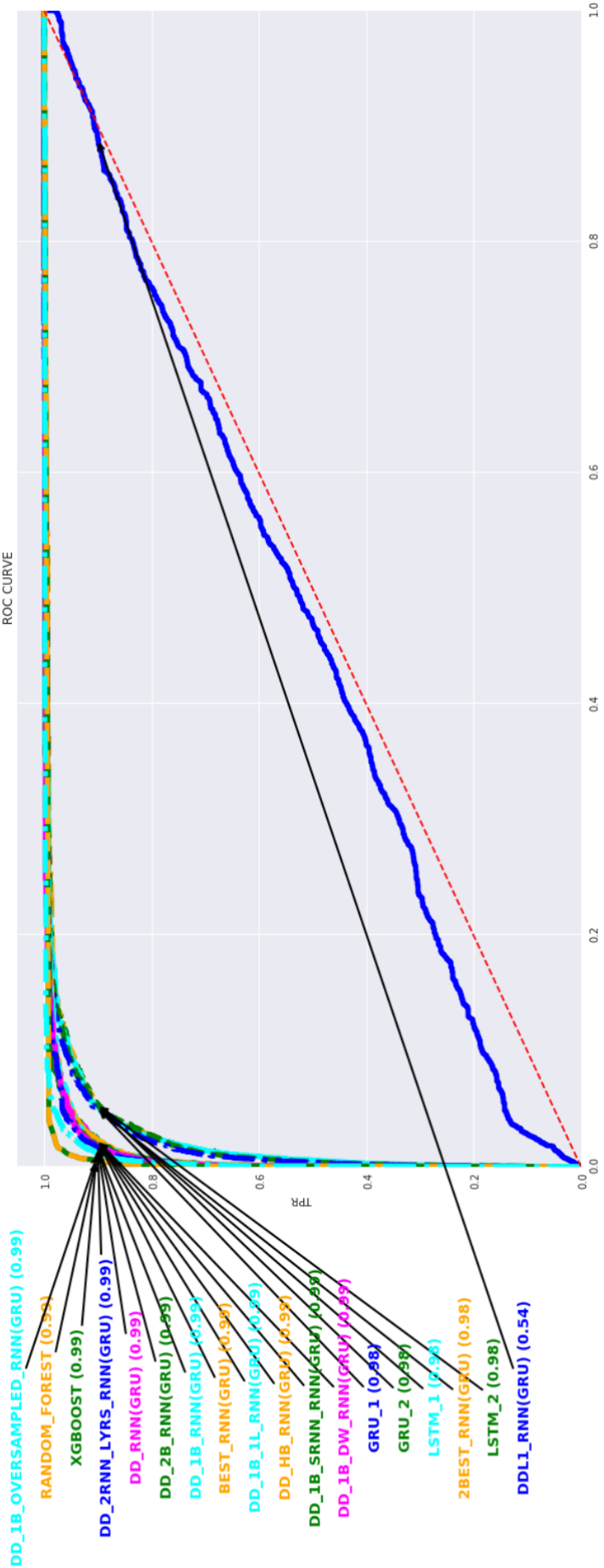


Figure 4.2: ROC Curve on BankSim's Dataset

Chapter 5

Conclusions

In this paper, we presented an alternative solution to build a transaction fraud detection model using recurrent neural networks. The study let us to conclude the following:

1. Both PaySim and BankSim operate based on population statistics but there is no customer-by-customer or merchant-specific behavior patterns generated by them. That could be the reason why increasing our batches of customer transactions doesn't make much a difference during training since the customer agents in the simulation do not have personal behavior patterns.

2. Even though we could not reach the level of performance obtained by tree models, it can be concluded that there is still room for optimization and improvement in our RNN models. This can bee seen during our different tests and grid-search assessment. The best grid-search architecture had the maximum number of dense layers and number of neurons provided by the hyper-parameter list of possibilities, which tells that there could be room for deeper and wider models.

3. Increasing the the amount of training data definitely improved the model's performance. This is common to see in models based on fully connected layers.

4. SMOTE over-sampling, dropout, and class weights were important tools that helped improve the performance on the BankSim's highly unbalance data with a ratio of 1.2% fraud versus 98.8% benign.

Bibliography

[1] Keras - SimpleRNN . https://www.tensorflow.org/api_docs/python/tf/keras/layers/SimpleRNN. (Accessed on 05/27/2020).

[2] GridSearchCV — scikit-learn 0.23.1 documentation. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html. (Accessed on 05/23/2020).

[3] Keras - Dropout Layer. https://keras.io/api/layers/regularization_layers/dropout/. (Accessed on 05/23/2020).

[4] Keras - Layer Weight Regularizers. https://keras.io/api/layers/regularizers/. (Accessed on 05/23/2020).

[5] KerasClassifier. https://www.tensorflow.org/api_docs/python/tf/keras/wrappers/scikit_learn/KerasClassifier. (Accessed on 05/23/2020).

[6] MASON multiagent simulation toolkit. https://cs.gmu.edu/ eclab/projects/mason/. (Accessed on 05/23/2020).

[7] RandomForestClassifier — scikit-learn 0.23.1 documentation. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html. (Accessed on 05/23/2020).

[8] Simple RNN vs GRU vs LSTM - Difference Lies in More Flexible control. https://medium.com/@saurabh.rathor092/simple-rnn-vs-gru-vs-lstm-difference-lies-in-more-flexible-control-5f33e07b1e57. (Accessed on 05/23/2020).

[9] SMOTE — imbalanced-learn 0.5.0 documentation. https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.over_sampling.SMOTE.html. (Accessed on 05/23/2020).

[10] StratifiedKFold — scikit-learn 0.23.1 documentation. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html. (Accessed on 05/23/2020).

[11] XGBoost Documentation. https://xgboost.readthedocs.io/en/latest/. (Accessed on 05/23/2020).

[12] AFP Fraud Survey 2020 Report Highlights. https://www.jpmorgan.com/content/dam/jpm/commercial-banking/documents/fraud-protection/afp-fraud-survey-2020-report-highlights.pdf, 2020 April. (Accessed on 05/25/2020).

[13] Y. Ando, H. Gomi, and H. Tanaka. Detecting fraudulent behavior using recurrent neural networks. In *Computer Security Symposium*, 2016.

[14] F. Carcillo, A. Dal Pozzolo, Y.-A. Le Borgne, O. Caelen, Y. Mazzer, and G. Bontempi. SCARFF : a Scalable Framework for Streaming Credit Card Fraud Detection with Spark. *Information Fusion*, 41, Sept 2017.

[15] F. Carcillo, Y.-A. Le Borgne, O. Caelen, Y. Kessaci, F. Oblé, and G. Bontempi. Combining unsupervised and supervised learning in credit card fraud detection. *Information sciences*, 2019.

[16] A. Dal Pozzolo, G. Boracchi, O. Caelen, C. Alippi, and G. Bontempi. Credit card fraud detection: A realistic modeling and a novel learning strategy. *IEEE Transactions on Neural Networks and Learning Systems*, PP:1–14, Sept 2017.

[17] A. Dal Pozzolo, O. Caelen, R. A. Johnson, and G. Bontempi. Calibrating probability with undersampling for unbalanced classification. In *2015 IEEE Symposium Series on Computational Intelligence*, pages 159–166. IEEE, 2015.

[18] A. Dal Pozzolo, O. Caelen, Y.-A. Le Borgne, S. Waterschoot, and G. Bontempi. Learned lessons in credit card fraud detection from a practitioner perspective. *Expert Systems with Applications*, 41:4915–4928, 08 2014.

[19] U. L. de Bruxelles. Anonymized credit card transactions labeled as fraudulent or genuine. https://www.kaggle.com/mlg-ulb/creditcardfraud, 2013. (Accessed on 05/23/2020).

[20] A. Drogoul, D. Vanbergue, and T. Meurisse. *Multi-agent Based Simulation: Where Are the Agents?* Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[21] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2015.

[22] L. N. Lata, I. A. Koushika, and S. S. Hasan. A comprehensive survey of fraud detection techniques. *International Journal of Applied Information Systems*, 10(2):26–32, 2015.

[23] B. Lebichot, Y.-A. Le Borgne, L. He, F. Oblé, and G. Bontempi. *Deep-Learning Domain Adaptation Techniques for Credit Cards Fraud Detection*, pages 78–88. 01 2019.

[24] Z. C. Lipton, J. Berkowitz, and C. Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015.

[25] E. Lopez. EdgarLopezPhD/PaySim. https://github.com/EdgarLopezPhD/PaySim, 2015. (Accessed on 05/23/2020).

[26] E. Lopez-Rojas, A. Elmir, and S. Axelsson. Paysim: A financial mobile money simulator for fraud detection. In *28th European Modeling and Simulation Symposium, EMSS, Larnaca*, pages 249–255. Dime University of Genoa, 2016.

[27] E. A. Lopez-Rojas and S. Axelsson. Banksim: A bank payment simulation for fraud detection research. *26th European Modeling and Simulation Symposium, EMSS 2014*, 09 2014.

[28] E. A. Lopez-Rojas and S. Axelsson. Synthetic datasets generated by the banksim payments simulator. https://www.kaggle.com/ntnu-testimon/banksim1, 2014. (Accessed on 05/23/2020).

[29] E. A. Lopez-Rojas, A. Elmir, and S. Axelsson. Synthetic financial datasets for fraud detection. https://www.kaggle.com/ntnu-testimon/paysim1, 2016. (Accessed on 05/23/2020).

[30] X. Lp, W. Yu, T. Luwang, J. Zheng, X. Qiu, J. Zhao, L. Xia, and Y. Li. Transaction fraud detection using gru-centered sandwich-structured model. In *2018 IEEE 22nd International Conference on Computer Supported Cooperative Work in Design ((CSCWD))*, pages 467–472. IEEE, 2018.

[31] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal. Long short term memory networks for anomaly detection in time series. In *Proceedings*, volume 89. Presses universitaires de Louvain, 2015.

[32] A. Ptak-Chmielewska and A. Matuszyk. Profile of the fraudulent customer. *Bezpieczny Bank*, 2(59):7–23, 01 2015.

[33] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[34] C. Vivian Amasiatu and M. Hussain Shah. First party fraud: a review of the forms and motives of fraudulent consumer behaviours in e-tailing. *International Journal of Retail Distribution Management*, 2014.

[35] B. Wiese and C. Omlin. Credit card transactions, fraud detection, and machine learning: Modelling time with lstm recurrent neural networks. In *Innovations in neural information paradigms and applications*, pages 231–268. Springer, 2009.