

Recurrent Neural Networks for Credit Card Fraud Detection

by

© Ruben Chevez-Guardado



A research paper submitted to ~~MITACS and~~ VERAFIN in partial fulfillment of the requirements for 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 per sample, 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 solve financial fraud problems. The first step during the study is to gather a publicly available dataset. Next is to design a supervised model that will be able to handle unbalanced and time-ordered transactions. The candidates and focus of this study are Recurrent Neural Network variations due to their ability to learn patterns from time series. [31] Additional to the theoretical analysis, the study also includes practical implementation of RNNs on the BankSim synthetic and publicly available credit card transaction dataset. [25] [24]



Table of contents

Title page	i
Abstract	ii
Table of contents	iii
List of tables	vi
List of figures	vii
1 Introduction	1
1.1 Problem Definition	1
1.2 Financial Fraud	1
1.2.1 Fraudsters Categories	1
1.2.2 Types of Fraud	2
1.3 Fraud Detection Methods	3
1.3.1 Public Data Sources	4
1.4 Recurrent Neural Networks	6
1.4.1 Advantages and Disadvantages	6
1.4.2 Types of Recurrent Neural Networks	7
2 Previous Work	8

2.1	Credit Card Transactions, Fraud Detection, and Machine Learning: Modelling Time with LSTM Recurrent Neural Networks (2007)	8
2.2	Long Short Term Memory Networks for Anomaly Detection in Time Series (2015)	9
2.3	Detecting Fraudulent Behavior Using Recurrent Neural Networks (2016)	9
2.4	Transaction Fraud Detection Using GRU-centered Sandwich-structured Model (2017)	10
3	Methodology	11
3.1	Data Analysis	11
3.1.1	Description	11
3.1.2	Agents	12
3.1.3	Features	13
3.2	Insights	15
3.3	Data Transformation	19
3.3.1	Categorical To Numerical	19
3.3.2	2D to 3D Dataset	19
3.3.3	New Feature Generation	20
3.3.4	Normalization	22
3.3.5	Oversampling with SMOTE	22
3.3.6	Class Weights	23
3.4	Model Architecture	23
3.4.1	Recurrent Neural Layers	23
3.4.2	Fully Connected Layers	24
3.4.3	Callbacks	25
3.5	Model Training and Selection	25
3.5.1	Adam Optimizer	26

3.5.2	Binary Cross Entropy Loss Function	26
3.5.3	Stratified K-Fold Cross Validation	26
3.5.4	Grid Search	26
3.5.5	Models	28
3.6	Model Validation	30
3.6.1	Baseline Models	30
3.7	Creating More Data Using PaySim	33
4	Results and Discussion	35
5	Conclusions	37
	Bibliography	38

List of tables

3.1	Age Ranges	14
3.2	Gender	14
3.3	Categories [25]	15
3.4	Fraud Percent per Category	16
3.5	Fraud Percentage and Values per Amount Range	17
3.6	Gender	25
3.7	Gender	27
3.8	Top 7 GridSearchCV Models	28
3.9	Models. Part 1	29
3.10	Models. Part 2	30

List of figures

3.1	BankSim Use Case Diagram including misuse cases [24]	13
3.2	Amount Spent per Category	16
3.3	Fraud vs Non-Fraud Amount and Occurrence	17
3.4	Feature Importance	18
3.5	Number of Transactions per Customer	20
3.6	Feature Importance Ranking Including New Features	21
3.7	Batch Division	22
3.8	SimpleRNN vs GRU vs LSTM [10]	24
3.9	GridSearchCV Best Model Performance on the Validation Data	27
3.10	PR Curve on BankSim Dataset	31
3.11	ROC Curve on BankSim Dataset	32
3.12	PaySim Models PR Curve	33
3.13	PaySim Models ROC Curve	34

Chapter 1

Introduction

1.1 Problem Definition

In the digital age or fourth industrial revolution, fraud has become a significant problem for financial institutions. Financial institutions create new fraud detection systems to protect their clients, but most of these solutions depend on inflexible rule-based systems, which are quickly left obsolete since fraudsters keep finding new ways to exploit the vulnerabilities of such.

The focus of this study is the identification of credit card fraud transactions. 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. [24]. This dataset has time-ordered transactions, which is a perfect environment to test Recurrent Neural Networks. [31]

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. [32]
2. **Second Party** - All fraud actions made inside the financial information ~~is~~ ~~consider~~ as second party fraud. For example, a bank employee creating false accounts and moving big amounts of money between ~~accounts~~. [30]
3. **Third Party** - When a person not related to a financial institution or its users is involved in the ~~deft~~ of sensitive information to act using a false identity to make fraud transactions, it is ~~consider~~ as third party frauds. For example, when card information is stolen, or cloned and used to ~~do~~ withdrawals ~~money~~ or do online transactions. [30]

1.2.2 Types of Fraud

The following are type of frauds and ~~methods~~ to identify them. [20]

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 overestimated ~~value~~ regarding a mortgage to access new credit. This prevents the bank ~~of~~ 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 ~~left~~.
7. **ACH Fraud** - ACH and wire transfer methods transfer funds from one account to another but wire transfers ~~sent 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 ~~left~~. Canadian Payments Association(CPA) is the ACH alternative in Canada.

1.3 Fraud Detection Methods

The following are anomaly detection techniques and their categories. [28]

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 ~~tries~~ 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 Sources

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. [29] 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. [27] [23] 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 ~~is richer in~~ features. [24] It is also based in MABS but uses a sample of credit card transactions data between November 2011~~12~~ 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 ~~this~~ attributes, this dataset is used ~~during the~~ study to test the proposed algorithms.

The goal of BankSim is ~~the same as its predecessor~~, 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 ~~category~~, customer gender and age. The data was generated using population statistical information from the real sample. The sample contains averages, ~~minim~~ 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. [7] The BankSim generated dataset available in Kaggle was generated by running the simulator for 180 steps (approx. six months), producing ~~594643 record total~~. Where 7200 are fraudulent transactions, and 587443 are normal ~~payments~~.

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. 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] [15] [21] [22]

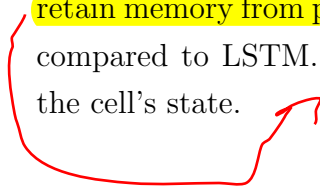
1.4 Recurrent Neural Networks

Recurrent Neural Networks are similar to traditional neural networks but have the ability to selectively send information between time steps and this way remember previous seen patterns. This is something traditional models are not able to do. Due to this feature, this algorithms 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. [34]

1.4.1 Advantages and Disadvantages

Recurrent neural networks have traditionally been difficult to train, often contain millions of parameters, and have a vanishing and exploding gradients problems. Exploding gradients are the accumulation of large values updates to the neural network model weights during training. Similar to exploding gradients, vanishing gradients are the opposite. This means, when small values less than one during the multiple back-propagation matrix multiplication can cause the weights to shrink exponentially. Advances in network architectures, optimization, and techniques have address most of this problems. The main advantage of recurrent neural networks over traditional models is that RNN 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. They solve 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)** - Both LSTM and GRU have the ability to retain memory from previous time-steps. The GRU cell is simpler in architecture compared to LSTM. GRU cells have an update and forget gates that control the cell's state.
- 

Chapter 2

Previous Work



~~2.1 Credit Card Transactions, Fraud Detection, and Machine Learning: Modelling Time with LSTM Recurrent Neural Networks (2007)~~

~~During the research, three algorithms were tested~~ 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 ~~paper is to test if the LSTM can outperform~~ the older techniques in fraud detection. [8]

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 ~~useful~~ 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 ~~compared~~ FFNN vs SVM using ~~the~~ dataset

~~contains~~ 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.

~~2.2 Long Short Term Memory Networks for Anomaly Detection in Time Series (2015)~~

~~During the study, a model made of multiple stacked recurrent hidden layers was~~ trained on four datasets to determine the efficacy of LSTM for anomaly detection. The datasets used were an electrocardiograms, 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. The LSTM model and a traditional RNN model were able to perform well in both the ECG and the engine datasets which had short-term temporal dependence but **they performed poorly** in the other datasets which had more long-term dependency. [9]

~~2.3 Detecting Fraudulent Behavior Using Recurrent Neural Networks (2016)~~


During this study, the authors 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 stole 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 ~~were better in performance compared to SVM but have equivalent results with LSTM.~~

The only difference between RNN and LSTM results was that LSTM converged to a local minima faster ~~than the previous one~~. [33]

~~2.4 Transaction Fraud Detection Using GRU-centered Sandwich-structured Model (2017)~~

The purpose of this research was to test 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 ~~consist~~ 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. [2]

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 amount of matrix multiplication over a very large dataset, which also reduced the time of convergence. The results showed that Random Forest (RF) and Gradient Boosting (GBDT) ~~models had better performance over SVM and Linear Regression models, but GRU out-performed the ensemble models.~~ With regards to the model combinations, GBDT-GRU-RF (WBW) had a better performance  than the GBDT-RF-GRU (WWB), the GRUGBDT+RF (BWW) models, and the simpler models.

Chapter 3

Methodology

3.1 Data Analysis

3.1.1 ~~Description~~

Financial fraud labeled datasets 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. ~~Therefore, it stops the possibility of mutual benefit and information exchange. In contrast, fraudsters adapt quicker and find vulnerabilities in their systems.~~

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" [24] and "PaySim: a financial mobile money simulator for fraud detection." **[?] 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.



The same authors developed both BankSim and PaySim, both programs share similar output format, but BankSim output is more abundant in features, and it is based on data from an anonymous bank in Spain. PaySim's output data is based on an African mobile money service. **A vital fact to note is that both papers describe the simulator development but do not address the generation and injection of malicious fraud behavior. They only describe possible scenarios that could be done by tweaking the simulator's parameters.** Both simulators require input parameters that define the output. Examples are sales probability distributions for each merchant, product categories, geographic locations, and demographic consumer behavior based on age and gender. Some parameters, such as age and gender, are based on averages, minimum, and maximum values instead of a standard deviation.

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.

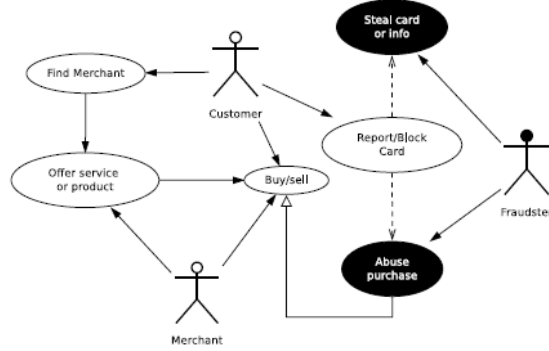
3.1.2 Agents

The BankSim simulation is based on three agents: **Merchant, Customer, and Fraudster.**

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: BankSim Use Case Diagram including misuse cases [24]



3.1.3 Features

The BankSim dataset contains nine features and a target fraud/non-fraud column. The features columns are the following. [25]

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:** The ages are divided in the following 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:** The customers' genders.

ID	Gender
E	Enterprise
F	Female
M	Male
U	Unknown

Table 3.2: Gender

8. **Category:** The purchase categories are the following:

category	percentage	avg	std
Auto	0.0049	224.35916667	267.52611111
Bars and restaurants	0.0244	31.03238095	39.19238095
Books and press	0.0014	33.34714286	45.01428571
Fashion	0.0076	49.73190476	59.08452381
Food	0.0726	32.49333333	30.87285714
Health	0.0179	59.39119048	113.98619048
Home	0.0021	75.48317073	121.77292683
Accommodation	0.0016	97.41071429	86.85047619
Hypermarkets	0.0178	33.06547619	35.78166667
Leisure	0.0001	74.86357143	22.01107143
Other services	0.009	52.9897619	76.65309524
Sports and toys	0.0043	74.8047619	75.45452381
Technology	0.0031	67.28285714	108.68452381
Transport	0.8176	24.56047619	20.76928571
Travel	0.0004	577.46285714	518.41885714
Wellness and beauty	0.0155	44.20809524	55.29142857

Table 3.3: Categories [25]

9. **Amount:** Purchase amount.

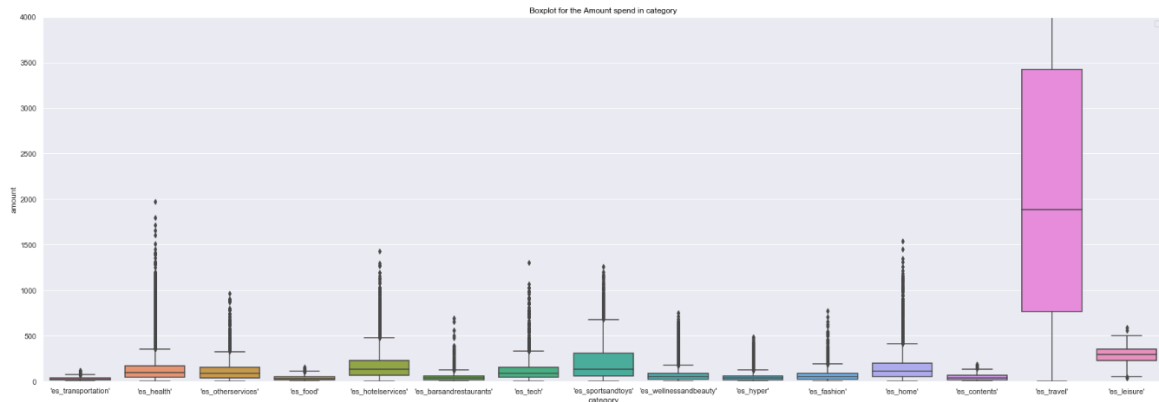
10. **Fraud:** Target variable which shows if the transaction is fraudulent (1) or benign (0).

3.2 Insights

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

1. "Leisure", "travel", "Sports and Toys" are the most selected categories for fraudsters.

Figure 3.2: Amount Spent per Category



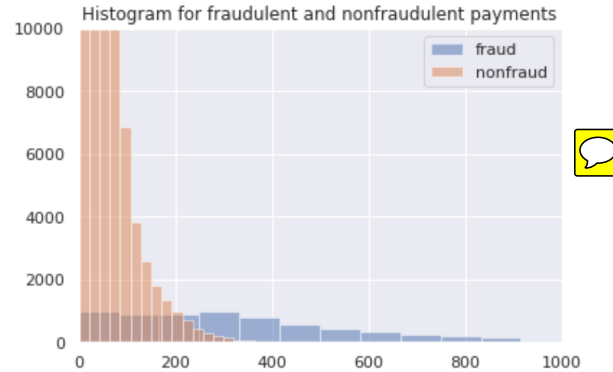
2. Fraudster agents prefer the categories ~~which~~ people spend more.

Category	Fraud Percent(%)
'es_transportation'	0.0
'es_food'	0.0
'es_hyper'	4.6
'es_barsandrestaurants'	1.9
'es_contents'	0.0
'es_wellnessandbeauty'	4.8
'es_fashion'	1.8
'es_leisure'	95.0
'es_otherservices'	25.0
'es_sportsandtoys'	49.5
'es_tech'	6.7
'es_health'	10.5
'es_hotelservices'	31.4
'es_home'	15.2
'es_travel'	79.4

Table 3.4: Fraud Percent per Category

3. The fraudulent transactions ~~are less in count but more in amount~~ as shown in the histogram below.

Figure 3.3: Fraud vs Non-Fraud Amount and Occurrence

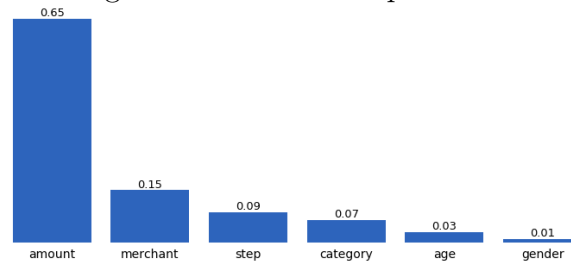


Transaction Amount[\$]	Total	Fraud	Safe	Fraud (%)	Safe (%)
\$ 0 555	592793.0	5430.0	587363.0	0.92	99.08
\$ 555 1110	1309.0	1257.0	52.0	96.03	3.97
\$ 1110 1665	143.0	121.0	22.0	84.62	15.38
\$ 1665 2221	81.0	75.0	6.0	92.59	7.41
\$ 2221 2776	65.0	65.0	0.0	100.00	0.00
\$ 2776 3331	59.0	59.0	0.0	100.00	0.00
\$ 3331 3887	48.0	48.0	0.0	100.00	0.00
\$ 3887 4442	48.0	48.0	0.0	100.00	0.00
\$ 4442 4997	38.0	38.0	0.0	100.00	0.00
\$ 4997 5553	21.0	21.0	0.0	100.00	0.00
\$ 5553 6108	14.0	14.0	0.0	100.00	0.00
\$ 6108 6663	15.0	15.0	0.0	100.00	0.00
\$ 6663 7219	3.0	3.0	0.0	100.00	0.00
\$ 7219 7774	5.0	5.0	0.0	100.00	0.00
\$ 7774 8329	1.0	1.0	0.0	100.00	0.00

Table 3.5: Fraud Percentage and Values per Amount Range

- Fraud occurs more in ages equal and below 18 (0th category).
- The complete dataset contains 98.79% of normal transactions and 1.21% fraudulent transactions.
- Based on the random forest feature importance ranking, the feature 'transaction amount' is the most influential feature in detecting (or classifying) a fraud.

Figure 3.4: Feature Importance



7. The BankSim [25] and PaySim [27] generated datasets are very similar in format, but BankSim's data output ~~is richer in~~ features. For example, it has demographic information such as age and gender.
8. 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.
9. 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".
10. 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 [23] is not from the real data but a synthetic generated approximation, since he did not want to disclose any real data values.
11. 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.
12. The customer and merchant agents in the simulation operate randomly but they try to approximate their spending frequency to the target population statistics.

13. 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 have two options, creating fix sized batches with the maximum number of



transactions and fill with padding 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, both have nine features. We decided to utilize batches with a size of 265, and even thou it will have more padding, we avoided deleting data points. The final data shape is (4112, 265, 9)

Figure 3.5: Number of Transactions per Customer



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

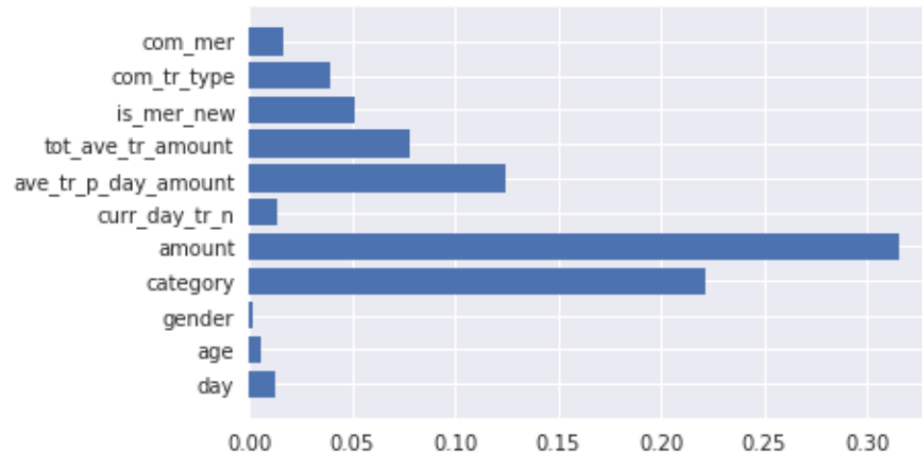
The most complex and time consuming part of the data pre-processing pipeline is to generate new features from the exiting data. The data generated is calculated inside

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.

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?

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.

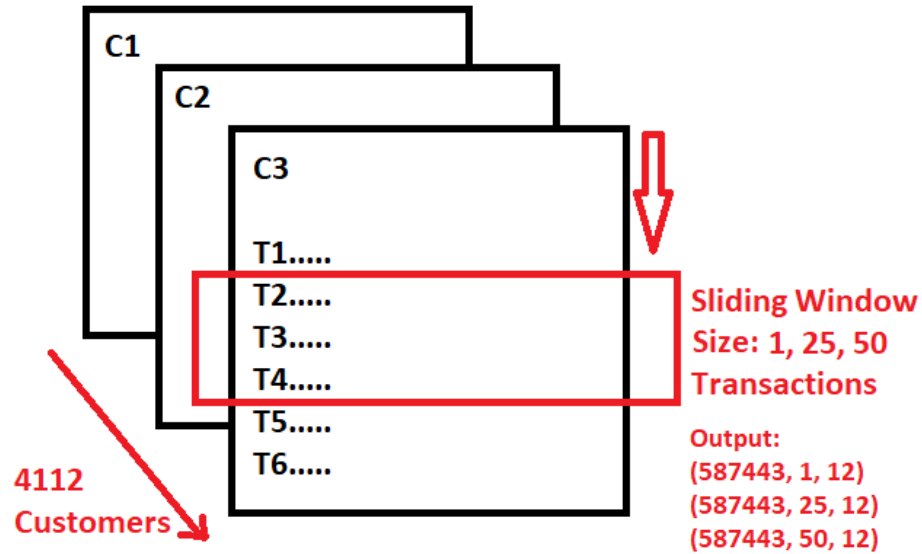
Figure 3.6: Feature Importance Ranking Including New Features



After generating the new values for the current transaction, it takes 24 or 49 transactions before the current transaction in which the sliding window index is currently located and generate batches with a smaller size. 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 batches of size 1, 25 and 50. 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 shape including the new features are (587443, 1, 12), (587443, 25, 12) or (587443, 50, 12).

Figure 3.7: Batch Division



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. [5]

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 tampered with 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.

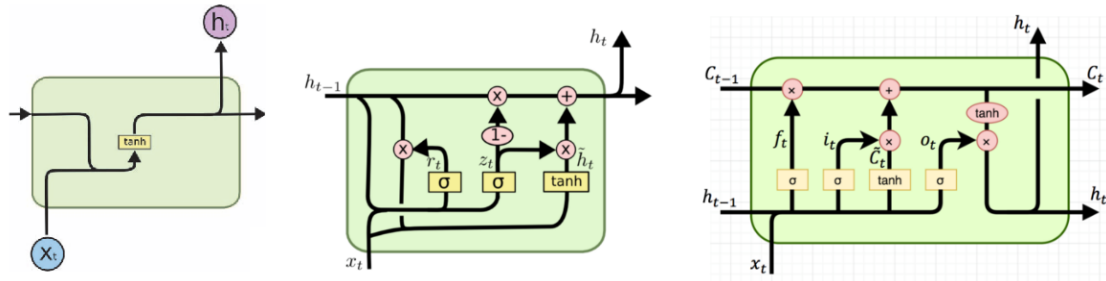
3.4.1 Recurrent Neural Layers

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. The recurrent layers used in this study are described below.

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

Figure 3.8: SimpleRNN vs GRU vs LSTM [10]



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. [4]

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. [6]

3.4.3 Callbacks

Callbacks are custom methods called during training, evaluation, or inference. The callbacks used in this study are described below.

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: Gender

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 [1], 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 ~~number of sections~~ where one of them is used for testing. The process is repeated k number of times until all ~~folds~~ are used for testing. Stratification means that each fold will contain the same percentage of each class of transactions. K-Fold is used together with grid-search to select the best performing model. [12]

3.5.4 Grid Search

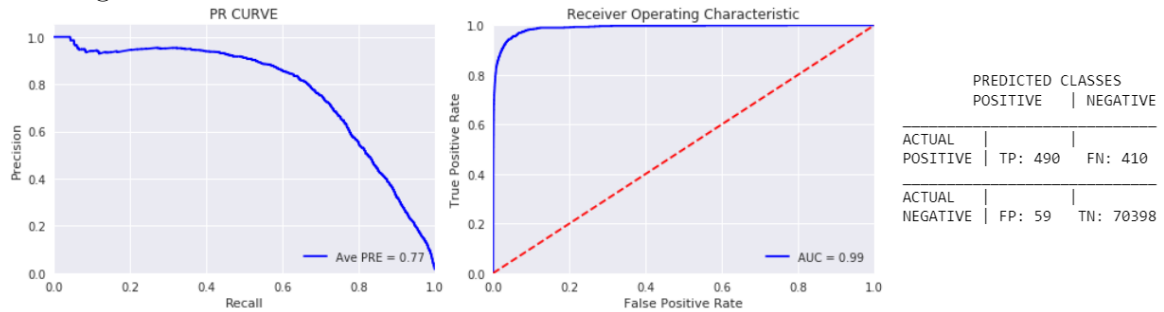
GridSearchCV is a Sklearn method that considers all parameter ~~combination~~. It generates candidates from a grid of parameters and selects the best performing during training. [11]. The KerasClassifier wrapper is used to be able to combine Keras model inside the Sklearn grid search. [13] The following combination were trained and tested during the 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: Gender

The best model 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.

Figure 3.9: GridSearchCV Best Model Performance on the Validation Data



The table below show the top 7 models including the ones used to test against the baseline validation models. The best model has the id 12 and the tag "BEST_MODEL(GRU)". Model 5, 13, 1, 4, and 9 have the identification tags "2BEST_RNN(GRU)", "GRU_1", "LSTM_1", "GRU_2", and "LSTM_2", respectively.

ID	Hyper-parameters	Train Acc(%)	Train Pre(%)	Train Rec(%)	Train ROC(%)	Test Acc(%)	Test Prec(%)	Test Rec(%)	Test ROC(%)
12	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
5	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
13	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
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
4	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
9	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

The following table includes all the models tested during this study and their performance results. The changes made to ~~model is mention~~ in the description column.

Model Identifier Tag	Description	PR-AUC	ROC-AUC
BEST_RNN(GRU)	The best and the 12th model obtained during grid-search.	0.79	0.99
2BEST_RNN(GRU)	The second best and the 5th model obtained during grid-search.	0.59	0.98
LSTM_1	The fourth best and the 1st model obtained during grid-search.	0.54	0.98
LSTM_2	The seventh best and the 9th model obtained during grid-search.	0.55	0.98
GRU_1	The third best and the 13th model obtained during grid-search.	0.59	0.98
GRU_2	The sixth best and the 4th model obtained during grid-search.	0.57	0.98
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 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. The reason for this change was to test if there could be an improvement by increasing the training set.	0.81	0.99
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
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_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_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
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_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
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 and using an increased dataset.	0.83	0.99

Table 3.9: Models. Part 1

Model Identifier Tag	Description	PR-AUC	ROC-AUC
DD_1B_OVERSAMPLED_RNN (GRU)	This model is has the same hyper-parameters as the best grid-search models but was trained on over-sampled data using SMOTE.	0.84	0.99
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

Table 3.10: Models. Part 2

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. [14]

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. [3]

Figure 3.10: PR Curve on BankSim Dataset

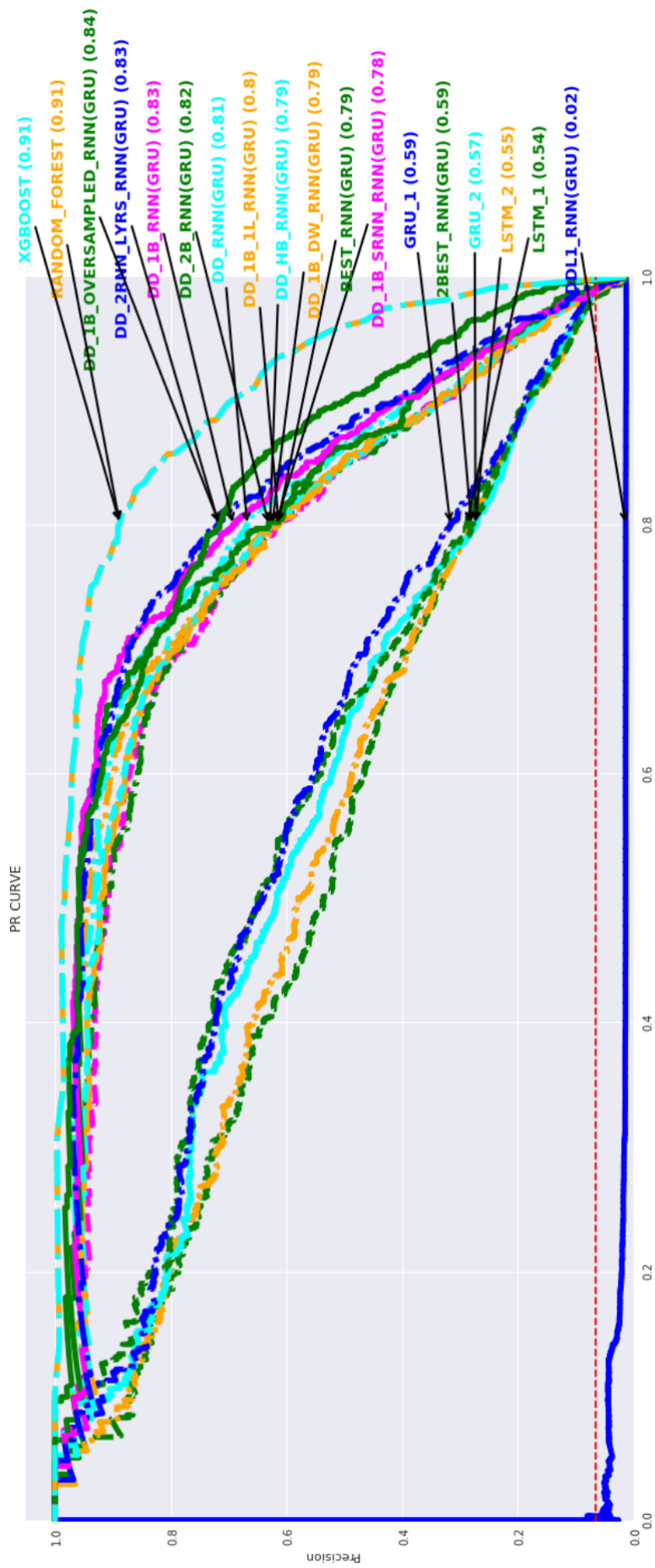
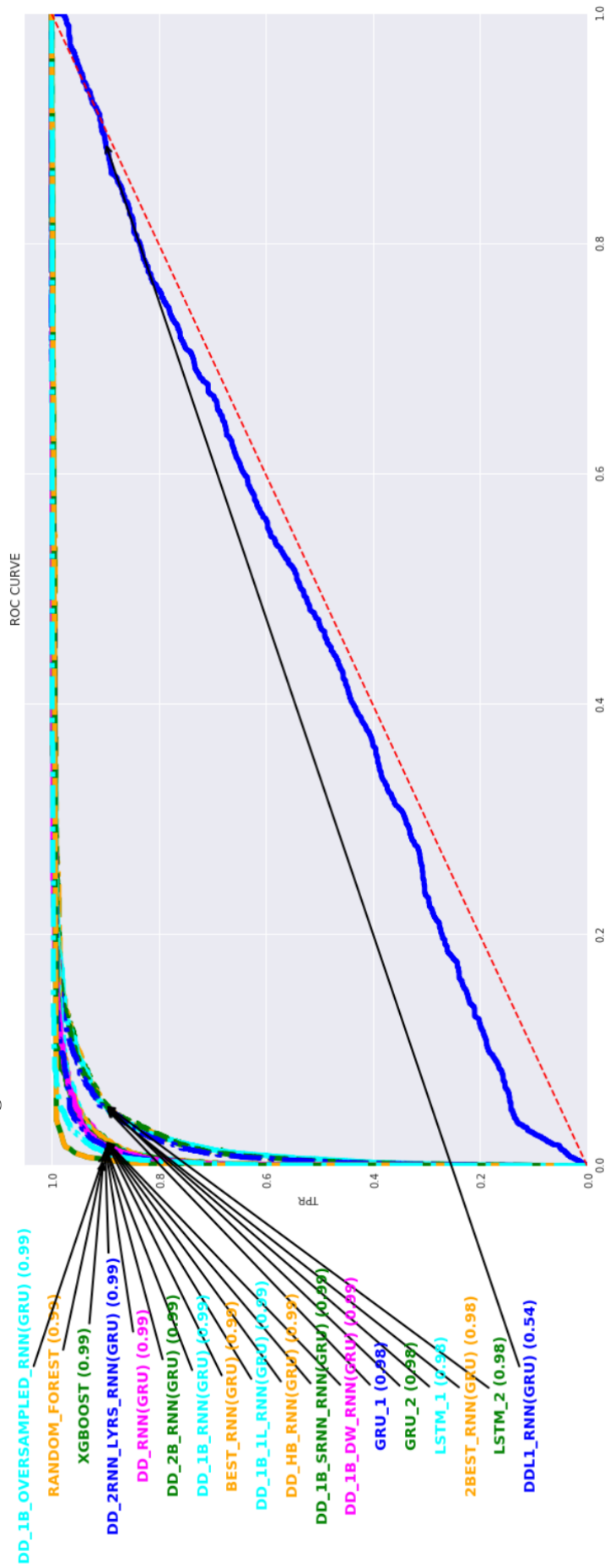


Figure 3.11: ROC Curve on BankSim Dataset



3.7 Creating More Data Using PaySim

Our last test was with PaySim. Paysim’s source code located publicly in GitHub [23] 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. These 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. 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 constraints, 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 proved 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 decreased the data size and only left one transaction per batch (10510453, 1, 12). The model’s performance improved significantly.

Figure 3.12: PaySim Models PR Curve

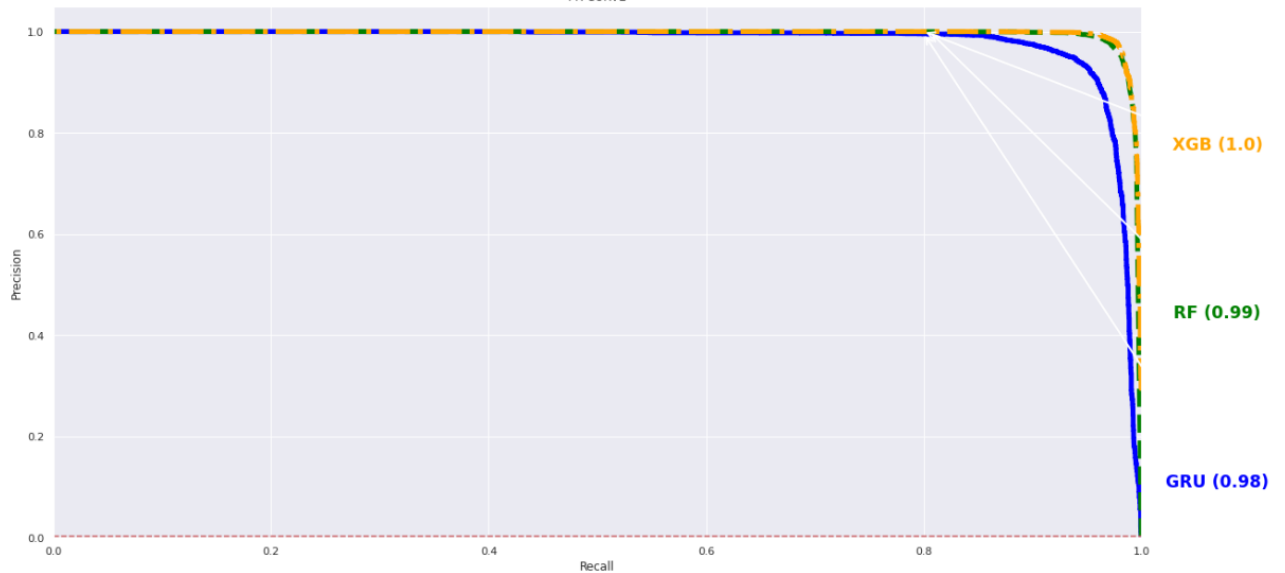
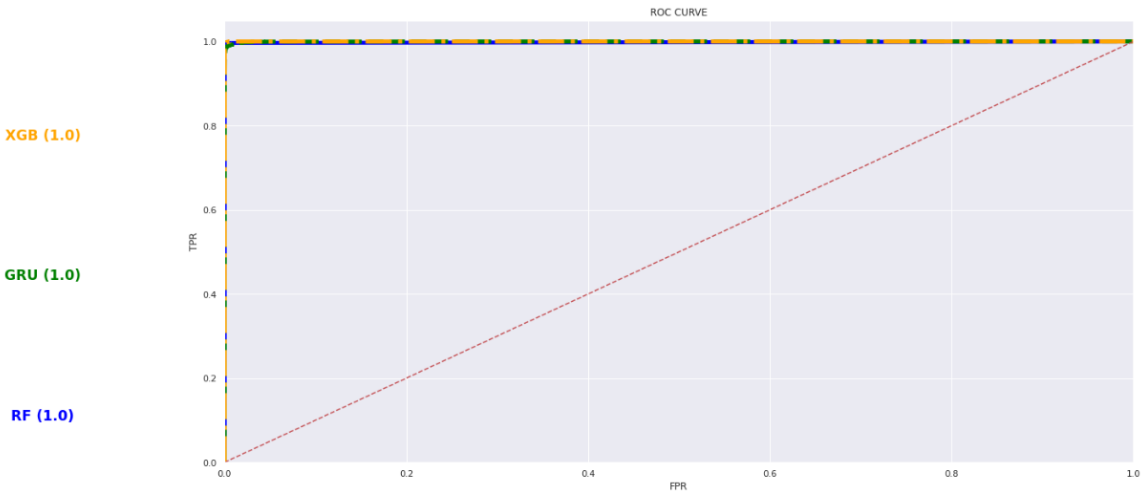


Figure 3.13: PaySim Models ROC Curve



Chapter 4

Results and Discussion



1. The models with GRU layers performed better than the models with LSTMs during training using the BankSim dataset.

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.

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. Contradicting the previous statement, increasing the batch size also improved the model's performance, but the performance was still below the ones that used smaller batches. This can be seen in the "DD_2B_RNN(GRU)" model's Precision-Recall curve.

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.



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 thou 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] [1412.6980] adam: A method for stochastic optimization. <https://arxiv.org/abs/1412.6980>. (Accessed on 05/23/2020).
- [2] [1711.01434] transaction fraud detection using gru-centered sandwich-structured model. <https://arxiv.org/abs/1711.01434>. (Accessed on 05/23/2020).
- [3] 3.2.4.3.1. sklearn.ensemble.randomforestclassifier — scikit-learn 0.23.1 documentation. <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>. (Accessed on 05/23/2020).
- [4] Dropout layer. https://keras.io/api/layers/regularization_layers/dropout/. (Accessed on 05/23/2020).
- [5] imblearn.over_sampling.smote — imbalanced-learn 0.5.0 documentation. https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.over_sampling.SMOTE.html.
- [6] Layer weight regularizers. <https://keras.io/api/layers/regularizers/>. (Accessed on 05/23/2020).
- [7] Mason multiagent simulation toolkit.
-  [8] (pdf) credit card transactions, fraud detection, and machine learning: Modelling time with lstm recurrent neural networks. https://www.researchgate.net/publication/226869920_Credit_Card_Transactions_Fraud_Detection_and_Machine_Learning:_Modelling_time_with_lstm_recurrent_neural_networks.
-  [9] (pdf) long short term memory networks for anomaly detection in time series. https://www.researchgate.net/publication/304782562_Long_Short_Term_Memory_Networks_for_Anomaly_Detection_in_Time_Series.
- [10] 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).
- [11] sklearn.model_selection.gridsearchcv — scikit-learn 0.23.1 documentation. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html.

- [12] sklearn.model_selection.stratifiedkfold — scikit-learn 0.23.1 documentation. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html. (Accessed on 05/23/2020).
- [13] tf.keras.wrappers.scikit_learn.kerasclassifier. https://www.tensorflow.org/api_docs/python/tf/keras/wrappers/scikit_learn/KerasClassifier.
- [14] Xgboost documentation — xgboost 1.1.0-snapshot documentation. <https://xgboost.readthedocs.io/en/latest/>. (Accessed on 05/23/2020).
- [15] 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, 09 2017.
- [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, 09 2017.
- [17] A. Dal Pozzolo, O. Caelen, R. Johnson, and G. Bontempi. Calibrating probability with undersampling for unbalanced classification. 12 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. 2013.
- [20] O. A. Djeflal Abdelhamid, Soltani Khaoula. Automatic bank fraud detection using support vector machines. 2014.
- [21] 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.
- [22] 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.
- [23] E. Lopez. Edgarlopezphd/paysim, 2015.
- [24] 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.
- [25] E. A. Lopez-Rojas and S. Axelsson. Synthetic datasets generated by the banksim payments simulator. 2014.

- [26] E. A. Lopez-Rojas, A. Elmir, and S. Axelsson. Paysim: A financial mobile money simulator for fraud detection. 09 2016.
- [27] E. A. Lopez-Rojas, A. Elmir, and S. Axelsson. Synthetic financial datasets for fraud detection. 2016.
- [28] S. S. H. Lutfun Nahar Lata, Israt Amir Koushika. A comprehensive survey of fraud detection techniques. *International Journal of Applied Information Systems (IJAIS)* –, 12 2015.
- [29] S. Moss. Multi-agent-based simulation - second international workshop, mabs 2000, boston, ma, usa, july 2000; revised and additional papers: Scott moss, 2001.
- [30] A. Ptak-Chmielewska and A. Matuszyk. Profile of the fraudulent customer. *Bezpieczny Bank*, 2(59):7–23, 01 2015.
- [31] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. , 323(6088):533–536, Oct. 1986.
- [32] 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.
- [33] H. T. Yoshihiro Ando, Hidehito Gomi. Detecting fraudulent behavior using recurrent neural networks. <https://pdfs.semanticscholar.org/5d31/8371549831a05825de6e4eae4461538aeb9b.pdf>. (Accessed on 05/23/2020).
- [34] J. B. Zachary C. Lipton. A critical review of recurrent neural networks for sequence learning, June 2015.