CuRL: Coupled Representation Learning of cards and merchants to detect transaction frauds

Maitrey Gramopadhye*, Shreyansh Singh*, Kushagra Agarwal, Nitish Srivasatava, Alok Mani Singh, Siddhartha Asthana, and Ankur Arora

AI Garage, Mastercard, India

Abstract. Payment networks like Mastercard or Visa process billions of transactions every year. A significant number of these transactions are fraudulent that cause huge losses to financial institutions. Conventional fraud detection methods fail to capture higher-order interactions between payment entities i.e., cards and merchants, which could be crucial to detect out-of-pattern, possibly fraudulent transactions. Several works have focused on capturing these interactions by representing the transaction data either as a bipartite graph or homogeneous graph projections of the payment entities. In a homogeneous graph, higher-order cross-interactions between the entities are lost and hence the representations learned are sub-optimal. In a bipartite graph, the sequences generated through random walk are stochastic, computationally expensive to generate, and sometimes drift away to include uncorrelated nodes. Moreover, scaling graph-learning algorithms and using them for real-time fraud scoring is an open challenge. In this paper, we propose `CuRL` and `tCuRL`, coupled representation learning methods that can effectively capture the higher-order interactions in a bipartite graph of payment entities. Instead of relying on random walks, proposed methods generate coupled session-based interaction pairs of entities which are then fed as input to the skip-gram model to learn entity representations. The model learns the representations for both entities simultaneously and in the same embedding space, which helps to capture their cross-interactions effectively. Furthermore, considering the session constrained neighborhood structure of an entity makes the pair generation process efficient. This paper demonstrates that the proposed methods run faster than many state-of-the-art representation learning algorithms and produce embeddings that outperform other relevant baselines on fraud classification task.

Keywords: Fraud detection · Payment network · Representation Learning · Skip-gram

1 Introduction

Detecting fraudulent transactions has been an ever-present problem in the payment industry with an average number of fraudulent transactions attempted per

* Both authors have equal contribution

merchant increasing at a rate of 34% per annum [25] and total worldwide fraud losses projected to reach $35.67 billion in 2023 [28].

Conventional fraud detection solutions have moved from manually coded rules to either traditional classification models or anomaly detection-based models [5], [22], [26]. The use of deep learning models has further improved the performance as they better capture the complex interactions between transaction features. Given the sequential nature of transactions, LSTM and other seq2seq models have previously been used for fraud detection. However, most of these methods [9], [1] treat each transaction independently and fail to capture any interaction information between the payment entities, which can be useful in detecting any out of pattern transaction.

Recent works have focused on capturing the interaction between payment entities by leveraging graph-based methods for learning card and merchant representations which serve as useful features for downstream supervised tasks. However, methods that consider the transaction network as a graph of homogeneous entities [18], [12], [24], [27], capture merchant-merchant relation or card-card relation but miss out on cross-entity interaction information. Whereas, most methods that consider the heterogeneous nature of the transaction graph rely on generating sequences through random walks [7], [4] which take into account the connections of the entities but without the temporal information. Furthermore, these random walks are stochastic and sometimes drift away to include uncorrelated nodes thus generating sub-optimal representations. Moreover, these graph-learning algorithms are computationally expensive, and scaling them for real-time fraud detection is a big challenge.

In this paper, we propose `CuRL`, a scalable method of learning card and merchant representations from a bipartite transaction graph. We explain how our method effectively captures the cross-interactions between cards and merchants along with their homogeneous interactions. Capturing these interactions allows our model to retain more information in the entity embeddings, which in turn function as richer features for downstream fraud detection. We adapt word2vec's skip-gram [19] as our embedding generation model and sample the neighborhood of a node in the transaction graph to form coupled pairs (Card-Card, Merchant-Merchant, Card-Merchant) to train the skip-gram. Additionally, as new transactions come in, the model has to be periodically retrained. To facilitate this, our model focuses on being lightweight and samples only the immediate neighborhood of a node. As an extension to `CuRL`, we also propose `tCuRL`, a session-based sampling method that further reduces the size of the neighborhood sampled for each node by removing uncorrelated neighbors. Thus, further reducing the training time for the embedding generation model and improving the quality of the representations.

We empirically compare our methods with other embedding generation methods and show that embeddings generated by `CuRL` and `tCuRL`, when included as features for fraud detection, beat all baseline methods. Moreover, since most of the graph-based models are computationally expensive on large graphs, we also introduce the embedding generation time as a metric while benchmarking.

The organization of the paper is as follows. In Section 2, we cover the related work done in the field of fraud detection and embedding generation, particularly in the payment industry domain. Section 3 talks about the synthetic data generation and data preparation process. Section 4 describes the proposed approach for card and merchant embedding generation. Section 5 has the details for the conducted experiments and section 6 shows the comparison of our method against the baselines. Section 7 concludes our work.

2 Related Work

Fraud detection has been an active area of research for a long time. We present here an overview of the fraud detection techniques and methods that have been applied in detecting different financial frauds. Bayes method [2], SVM [30], decision trees and neural networks [26] have been applied extensively for solving fraud detection problems. The general problem with above-mentioned models is that they usually rely on hand-crafted features and manual feature engineering to capture transaction information. Deep learning methods like [20], [31] use autoencoders and denoising autoencoders to transform transaction features to a lower dimension to classify the transaction as fraud. [16] tried to solve the fraud detection problem as a transaction sequence classification technique and hence used an LSTM model, but it failed to also capture the payment entity interactions.

To capture these interactions, several works have considered the transaction network as a bipartite graph with cards and merchants as nodes. [14] was a graph-based method for detecting fraudulent behavior in social media networks. [15], [8] model fraud detection as graph-based anomaly detection. Other methods involve creating node embeddings using the network structure. Earlier approaches for node embedding generation involved using hand-crafted features based on network properties [10], [13]. [29], [21] propose methods to analyze the transaction graph and extract network features manually to use along with intrinsic transaction features relevant for fraud detection task.

Recent advancements in representational learning for natural language processing opened new ways of feature learning for discrete objects such as words. In particular, the Skip-gram model [19] aims to learn continuous feature representations for words by optimizing a neighborhood preserving likelihood objective. Inspired by the Skip-gram model, recent research established an analogy for networks by representing a network as a "document" [24], [27]. The same way as a document is an ordered sequence of words, one could sample sequences of nodes from the underlying network and turn a network into an ordered sequence of nodes. However, there are many possible sampling strategies for nodes, resulting in different learned feature representations. A popular theme we noticed in earlier works was that of using random walks to sample the neighbors of a node. node2vec [12] performs a biased random walk to obtain the neighborhood while DeepWalk [24] deploys a truncated random walk for social network embedding, and metapath2vec [7] uses meta-path-based random walks. Unlike

Deepwalk, metapath2vec defines metapaths along which we want the walker to move. Pin2Vec [18], originally created for bipartite graphs, forms homogeneous graphs for each entity and trains separate skip-gram models for them. DeepTrax [3] learns embeddings for cards and merchants from time-constrained random walks taken on the transaction graph. HitFraud [4] forms a heterogeneous information network from the transactions and analyzes it to form graph-based features by generating meta-paths on the graph. A limitation of such models is that they often tend to drift away to include uncorrelated nodes while creating the sequences. In practice, we also noticed that sampling a neighborhood using random walks is a computationally expensive process. These observations are evident in our results section (Section 6) as well. BigGraph [17] is another algorithm for learning node embeddings for large graphs, with up to billions of nodes and trillions of edges. It learns node embeddings through knowledge graph implementation. BiNE, short for Bipartite Network Embedding [11], generates sequences that preserve the long-tail distribution of nodes in a bipartite graph. It however uses a biased random walk generator to generate node neighborhood for subsequent sequence generation.

3 Dataset

3.1 Description of Data

Pursuant to internal controls to protect data, confidentiality, and privacy, we don't use real transaction data for this paper. As this is an exploratory research, we have used synthetically generated transaction data for our experiments, created by applying SMOTE-NC [6] on real transaction data. Data created is synthetic and cannot be traced back to any original transaction. Synthetic data was generated for binary classes - fraud and non-fraud. It consists of 537k transactions, of which $1,100$ transactions are fraudulent (0.2% of transactions). There are $125,019$ unique cards and 220 unique merchants in the dataset. Our dataset mimics the skewed distribution of cards found in transaction data. Several cards have just 1-2 transactions whereas a few cards have many transactions.

From the dataset, we only use the card number and merchant ID to generate embeddings for cards and merchants. Note that the card numbers and merchant IDs used are from the synthetic dataset and not the real data. The same dataset is used for all the experiments and the embeddings generated are then used to train the model for the downstream task of fraud detection.

3.2 SMOTE-NC

SMOTE-NC (Synthetic Minority Oversampling Technique) [6] is a data augmentation technique for generating synthetic data. It is a variant of SMOTE and is used when the data to be generated has continuous as well as non-continuous features. Since we need to generate synthetic data for both the classes, we train SMOTE-NC on two different datasets, one for each class.

Statistic	txn_amt	30d_avg	14d_avg
count	538,125	538,125	538,125
mean	46.31	0.86	0.82
std	150.45	0.27	0.34
min	0.01	0.00	0.00
25%	8.97	0.88	0.83
50%	16.98	1.00	1.00
75%	43.96	1.00	1.00
max	44,418.73	1.00	1.00

Table 1: Distribution of transaction amount in real data with 30-day and 14-day average

Statistic	txn_amt	30d_avg	14d_avg
count	538,125	538,125	538,125
mean	46.28	0.87	0.83
std	144.30	0.24	0.30
min	0.01	0.00	0.00
25%	9.08	0.87	0.82
50%	17.02	0.98	0.99
75%	43.95	1.00	1.00
max	36,661.17	1.00	1.00

Table 2: Distribution of transaction amount in synthetic data with 30-day and 14-day average

3.3 Distribution Similarity

Distribution similarity between the synthetic data and real data will ensure that a model performing well on synthetic data, will perform well on the real data as well. Due to privacy concerns, we cannot compare the two datasets at the individual data point level but in Tables 1 and 2 we show the comparisons on an aggregate level. Furthermore, we also use the Evaluation Framework of SDV [23] to calculate the quality of the synthetic data. The Chi-squared test on the two datasets gives a value of 0.998 and the Kolmogorov–Smirnov test gives a value of 0.885. These metrics show that the two datasets are very similar.

4 Methodology

4.1 CuRL

The `CuRL` framework, visually explained in the system architecture Figure 1, generates Card and Merchant embedding by a novel strategy of inducing Card-Card, Merchant-Merchant, and Merchant-Card pairs from bipartite transaction graph. These pairs of C-M, M-M and C-C are then fed to word2vec based skip-gram model to learn entities representations in the same embedding space.

To create M-M, C-M and C-C pairs, joint neighborhood (N_i) is calculated collectively for card (C_i) and Merchant (M_i), for each transaction (T_i). As shown in equation 2, N_i is defined as the union of the individual neighborhoods of C_i (N_i^C) and M_i (N_i^M). From equation 1, N_i^C consists of all merchants that C_i transacted with, in an analysis window w, immediately before and after T_i, i.e. w merchants the C_i transacted with each before and after T_i. Similarly, N_i^M consists of the cards which transacted with M_i immediately before and after T_i, within the analysis window. Also, relationship between C_i and M_i is also included in N_i. After calculating the joint neighborhood N_i, all nodes in N_i are paired with both C_i and M_i to train the skip-gram model. We consider M_k to be the position of M_i in its own neighborhood. Similarly for C_k and C_i.

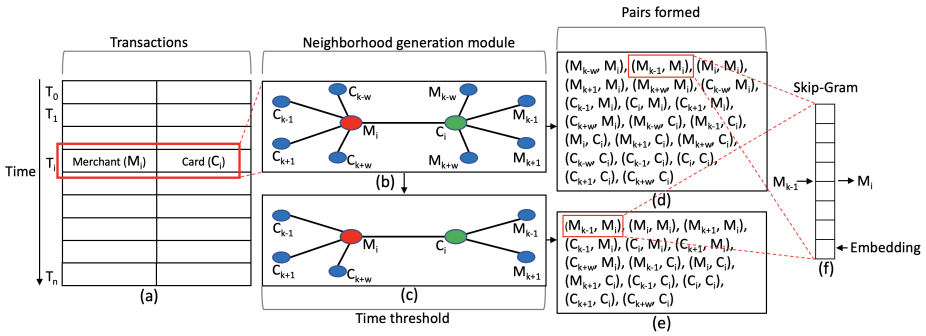

Fig. 1: CuRL/tCuRL embedding generation process: (a) For each transaction T_i, the pair of card C_i and merchant M_i, is selected (b) The neighborhood generation module constructs the joint neighborhood of the card and merchant (c) The time threshold used in `tCuRL`, helps in removing uncorrelated interactions by limiting the card and merchant neighborhood to a dynamic time window (d,e) $card-card$, $card-merchant$, and $merchant-merchant$ pairs are formed from the collected neighborhood (f) The skip-gram model is trained on these coupled pairs to predict M_i and C_i from their neighborhood to learn cross-entity as well as homogenous relationships. The model weights are then used as embeddings for the cards and merchants

$$N_i^C = \bigcup_{-w \le j \le w} M_{k+j} \quad and \quad N_i^M = \bigcup_{-w \le j \le w} C_{k+j} \tag{1}$$

$$N_i = N_i^C \bigcup N_i^M \tag{2}$$

And the pairs formed for transaction T_i are,

$$\{(C, x);\ \forall x \in N_i\} \bigcup \{(M, x);\ \forall x \in N_i\} \tag{3}$$

As shown in equation 3, `CuRL` includes both 1st and 2nd order neighbors to form $card-card$, $card-merchant$, $merchant-merchant$ entity pairs which helps in capturing historic interaction pattern of entities involved.

4.2 tCuRL

Optimizing the analysis window to derive card and merchant neighborhoods is essential in building a robust solution. Hence, session-based `CuRL` is introduced in this paper named `tCuRL` which decides on analysis window (henceforth called session) based on card's current spend frequency. Session constrained card and merchant neighborhood for a transaction would ensure that only relevant interactions are captured between entities and the performance of the model can be improved. To explain this further, let us again consider N_i^C in equation (1) for card C. There is a possibility that difference between the time of the

Algorithm 1: The CuRL Framework

Result: Vector representations of Payment Entities
for *Transaction* T_i; $\forall 0 \leq i \leq n$ **do**
  M_i = T_i*(Merchant);*
  C_i = T_i*(Card);*
  N_i^M = *neighborhood*(M_i)*;*
  #*The cards* M_i *transacts with, immediately before and after* T_i*;*
  N_i^C = *neighborhood*(C_i)*;*
  #*The merchants* C_i *transacts with, immediately before and after* T_i*;*
  N_i^M = *Time-threshold*(N_i^M)*;*
  N_i^C = *Time-threshold*(N_i^C)*;*
  # *All cards/merchants that transacted with* M_i/C_i *before or after the time-threshold from* T_i *are removed (this step is skipped in CuRL);*
  $N_i = N_i^M \bigcup N_i^C$;
  $P_i^M = \{(M_i,x)\ \forall x \in N_i\}$;
  $P_i^C = \{(C_i,x)\ \forall x \in N_i\}$;
  $P_i = P_i^M \bigcup P_i^C$;
end
$Pairs = \bigcup_{1 \leq i \leq n} P_i$;
Embeddings = skipgram(Pairs);
#*The skip-gram model is trained to learn the entity embeddings;*

transactions between C and M_k and between C and M_{k+1} is large. Keeping both these merchants in the same card neighborhood would add noise and force the model to learn spurious interactions. To incorporate this `tCuRL` uses a dynamic analysis window, which ensures that in such a case, M_k and M_{k+1} would never come together in the same neighborhood. While calculating the dynamic analysis window we don't consider the merchants because, in the transaction network, merchants usually have a much higher degree than cards and also have a more uniform timestamp difference distribution. We found that a static window size works best for calculating merchant neighborhoods. Algorithm 1 provides the pseudo-code for `CuRL` and `tCuRL`.

5 Experiments

In this paper, we benchmark `CuRL` and `tCuRL` with five baseline algorithms of entity representation learning and evaluate the performance of the embeddings in fraud detection task on the metrics - AUCPR and F_1 score. The paper also benchmarks models on running time to create entity embeddings.

5.1 Baseline Models

The size of embeddings is kept the same i.e., 128 in all the baseline models for fair benchmarking. After we create the embeddings for the cards and merchants,

we use an internal classification tool to classify the transactions into fraud and non-fraud. The features fed into the fraud classification task are - card embedding and merchant embedding.

node2vec - The paper uses the StellarGraph implementation[1] of node2vec [12] in the experiments. The maximum length of random walk is set to 50 with 10 random walks per node. The skip-gram model uses a window size of 20, the min count is set to 0 and the number of epochs is set to 50. The *return hyperparameter* p is taken as 0.5 and the *in-out hyperparameter* q is set to 2.

metapath2vec - The paper again uses the StellarGraph implementation[1] of metapath2vec [7]. For the experiments, we consider the metapaths as *card-merchant-card* and *merchant-card-merchant* and define the maximum length of the random walk to be 50 with 2 random walks per root node. The word2vec based skip-gram model is used with a window size of 20, the min count set to 0 and the number of epochs is 50.

BigGraph - Although the dataset considered for the experiments is relatively small, BigGraph [17] is taken into consideration since transaction networks involve billions of transactions. For the experiments, we use the default parameters as used by the authors in the official code repository[2] of the paper. The learning rate is set to 0.001 and the model is run for 30 epochs.

LINE - For learning the node representations using LINE we use the implementation provided by the authors[3]. The negative sample size is set to 5 and 0.025 is taken as the starting learning rate. The order-1 and order-2 embeddings are concatenated as recommended by the authors.

BiNE - We used the official implementation provided by the authors[4]. The default parameters are used i.e, 32 maximum walks per vertex, learning rate of 0.01, walk-stopping probability of 0.15, and window size of 5.

Pin2Vec - This is our own implementation of Pin2Vec [18] as we could not find any implementation by the authors. We trained the skip-gram models for 50 epochs to generate embeddings. We use a window size of 5 to pick neighbors from the sequences generated, to train the skip-gram models. We also use a timed approach with Pin2Vec and we name it **tPin2Vec**. tPin2Vec uses a variable window size for picking neighbors from sequences to train the skip-gram model. The calculation of the variable window sizes is similar to what we described in Section 4.2 for `tCuRL`. For experimentation purposes, tPin2Vec uses the same parameters as Pin2Vec.

5.2 Proposed Models

CuRL generates pairs of cards and merchants which are used to train a single skip-gram model which learns the embeddings for cards and merchants. To get the best possible results we trained the skip-gram model for 30 epochs (*ep*) to

[1] `https://github.com/stellargraph/stellargraph`
[2] `https://github.com/facebookresearch/PyTorch-BigGraph`
[3] `https://github.com/tangjianpku/LINE`
[4] `https://github.com/clhchtcjj/BiNE`

generate embeddings of length 128 (len). We use a window size of 15 (ws) to pick entity neighbors from the transaction graph. The min count parameter ($minc$) of the skip-gram model was set to 3.
tCuRL incorporates a time threshold to have a variable window size for forming pairs to train the skip-gram model. The cards are grouped into bins based on the number of transactions they are making. Further, for each bin, a time threshold is decided for the time between two transactions in a session, in a way such that 95% of the consecutive transactions have a time difference lesser than the threshold. We tune this percentage as a hyperparameter, $perc\_thresh$. The other parameters have the same values as `CuRL`.

6 Results

All the models were trained on a machine with a 2.6 GHz 6-Core Intel Core i7 processor with 16 GB of RAM. The runtimes are given in Table 4 while Table 3 gives the fraud detection metrics for the different algorithms. Section 6.1 compares our models' performance with the baseline models. In Section 6.2, we show the effect of the various parameters of the `tCuRL` model on its performance.

Algorithm	AUCPR	F_1 Score	Precision	Recall
metapath2vec	0.156	0.212	0.298	0.113
node2vec	0.159	0.230	0.267	0.158
BigGraph	0.771	0.833	0.890	0.770
LINE	0.777	0.823	0.865	**0.792**
BiNE	0.776	0.826	0.865	0.790
Pin2Vec	0.764	0.820	0.890	0.760
tPin2Vec	0.782	0.817	0.870	0.770
CuRL	0.797	0.829	0.878	0.785
tCuRL	**0.847**	**0.846**	**0.920**	0.783

Table 3: Fraud classification results contrasting CuRL and tCuRL with the baseline methods on relevant metrics

Algorithm	Time
metapath2vec	42 minutes
node2vec	77 minutes
BigGraph	3 minutes
LINE	50 seconds
BiNE	20 hours
Pin2Vec	50 seconds
tPin2Vec	2.33 minutes
CuRL	3 minutes
tCuRL	3.13 minutes

Table 4: Comparing embedding generation time of CuRL and tCURL with the baseline methods

6.1 Fraud Detection

We found that `tCuRL` outperforms all the baselines in terms of the AUCPR score. In general for imbalanced datasets like ours, AUCPR gives a good measure of performance. For the sake of comparison, we also present the results on other metrics as well. Methods that use simple random walks to generate sequences like metapath2vec and node2vec perform quite poorly. BiNE also employs random walks but preserves entity relationships and is able to achieve better performance.

However, these methods require a lot of time to generate embeddings. BigGraph, Pin2Vec, and tPin2Vec are efficient in generating embeddings but they fail to capture the cross-interactions between cards and merchants. The embedding generation process using LINE is also efficient but it learns the embeddings for first and second order neighbors separately rather than learning a single embedding. From Table 3, we see that `CuRL` and `tCuRL` outperform all the baseline models and also have an efficient embedding generation process, as shown in Table 4.

Furthermore, to show the quality of our generated embeddings, we use t-SNE to plot the transaction embeddings from `tCuRL` in a two-dimensional space. The transaction embeddings are the concatenation of the card and merchant embeddings involved in the transaction. It can be seen in Figure 2 that most of the fraudulent transactions separate out from the non-fraudulent transactions, which makes it easier for the classification model to identify the frauds.

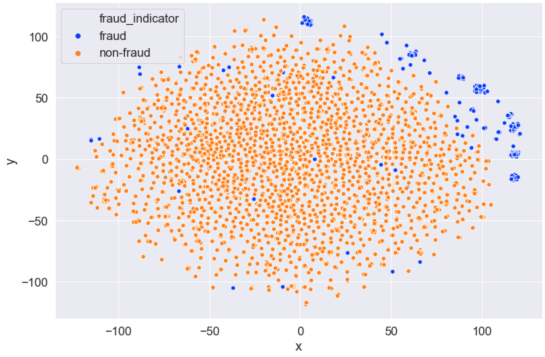

Fig. 2: t-SNE plot of the transaction embeddings from `tCuRL` (concatenation of card and merchant embeddings) mapped to a 2-D space.

6.2 Parameter Sensitivity

`CuRL` and `tCuRL` involve many parameters and this section examines how changing them affects the performance. Since `CuRL` and `tCuRL` share all parameters except the percentage threshold and Table 3 shows that `tCuRL` outperforms `CuRL`, we limit our parameter sensitivity experiments to `tCuRL`.

Table 5 shows that a smaller embedding size in the case of our model gives a better AUCPR. However, we keep an embedding size of 128 as transaction datasets are much larger in practice and a larger embedding size can store more information. This also helps to maintain uniformity among experiments. Similarly, Table 6 shows that increasing the window size improves the AUCPR for the model. This can be because increasing the window size while generating pairs allows the model to learn from a larger entity neighborhood. From Table 7, we see that the AUCPR of the model is highest at the middle min count values. A low

len	AUCPR	F_1 Score	Precision	Recall
16	**0.789**	0.821	0.874	0.774
32	0.786	0.816	0.874	0.774
64	0.778	0.816	0.865	0.772
128	0.784	0.819	0.864	**0.778**
256	0.766	**0.824**	**0.894**	0.764
512	0.780	0.823	0.876	0.776

Table 5: Effect of varying embedding length on performance metrics

ws	AUCPR	F_1 Score	Precision	Recall
2	0.773	0.819	0.865	0.778
3	0.761	0.813	0.867	0.765
5	**0.784**	0.819	0.864	0.778
7	0.778	0.833	**0.909**	0.769
10	0.782	0.819	0.863	**0.779**
12	0.780	**0.828**	0.904	0.764
15	**0.784**	0.821	0.883	0.767

Table 6: Effect of varying window size for neighborhood calculation

minc	AUCPR	F_1 Score	Precision	Recall
1	0.784	0.819	0.864	0.778
2	0.775	0.811	0.861	0.766
3	**0.789**	0.825	0.879	0.777
4	0.783	0.828	**0.891**	0.773
5	0.774	0.811	0.859	0.768
6	0.788	0.823	0.883	0.771
8	0.766	0.822	0.863	**0.785**
10	0.765	**0.829**	0.885	0.780

Table 7: Effect of varying minimum count of cards for skip-gram training

ep	AUCPR	F_1 Score	Precision	Recall
5	0.765	0.816	0.862	0.774
10	0.767	0.816	0.864	0.773
15	0.781	0.823	0.873	0.778
20	0.783	0.819	0.864	0.778
25	0.784	0.819	0.864	0.778
30	**0.787**	0.823	0.876	0.776
35	0.782	0.820	0.869	0.776
40	0.778	**0.824**	0.867	**0.783**
45	0.778	0.819	**0.893**	0.756
50	0.786	0.819	0.876	0.769

Table 8: Effect of varying number of epochs for skip-gram training

perc_thresh	AUCPR	F_1 Score	Precision	Recall
60	0.780	0.816	0.864	0.774
65	0.778	0.823	0.947	0.801
70	0.787	0.820	0.872	0.774
75	0.790	0.819	0.864	0.778
80	0.784	0.819	0.864	0.778
85	0.790	0.834	**0.905**	0.774
90	0.802	**0.836**	0.868	**0.805**
95	**0.810**	0.833	0.879	0.792
100	0.797	0.829	0.878	0.785

Table 9: Varying the percentage of transactions that have time intervals below the threshold

min count can lead to a lot of noise during training. Whereas, a large min count is also not acceptable as we could be missing out on important entity relationships. It can be seen from Table 8 that the AUCPR of the model drops when the number of epochs is too large, which could be because of the model overfitting on the training data. The results in Table 9 show that a higher AUCPR score is obtained on increasing the percentage threshold. Note that `CuRL` is the case when the percentage threshold is 100% and time between transactions is not

considered. Also, it can be observed that reducing the percentage threshold to just 95% from 100% effectively removes the noise and improves the AUCPR considerably. Reducing it further leads to loss of information.

7 Conclusion

In this paper, we have proposed a method to generate embeddings for cards and merchants from transaction data. Our method introduced a novel technique for creating pairs of cards and merchants, that effectively captured their cross interactions, before applying the skip-gram model to generate the embeddings. We also introduced a dynamic session-based approach to reduce the noise in the embedding generation process by limiting the entity neighborhoods created while generating pairs. We also discussed our model's scalability and efficiency, while processing large number of transactions, and have compared the results with the baselines.

Future work directions could include testing the performance of our generated embeddings for other financial downstream tasks. Additional card and merchant features (geographical data, spend information) can also be incorporated in the embedding generation process. Researchers can also look into a more extensive hyperparameter testing for the experiments. Furthermore, efficient dynamic updation of embeddings is also something that can be looked into.

References

1. Akhilomen, J.: Data mining application for cyber credit-card fraud detection system. In: Perner, P. (ed.) Advances in Data Mining. Applications and Theoretical Aspects. pp. 218–228. Springer Berlin Heidelberg (2013)
2. Bahnsen, A.C., Stojanovic, A., Aouada, D., Ottersten, B.: Improving credit card fraud detection with calibrated probabilities. In: SDM (2014)
3. Bruss, C., et al.: DeepTrax: Embedding graphs of financial transactions. 2019 18th IEEE International Conference On Machine Learning And Applications (2019)
4. Cao, B., Mao, M., Viidu, S., Yu, P.S.: Hitfraud: A broad learning approach for collective fraud detection in heterogeneous information networks. In: 2017 IEEE International Conference on Data Mining (ICDM). pp. 769–774 (2017)
5. Chandola, V., et al.: Anomaly detection: A survey. ACM Comput. Surv. (2009)
6. Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P.: Smote: Synthetic minority over-sampling technique. Journal of Artificial Intelligence Research (2002)
7. Dong, Y., Chawla, N.V., Swami, A.: metapath2vec: Scalable representation learning for heterogeneous networks. In: KDD '17. pp. 135–144. ACM (2017)
8. Eberle, W., Holder, L.: Mining for insider threats in business transactions and processes. In: 2009 IEEE Symposium on Computational Intelligence and Data Mining. pp. 163–170 (2009)
9. El hlouli, F.Z., Riffi, J., et al.: Credit card fraud detection based on multilayer perceptron and extreme learning machine architectures. In: International Conference on Intelligent Systems and Computer Vision (2020)
10. Gallagher, B., Eliassi-Rad, T.: Leveraging label-independent features for classification in sparsely labeled networks: An empirical study. In: Lecture Notes in Computer Science: Advances in Social Network Mining and Analysis. Springer

11. Gao, M., Chen, L., He, X., Zhou, A.: BiNE: Bipartite network embedding. SIGIR '18, Association for Computing Machinery (2018)
12. Grover, A., Leskovec, J.: Node2vec: Scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. KDD '16, Association for Computing Machinery (2016)
13. Henderson, K., Gallagher, B., Li, L., Akoglu, L., Eliassi-Rad, T., Tong, H., Faloutsos, C.: It's who you know: graph mining using recursive structural features. KDD
14. Hooi, B., Song, H.A., Beutel, A., Shah, N., Shin, K., Faloutsos, C.: Fraudar: Bounding graph fraud in the face of camouflage. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining
15. Huang, D., Mu, D., Yang, L., Cai, X.: Codetect: Financial fraud detection with anomaly feature detection. IEEE Access **6**, 19161–19174 (2018)
16. Jurgovsky, J., Granitzer, M., et al.: Sequence classification for credit-card fraud detection. Expert Syst. Appl. (2018)
17. Lerer, A., Wu, L., et al.: PyTorch-BigGraph: A Large-scale Graph Embedding System. In: Proceedings of the 2nd SysML Conference. Palo Alto, CA, USA (2019)
18. Liu, D.C., Rogers, S., Shiau, R., Kislyuk, D., Ma, K.C., Zhong, Z., Liu, J., Jing, Y.: Related pins at pinterest: The evolution of a real-world recommender system. In: Proceedings of the 26th International Conference on World Wide Web Companion
19. Mikolov, T., Chen, K., Corrado, G.S., Dean, J.: Efficient estimation of word representations in vector space
20. Misra, S., Thakur, S., Ghosh, M., Saha, S.K.: An autoencoder based model for detecting fraudulent credit card transaction. Procedia Computer Science (2020), International Conference on Computational Intelligence and Data Science
21. Molloy, I., Chari, S., et al.: Graph analytics for real-time scoring of cross-channel transactional fraud. In: Financial Cryptography and Data Security (2017)
22. Moschini, G., Houssou, R., Bovay, J., Robert-Nicoud, S.: Anomaly and Fraud Detection in Credit Card Transactions Using the ARIMA Model (2020)
23. Patki, N., Wedge, R., Veeramachaneni, K.: The Synthetic Data Vault. In: 2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)
24. Perozzi, B., Al-Rfou, R., Skiena, S.: DeepWalk: Online Learning of Social Representations. In: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2014)
25. Salim Hasham, R.H., Wavra, R.: Combating payments fraud and enhancing customer experience. `https://mck.co/2Qi4ead` (2018)
26. Shen, A., et al.: Application of Classification Models on Credit Card Fraud Detection. In: International Conference on Service Systems and Service Management
27. Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., Mei, Q.: Line: Large-scale information network embedding. In: Proceedings of the 24th International Conference on World Wide Web (2015)
28. The Nilson Report, Issue 1164: Card Fraud Worldwide 2010–2027 - Card Fraud Losses Reach $27.85 billion. `https://bit.ly/3uZ1v4D` (2019)
29. Van Vlasselaer, V., et al.: APATE: A novel approach for automated credit card transaction fraud detection using network-based extensions. Decision Support Systems (2015)
30. Zheng, E.H., Zou, C., Sun, J., Chen, L., Li, P.: Svm-based cost-sensitive classification algorithm with error cost and class-dependent reject cost. 2010 Second International Conference on Machine Learning and Computing pp. 233–236 (2010)
31. Zou, J., Zhang, J., Jiang, P.: Credit card fraud detection using autoencoder neural network (2019)