Session-Based Fraud Detection in Online E-Commerce Transactions Using Recurrent Neural Networks

Shuhao Wang[1], Cancheng Liu[2], Xiang Gao[2], Hongtao Qu[2], and Wei Xu[1]

[1] Tsinghua University, Beijing 100084, China
[2] JD Finance, Beijing 100176, China

Abstract. Transaction frauds impose serious threats onto e-commerce. We present CLUE, a novel deep-learning-based transaction fraud detection system we design and deploy at JD.com, one of the largest e-commerce platforms in China with over 220 million active users. CLUE captures detailed information on users' click actions using neural-network based embedding, and models sequences of such clicks using the recurrent neural network. Furthermore, CLUE provides application-specific design optimizations including imbalanced learning, real-time detection, and incremental model update. Using real production data for over eight months, we show that CLUE achieves over 3x improvement over the existing fraud detection approaches.

Keywords: Fraud detection · Web mining · Recurrent neural network

1 Introduction

Retail e-commerce sales are still quickly expanding [10]. A large online e-commerce website serves millions of users' requests per day. Unfortunately, frauds in e-commerce have been increasing with legitimate user traffic, putting both the financial and public image of e-commerce at risk [5]. In 2015, Internet Crime Complaint Centre (IC3) has received about 280,000 complaints, which directly led to the financial loss of over one billion USD [16].

Two common forms of frauds in e-commerce websites are *account hijacking* and *card faking* [12]: Fraudsters can steal a user's account on the website to use her account balance, or use a stolen or fake credit card to register a new account. Either case causes losses for both the website and its users. Thus, it is urgent to build effective fraud detection systems to stop such behavior.

Researchers have proposed different approaches to detect fraud [2] using various approaches from rule-based systems to machine learning models like decision tree, support vector machine (SVM), logistic regression, and neural network. All these models use aggregated *features*, such as the total amount of items a user has viewed over the last month, yet many frauds are only detectable by using individual actions instead of aggregates. Also, as fraudulent behaviors change over time to avoid detection, simple features or rules become obsolete quickly. Thus, it is essential for a fraud detection system to 1) capture users' behaviors in a way that is as detailed as possible (knowns as *feature extraction*); and 2) choose algorithms to detect the frauds from the vast amount of data.

Legitimate User	Fraudulent User
Visit JD.com	**#1**
Search `Apple`	Visit JD.com
Visit `Apple iPad Pro 9.7-inch (128G WLAN, Gold)`	Search `Game Card`
Visit `Apple iPad Pro 9.7-inch (128G WLAN + Cellular, Gold)`	Visit `Shanda Game Card (10,000 Game Points)`
Visit `Apple iPad Pro 9.7-inch (128G WLAN, Space gray)`	Checkout `Shanda Game Card (10,000 Game Points)`
...	**#2**
Visit `Apple iPad Pro 9.7-inch (128G WLAN, Gold)`	Visit JD.com
Visit `Apple iPad Pro 9.7-inch (128G WLAN, Rose gold)`	Search `Apple`
Visit `Apple iPad Pro 9.7-inch (32G WLAN, Rose gold)`	Visit `Apple iPad Pro 9.7-inch (128G WLAN, Rose gold)`
...	Visit `Apple MacBook Air 13.3-inch (128G)`
Visit `Apple iPad Pro 9.7-inch (128G WLAN, Gold)`	Visit `Apple iPhone 7 Plus (128G, Gold)`
Visit `Apple iPad Pro 9.7-inch (32G WLAN, Gold)`	...
Visit `Apple iPad Pro 9.7-inch Case`	Visit `Apple iPad Pro 9.7-inch (128G WLAN, Rose gold)`
Visit `Apple iPad Pro 9.7-inch Screen Protector`	Visit `Apple iMac 21.5-inch`
Visit `Apple iPad Pro 9.7-inch (128G WLAN, Rose gold)`	Visit `Apple iPhone 6 (32G, Space gray)`
Checkout `Apple iPad Pro 9.7-inch (128G WLAN, Rose gold)`	Checkout `Apple iPad Pro 9.7-inch (128G WLAN, Rose gold)`

Fig. 1. Examples of legitimate and fraudulent user browsing behaviors.

The algorithm must tolerate the dynamics and noise over a long period of time. Previous experience shows that machine learning algorithms outperform rule-based ones [2].

One of the most important piece of information for fraud detection is a user's browsing behavior, or the *sequence* of a user's *clicks* within a session. Statistically, the behaviors of the fraudsters are different from legitimate users. Real users browse items following a certain pattern (left column of Fig. 1). They are likely to browse a lot of items similar to the one they have bought for research. In contrast, fraudsters behave more uniformly (e.g. go directly to the items they want to buy, which are usually virtual items, such as #1 on the right column), or randomly (e.g. browse unrelated items before buying, such as #2 on the right column. Note that in this case, although every item is from Apple, they are not related as they contain both PC products, cell phones, and tablets). Thus, it is important to capture the sequence of each user's clicks, while automatically detect the abnormal behavior patterns.

We describe our experience with CLUE, a fraud detection system we have built and deployed at JD.com. JD is one of the largest e-commerce platforms in China serving millions of transactions per day, achieving an annual gross merchandise volume (GMV) of nearly 100 billion USD. CLUE is part of a larger fraud detection system in the company. CLUE complements, instead of replacing, other risk management systems. Thus, CLUE only focuses on users' purchase sessions, while leaving the analysis on users' registration, login, payment risk detections, and so on, to other existing systems.

CLUE uses two deep learning methods to capture the users' behavior: Firstly, we use Item2Vec [3], a technique similar to Word2Vec [21], to learn to embed the details of each click (e.g. the item being browsed) into a compact vector representation; Secondly,

we use a recurrent neural network (RNN) to capture the sequence of clicks, revealing the browsing behaviors on the time-domain.

In practice, there are three challenges in the fraud detection applications:

1) The number of fraudulent behaviors is far less than the legitimate ones [2,22], resulting in a highly imbalanced dataset. To capture the degree of imbalance, we define the *risk ratio* as the number of the portion of fraudulent transactions in all transactions. The typical risk ratio in previous studies is as small as 0.1% [4]. We use a combination of under-sampling legitimate sessions and thresholding [24] to solve the problem.

2) As the user browsing behaviors, both legitimate and fraudulent, change over time, we observe significant *concept drift* phenomenon [2] (see Fig. 3(b)). To continuously fine-tune our model, we have built a mechanism that automatically fine-tunes the model with new data points incrementally.

3) There are tens of millions of user sessions per day. It is challenging to scale the deep learning computation. Our training process is based on TensorFlow [1], using graphics processing units (GPUs) and data parallelism to accelerate computation. The serving module leverages TensorFlow Serving framework, providing real-time analysis of millions of transactions per day.

In summary, our major contributions are

1. We propose a novel approach to capture detailed user behavior in purchasing sessions for fraud detection in e-commerce websites. Using a RNN-based approach, we can directly model user sessions using intuitive yet comprehensive features.
2. Although the session-modeling approach is general, we optimize it for the fraud detection application scenario. Specifically, we optimize for highly imbalanced datasets, as well as the concept drift problem caused by the ever-changing user behaviors.
3. Last but not least, we have deployed CLUE on JD.com serving over 220 million active users, achieving real-time detection of fraudulent transactions.

In the remainder of the paper, Sect. 2 introduces the data acquisition and feature extraction methods of CLUE. The RNN model and system architecture are given in Sect. 3 and 4. We evaluate its performance in Sect. 5. We review the related work in Sect. 6 and conclude in Sect. 7.

2 Data and Feature Extraction

In this section, we describe the feature extraction process of turning raw click logs into sequences representing user purchase sessions that we can feed into the deep learning model.

2.1 Data Preprocessing

The inputs to CLUE are raw web server logs from standard log collection pipelines. The server log includes standard fields like the requested URL, browser name, client operating system, etc. We remove all requests to helper objects like css, js or image

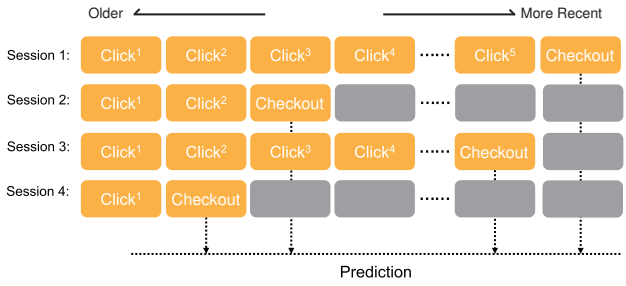


Fig. 2. Session padding illustration.

files, leaving only the request to the main web page. For this analysis, we remove all personal identifiable information (PII) to protect users' privacy.

The web server assigns a *session ID* on the first request of a user and maintains this ID throughout the user session. Every log entry contains the session ID, and we use this ID to sort the log into *user sessions* that CLUE takes as raw input.

Some sessions are associated with an *order ID*, an application specific field indicating the user's purchase actions. We can easily match the corresponding session ID with the order ID in the logs. In this work, we ignore all sessions that do not lead to an order, and we are working on capturing the non-purchasing sessions as a future direction.

We label the fraudulent orders using the business department's *case database* that records all fraudulent case reports. An order is labeled as fraudulent if and only if it appears as a complaint recorded in the case database. Of course, this labeling practice is quite incomplete, but we will show that even the simple labeling can help us identifying many fraudulent patterns.

2.2 Feature Extraction

Overview. The key feature that we capture is the sequence of a user's browsing behavior. Specifically, we capture the behavior using a sequence that consists of a number of clicks within the same session. As we only care about purchasing sessions, the final action in a session is always a *checkout* click. Figure 2 illustrates four sample sessions. Note that there are a different number of clicks per session, so we only use the last k clicks for each session. For short sessions with less than k clicks, we add empty clicks after the checkout (practically, we pad non-existing sessions with zeros to make the session sequences the same length). Our experience and [14] both indicate that the average number of clicks per session on e-commerce websites is about 7. In CLUE, we use a $k = 50$ that is more than enough to capture the entire sessions in most of the cases.

Two questions remain on how to perform fraud detection on these sequences: 1) How we can encode the information on a single click in a compact representation while keeping the information we need; and 2) How we represent the entire sequence (i.e. one row in Fig. 2). Previous work uses aggregated features and ignores the actual "sequence". The aggregation leads to information loss.

In this section, we focus on the first question. We leave the second question to Sect. 3, as the RNN is an end-to-end model that includes both creating the sequence feature and the classification.

Encoding Common Fields of a Click. The standard fields in click logs are straightforward to encode in a feature vector. For example, we include numerical data fields like *dwell time* (i.e. the time a user spends on a particular page) and page loading time. We encode the fields with categorical types using one-hot encoding. These types include the browser language, text encoding settings, client operating systems, device types and so on. Specifically, for the source IP field, we first look up the IP address in an IP geolocation database and encode the location data (to the city level) as categorical data.

Encoding the URL Information. The requested URL contains the most important information in a click. The URL schema is complicated for a large website like ours. We want to capture all detailed information, including which item the user is browsing, from the URL.

We mainly focus on two types of the pages: category pages that list a number of items of a category (e.g. toys), and item pages that show the detail of a single item. We can identify the type of a page using simple URL patterns. For example, "list.jd.com/$*$" and "item.jd.com/$*$" indicate each type. As there are only dozens of merchandise categories, we encode the category using one-hot encoding.

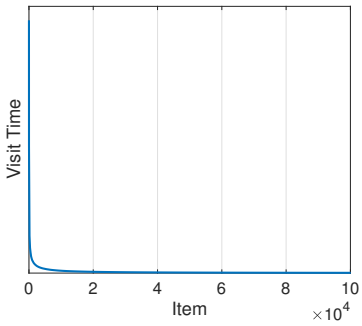
The difficulty is with the items, as there are hundreds of millions of items on JD.com. One-hot encoding will result in a sparse vector with hundreds of millions of dimensions per click, making it hard to further group these clicks into a session. Even worse, the one-hot encoding eliminates the correlations among separate items. For example, an iPad with 128GB flash is certainly more *similar to* an iPad with 64GB flash, than it is similar to a refrigerator. One-hot encoding ignores such similarities.

Thus we adapt the Item2Vec technique [3] to encode items. Item2Vec is a variation of Word2Vec [21], we regard each item as a "word", while regarding each session as a "sentence". Same as Word2Vec training process, we can train the Item2Vec embedding using historical user sessions containing different items. So the items that commonly appear at the same positions of a user session are embedded into vectors with smaller Euclidean distance. That is, the similarity of items are defined by user behavior, not the syntactic similarity.

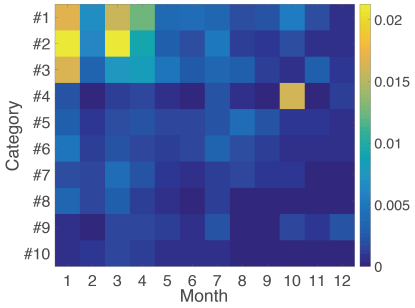
A practical question is how many dimensions we need for Item2Vec to capture the large variety of items. Of course, we need more dimensions to encode more items. We observe that the visit frequency follows a steep power-law distribution, as Fig. 3(a) shows. If we choose to cover 90% of all the items in the click history, we only need 25 dimensions for Item2Vec, a significant saving on data size. Data size reduction is essential as our RNN computation time highly depends on the input dimensions. We embed all other 10% items that appear rarely as the same constant vector.

It is common that people constantly add new items to the website every day, and the popularity of items change too. We take advantage of the feature of Item2Vec that can embed a new item conditioning on the existing ones to update the encoding in an online learning setting.

In summary, we embed a URL into three parts, the type, category and item, and Table 1 provides some examples.



(a) Click counts for the top 100,000 most frequently visited items.



(b) Heat map of the proportion of fraudulent sessions for ten randomly-selected categories over a 12-month period.

Fig. 3. Patterns in user click logs.

Table 1. Some sample URL embeddings

URL	Type	Category	Item Vector
www.jd.com	10	0	[0.0, 0.0, ...]
sale.jd.com/act/bWf0uqALD6pC4T.html	15	0	[0.0, 0.0, ...]
red.jd.com	12	0	[0.0, 0.0, ...]
search.jd.com/Search?keyword=Apple	2	0	[0.0, 0.0, ...]
list.jd.com/list.html?cat=670,671,674	3	6	[0.0, 0.0, ...]
item.jd.com/2538742.html	4	0	[-0.119, -0.077, $\cdots$]

3 RNN Based Fraudulent Session Detection

Our detection is based on sequences of clicks in a session (as Fig. 2 shows). We feed the clicks of the same session into the model in the time order, and we want to output a *risk score* at the last click (always the checkout action) for each session, indicating how suspicious the session is.

To do so, we need a model that can capture a sequence of actions. We find recurrent neural network (RNN) a good fit. RNNs have been successfully applied in applications such as language translation and speech recognition [23]. A RNN has a recursive structure across the time domain, thus it is able to "remember" the information in the previous action and carry the "memory" into the current learning process. In the meanwhile, the parameters of a RNN are shared among different time slots, enabling it to deal with sequential inputs with variable sizes. We feed each click to the corresponding time slot of the RNN, and the RNN finally outputs the risk score. Figure 4 illustrates the RNN structure and its input / output. In the following, we use "depth" and "width" to denote the layer number and the number of hidden units per layer, respectively.

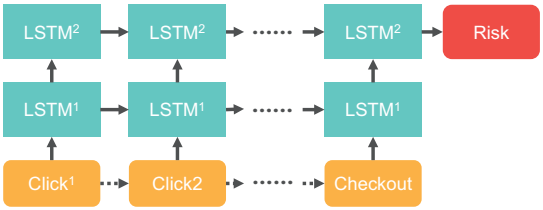


Fig. 4. Illustration of the RNN with LSTM cells.

Off-the-shelf RNN has problems of gradient exploding and gradient vanishing [18]. This is because the training process (optimization) of a RNN model involves back-propagation through time. Since a sequence can contain tens to hundreds of time slots, the back-propagation can cause the gradient to become too small or too large. Thus vanilla RNN does not work well on long-term dependency, a key requirement of fraud detection tasks. The common solution to these two problems are gradient clip and variant RNN cells, and there are two important types of variant RNN cells, *long short-term memory (LSTM)* and *gated recurrent unit (GRU)* [8,15]. The LSTM cells use "memory units" to learn what information to memorize for long-term prediction. Similar to the LSTM cell, the GRU has gating units controlling the information flow without separate memory units. By default, we use LSTM in CLUE to characterize the long-term dependency of the prediction on the previous clicks. In Sect. 5, we also compare the performance of the GRU alternative.

3.1 Dealing with Imbalanced Datasets

One practical problem in fraud detection is the highly imbalanced dataset (as frauds are rare comparing to normal sessions). There are two classes of approaches to deal with the imbalanced data problem [13], either on data level or on model level. On the data level, people use over-sampling or data synthesis to increase the minority class, or use under-sampling to reduce the majority. On the model level, people use *cost-sensitive learning* to impose a larger punishment on a misclassified minority class. In CLUE, we employ both data and model level approaches.

On the data level, we under-sample the legitimate sessions by random skipping, boosting the risk ratio to around 0.5%. We perform the under-sampling on both the training and validation sets, so they have the same distribution of these two classes. After under-sampling, the dataset contains 1.6 million sessions, among which 8,000 are labeled as fraudulent. We use about 6% (about 100,000 sessions) of the dataset as the validation set. We choose test sets, with the risk ratio of 0.1%, from the next continuous time period (e.g. two weeks of data), which is outside of the 1.6 million sessions.

On the model level, we leverage the thresholding approach [24] to implement cost-sensitive learning. By choosing the threshold from the range $[0, 1]$, we can obtain an application specific punishment level imposed on the model for misclassifying minority classes (false negatives) vs. misclassifying the normal class (false positives).

3.2 Model Update

Figure 3(b) illustrates the fact that fraudulent behaviors changes over time (known as the *concept drift* phenomenon). In this figure, we plot a heat map of the proportion of fraudulent sessions for ten randomly-selected categories over a 12-month period. It is clear that the fraud behavior changes over time. Many frauds occurred on categories 1-3 during the first four months, and they stopped on the fifth (probably because some upgrades of the fraud detection system stopped them). Then the number of frauds rose again on the tenth month on category 4. It is evident that frauds exhibits clear concept drift phenomenon.

To solve the concept drift issue, there are two general approaches to updating our model. The first approach is to train a new model using full data. However, with the increase of data amount accumulated across time, the model update process can be lengthy. Another approach is to use incremental data to fine-tune the current model. Our experience shows that the incremental update works both efficiently and achieves comparable accuracy as the full update.

In order to ensure the model quality, we run the updated model for the next couple days with incoming sessions in a separate quality assurance module that runs independently with the production system, to evaluate its performance (see Fig. 5). If we observe better performance over the current model, we switch the updated model into production.

4 System Architecture and Operation

We have deployed CLUE in real production, analyzing millions of transactions per day. From an engineering point of view, we have the following design goals: 1) *Scalability*: CLUE should scale with the growth of the number of transactions; 2) *Real-time*: we need to detect suspicious sessions before the checkout completes in a synchronous way, giving the business logic a chance to intercept potential frauds; and 3) *Maintainability*: we must be able to keep the model up-to-date over time, while not adding too much training overhead or model switching cost.

4.1 Training - Serving Architecture

To meet the goals, we design the CLUE architecture with four tightly coupled components, as Fig. 5 shows.

Data Input. We import raw access logs from the centralized log storage into an internal session database within CLUE using standard ETL (i.e. Extract, Transform and Load) tools. During the import process, we sort the logs into different sessions. Then we join the sessions with the purchase database to filter out those sessions without an order ID.

Then we obtain the manual labels whether a session is fraudulent or not. We connect to the case database at the business units storing all fraud transaction complaints. We join with the case database (using the order ID) to label those known fraudulent sessions. Note that the join needs to happen on demand right before the training process. This is because the fraud case reports come in over time, and we always want to use most recent reports as training labels.

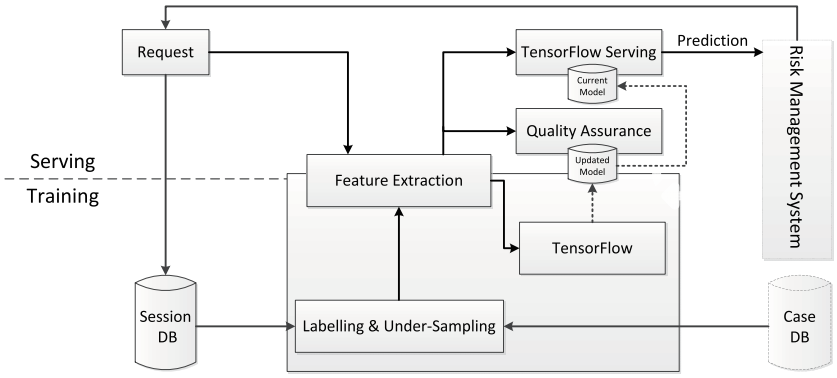


Fig. 5. The system architecture of CLUE.

Model Training. We perform under-sampling to balance the fraudulent and normal classes. Then the data preprocessing module performs all the feature extraction, including the URL encoding. Note that the item embedding model is trained offline. We then pass the preprocessed data to the TensorFlow-based deep learning module to train the RNN model. Between the stages, the intermediate data are serialized to disk.

Online Serving. After training and model validation, we transfer the trained RNN model to the TensorFlow Serving module for production serving. Requests containing session data from the business department are preprocessed using the same feature extraction module and then fed into the TensorFlow Serving system for prediction. Meanwhile, we persist the session data into the session database for further model updates.

Model Update. We perform periodic incremental updates to the model. It uses the latest updated model as the initial parameter and fine-tunes it with the incremental session data. Once the fine-tuned model is ready, and passes the model quality test as mentioned above, it is passed to TensorFlow Serving for production deployment.

4.2 Implementation Details

For the RNN Training, we set the initial learning rate to 0.001 and let it exponentially decay by 0.5 at every 5,000 iterations. We adopt Adam [17] for optimization with TensorFlow default configurations, by combining AdaGrad (Adaptive Gradient) [9] and RMSProp, Adam speeds up the training process better than standard stochastic gradient decent (SGD). The training process terminates when the loss on the validation set stops decreasing. The typical training duration is 12 hours (roughly 6-8 epochs).

We use TensorFlow as the deep learning framework [1] and leverage its built-in RNN network. Because of the highly imbalanced dataset, we raise the batch size to 512.

We train our model using a server with two Nvidia Tesla K40m GPUs, each with 12GB memory. The server is equipped with an Intel Xeon E5-2640 (2.60GHz) CPU and 128GB memory. To utilize both GPUs, we employ data parallelism.

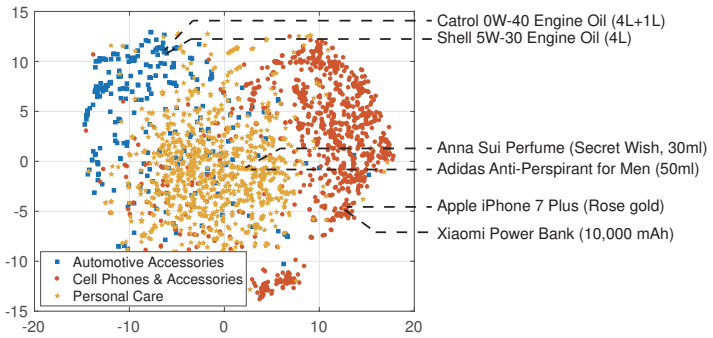


Fig. 6. Visualization of samples of item embeddings in three different categories.

The serving module contains tens of CPU-only servers, as we find CPUs are cost efficient while providing a satisfactory prediction latency.

5 Performance Evaluation

We first show that our URL embedding scheme produces a proper embedding with a short vector, and present our general detection performance on real production data. Then we evaluate the effects of different design choices of CLUE, and their effects on the model performance, such as different RNN structure, embeddings, and RNN cells. We also compare the RNN-based detection method with other features and learning methods. Finally, we evaluate the model update results.

5.1 URL Embedding Result

The performance of CLUE highly depends on the quality of features. The most important feature is the embedding of items using Item2Vec. Item2Vec does the embedding using the correlations of the items and the browsing behavior. To illustrate the effectiveness of the embeddings, we randomly select about 2,000 items from three different categories, i.e. automotive accessories, cell phones and accessories, and personal care. In Fig. 6, we plot their embeddings (25 dimensions) in a two-dimensional space using t-SNE [19]. We illustrate some sample item examples. We can see that many items are similar not only because they are syntactically similar but also because they are often browsed within the same session (e.g. the iPhone and Xiaomi Power Bank). Thus, we believe Item2Vec does reveal the statistical correlations among different items from a user browsing perspective.

5.2 Performance on Real Production Data

The best performance is achieved by an RNN model with 4 layers and 64 units per layer with LSTM cells. It is the configuration we use in production. We evaluate other RNN

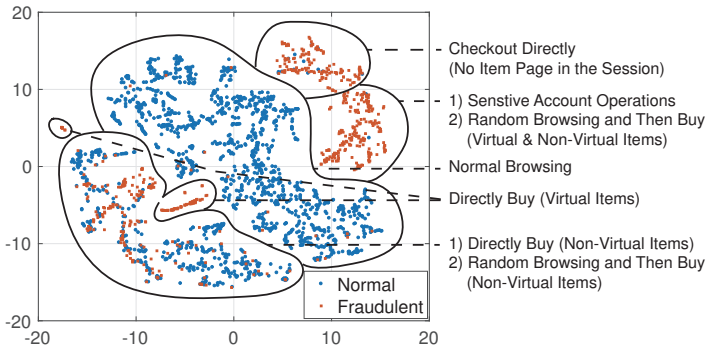


Fig. 7. Visualization of normal and fraudulent browsing behaviors using t-SNE.

structures in Sect. 5.3. Compared with traditional machine learning approaches (see Fig. 8(b)), CLUE achieves over 3x improvement over the existing system. Integrating CLUE with existing risk management systems for eight months in production, we have observed that CLUE has brought a significant improvement of the system performance.

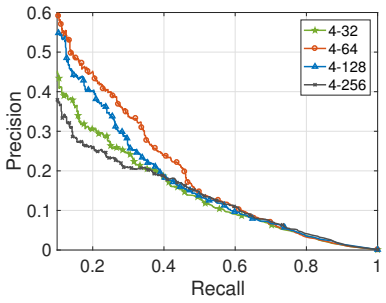
Visualization of the Features Captured by RNN. We would like to provide some intuition of what kind of features are captured by the RNN. To do so, we remove the last softmax layer in the RNN, take the output 64-dimensional vector from the previous layer (a.k.a. a *representation* learned by the RNN), and project the vector onto a two-dimensional space using t-SNE [19]. As Fig. 7 illustrates, each dot represents a user session and we color the dots using the ground truth. Interestingly, we observe a clear clustering / separation between normal and fraudulent user behaviors. Examining the data manually, we find different regions represent different browsing patterns. The most frequent patterns within various regions are given on the right side. Therefore, the features learned by the RNN characterize different types of browsing behaviors.

5.3 Effects of Different RNN Structures

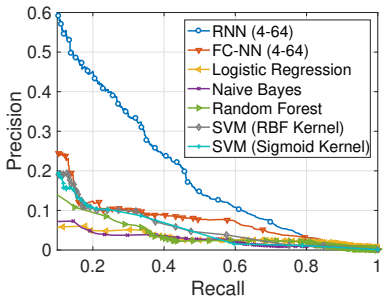
Our model outputs a numerical probability of a session being fraudulent. We use a threshold T to provide a tradeoff between precision and recall [24]. Varying T between $[0, 1]$, we can get the precision of our model corresponding to a particular recall. Figure 8(a) shows the performance of RNNs with different widths using the Precision-Recall (P-R) curve. Throughout the evaluation, we use 4 layers for the RNN model.

The previous study points out that wider neural networks usually provide better memorization abilities, while deeper ones are good at generalization [7]. We want to evaluate the width and depth of the RNN structure to the fraud detection performance. We use α-β RNN to denote a RNN structure with α layers and β hidden units per layer.

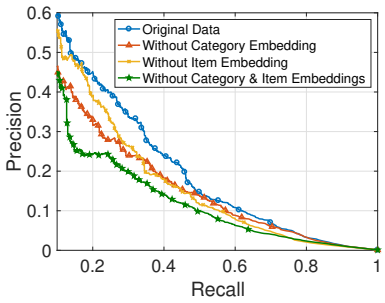
Table 2 provides the fraud detection precision with 30% recall, using different α and β. We see that given a fixed width, the performance improves with the depth increases. However, once the depth becomes too large, overfitting occurs. We find a 4-64 RNN



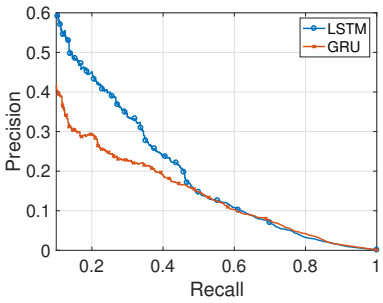
(a) Performance of 4 layer RNNs with different widths.



(b) Performance of 4-64 RNN, 4-64 FC-NN and traditional machine learning approaches.



(c) Performance of the 4-64 RNN models with / without item and category embeddings.



(d) Performance of LSTM and GRU cells, the RNN structure is fixed to be 4-64.

Fig. 8. Experiment results.

performs the best, outperforming wider models such as 4-128 and 4-256. We believe the reason is that, given the relatively small number of fraudulent sample, the 4-128 RNN model begins to overfit. Also, we can see that the generalization ability of a model seems more important than its memorization ability in our application.

5.4 Compare with Traditional Methods

We show that CLUE performs much better than traditional features and learning methods including logistic regression, naive Bayes, SVM, and random forest. Furthermore, we have investigated the performance of fully connected neural networks (FC-NNs). To leverage these methods, we need to leverage traditional feature engineering approaches by combining the time-dependent browsing history data into a fixed length vector. We follow many related researches and use *bag of words*, i.e., count the number of page views of different types of URLs in a session, and summarize the total dwell time of these URLs into the feature vector. Note that with bag-of-word, we cannot leverage the

Table 2. Precision of different RNN structures under the recall of 30%

#Layer / #Unit	32	64	128	256
1	19.3%	23.1%	24.3%	25.1%
2	23.4%	23.6%	26.1%	27.2%
3	24.7%	24.6%	29.0%	27.8%
4	24.8%	**33.8**%	26.4%	20.8%

category and item embedding approaches as introduced in Section 2, but we only use one-hot encoding for these data. We plot the results in Fig. 8(b).

FC-NN performs better than other traditional machine learning methods, indicating that with abundant data, deep learning is not only straightforward to apply but also performs better. Meanwhile, with more detailed feature extraction and time-dependent learning, RNNs perform better than FC-NNs. Therefore, we can infer that our performance improvement comes from two aspects: 1) By using more training data, deep learning (FC-NN) outperforms traditional machine learning approaches; 2) RNN further improves the results over FC-NN as it captures both sequence information, and it allows us to use detailed category and item embeddings in the model.

5.5 Effects of Key Design Choices

Here we evaluate the effectiveness of various choices in CLUE's feature extraction and learning.

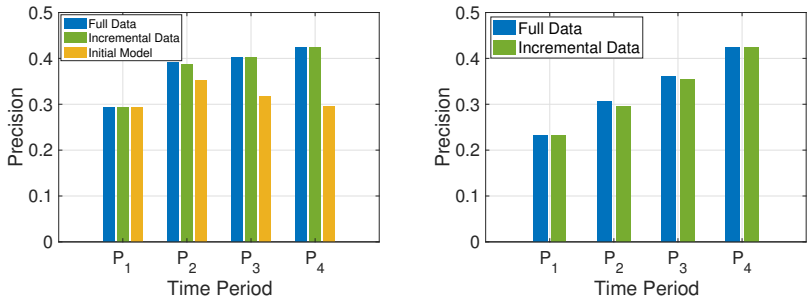
Category and Item Embeddings. To show that category and item data are essential features for fraud detection, we perform the following experiment by removing these features from our click data representation. Figure 8(c) shows that the detection accuracy is significantly lower without such information. Clearly, fraudulent users are different from normal users not because they are performing different clicks, but the real difference is *which* item they click on and in what *order*.

Using GRU as RNN Cell. Except for the LSTM, GRU is also an important RNN cell type that deals with the issues in vanilla RNN. Here we investigate the performance of RNNs with GRU cells. In Fig. 8(d), we compare the performance of different RNN cells. We find the LSTM shows better performance.

5.6 Model Update

As we discuss in Sect. 3.2, the fraud patterns change significantly over time. Thus, it is necessary to show that our method can adapt to these pattern changes by incrementally fine-tuning the RNN model.

To show the model update effectiveness using historical data, we consider a four consecutive equal-length time periods $P_1, \ldots, P_5$. The proportion of the number of sessions contained in these periods is roughly 2:2:1:1:1. We perform two sets of experiments, and both show the effectiveness of our model update methods.



(a) Precision for the next time period. (b) Precision for the last time period.

Fig. 9. The precision of the models trained with incremental and full data for the (a) next time period and (b) last time period, the recall is fixed to be 30%.

First, we evaluate the performance of using history up to P_n to predict P_{n+1}. We compare the performance of three update strategies: *incremental update*, *full update* (i.e. retrain the model from scratch using all history before the testing period) and *no update*. We use data from P_1 to train the initial model. Then we compute the precision (setting the recall to 30%) for the following four time periods using these three strategies. Figure 9(a) presents the results. We can see that incremental update achieves similar performance as the full model update, while the no update strategy performs the worst. It is also interesting that P_2 actually works worse than P_3 and beyond. We believe it is because the training data from P_1 is too few to produce a reasonable model.

Second, we show the performance of using different amounts of training data to predict the last time period, P_5. From Fig. 9(b), we can see that using either the full model update or incremental update, adding more data significantly improves the prediction results. It is not only because we are adding more data, but also because we use training data that are closer to time P_5, and thus tends to have more similar distributions.

6 Related Work

Fraud Detection. Researchers have investigated fraud detection for a long time. Existing work focuses on credit card [11,20], insurance [25], advertisement [27,30], and online banking [6] fraud detections. The fraud detection approaches used in these work include rule-based, graph-based, traditional machine learning, convolutional neural network (CNN) approaches [2,11,28]. They have two drawbacks: 1) These models have difficulties in dealing with time-dependent sequence data; and 2) The model can only take aggregated features (like a count), which directly leads to the loss of the detailed information about individual operations. Our system extracts user browsing histories with detailed feature encodings, and it is able to deal with high-dimensional complex time-dependent data using RNN.

Recurrent Neural Networks. Out of the natural language domain, researchers have RNNs to model user behaviors in similar web server logs, especially in session-based

recommendation tasks [14,26,29]. To our knowledge, our work is the first application of the RNN-based model in fraud detection. Fraud detection is more challenging in that 1) there are too many items to consider, and thus we cannot use the one-hot encoding in these works; and 2) the frauds are so rare, causing a highly imbalanced dataset.

7 Conclusion and Future Work

Frauds are intrinsically difficult to analyze, as they are engineered to avoid detection. Luckily, we are able to observe millions of transactions per day, and thus accumulate enough fraud samples to train an extremely detailed RNN model that captures not only the detailed click information but also the exact sequences. With proper handling of imbalanced learning, concept drift, and real-time serving problems, we show that our features and model, seemingly detailed and expensive to compute, actually scale to support the transaction volumes we have, while providing an accuracy never achieved by traditional methods based on aggregate features. Moreover, our approach is straightforward, without too much ad hoc feature engineering, showing another benefit of using RNN.

As future work, we can further improve the performance of CLUE by building a richer history of a user, including non-purchasing sessions. We are also improving the item embedding by adding the image, title, and description of an item into the feature. Finally, we are going to apply the RNN-based representation of sessions into other tasks like recommendation or merchandising.

References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., et al.: Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv:1603.04467 (2016)
2. Abdallah, A., Maarof, M.A., Zainal, A.: Fraud detection system: A survey. Journal of Network and Computer Applications 68, 90–113 (2016)
3. Barkan, O., Koenigstein, N.: Item2vec: Neural item embedding for collaborative filtering. In: IEEE 26th International Workshop on Machine Learning for Signal Processing. pp. 1–6. IEEE (2016)
4. Bellinger, C., Drummond, C., Japkowicz, N.: Beyond the boundaries of smote. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. pp. 248–263. Springer International Publishing (2016)
5. Bianchi, C., Andrews, L.: Risk, trust, and consumer online purchasing behaviour: A chilean perspective. International Marketing Review 29(3), 253–275 (2012)
6. Carminati, M., Caron, R., Maggi, F., Zanero, S., Epifani, I.: BankSealer: A decision support system for online banking fraud analysis and investigation. Computers & Security 53, 175–186 (2015)
7. Cheng, H.T., Koc, L., Harmsen, J., Shaked, T., Chandra, T., Aradhye, H., Anderson, G., Corrado, G., Chai, W., Ispir, M., et al.: Wide & deep learning for recommender systems. In: Proceedings of the 1st Workshop on Deep Learning for Recommender Systems. pp. 7–10. ACM (2016)
8. Chung, J., Gulcehre, C., Cho, K.H., Bengio, Y.: Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv:1412.3555 (2014)

9. Duchi, J., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research 12(7), 2121–2159 (2011)
10. eMarketer, http://www.emarketer.com/newsroom/index.php/apparel-drives-retail-ecommerce-sales-growth/
11. Fu, K., Cheng, D., Tu, Y., Zhang, L.: Credit card fraud detection using convolutional neural networks. In: International Conference on Neural Information Processing. pp. 483–490. Springer (2016)
12. Glover, S., Benbasat, I.: A comprehensive model of perceived risk of e-commerce transactions. International Journal of Electronic Commerce 15(2), 47–78 (2010)
13. He, H., Garcia, E.A.: Learning from imbalanced data. IEEE Transactions on Knowledge & Data Engineering 21(9), 1263–1284 (2009)
14. Hidasi, B., Quadrana, M., Karatzoglou, A., Tikk, D.: Parallel recurrent neural network architectures for feature-rich session-based recommendations. In: Proceedings of the 10th ACM Conference on Recommender Systems. pp. 241–248 (2016)
15. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Computation 9(8), 1735 (1997)
16. 2015 Internet Crime Report, https://pdf.ic3.gov/2015_IC3Report.pdf
17. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv:1412.6980 (2014)
18. Kolen, J.F., Kremer, S.C.: Gradient Flow in Recurrent Nets: The Difficulty of Learning LongTerm Dependencies, pp. 237–243. Wiley-IEEE Press (2001)
19. Laurens, V.D.M., Hinton, G.: Visualizing data using t-SNE. Journal of Machine Learning Research 9(2605), 2579–2605 (2008)
20. Lim, W.Y., Sachan, A., Thing, V.: Conditional weighted transaction aggregation for credit card fraud detection. Advances in Information & Communication Technology 433, 3–16 (2014)
21. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Advances in Neural Information Processing Systems. pp. 3111–3119 (2013)
22. Phua, C., Alahakoon, D., Lee, V.: Minority report in fraud detection: Classification of skewed data. ACM SIGKDD Explorations Newsletter 6(1), 50–59 (2004)
23. Schmidhuber, J.: Deep learning in neural networks: An overview. Neural Networks 61, 85–117 (2015)
24. Sheng, V.S., Ling, C.X.: Thresholding for making classifiers cost-sensitive. In: National Conference on Artificial Intelligence. pp. 476–481 (2006)
25. Shi, Y., Sun, C., Li, Q., Cui, L., Yu, H., Miao, C.: A fraud resilient medical insurance claim system. In: Thirtieth AAAI Conference. pp. 4393–4394 (2016)
26. Tan, Y.K., Xu, X., Liu, Y.: Improved recurrent neural networks for session-based recommendations. In: Proceedings of the 1st Workshop on Deep Learning for Recommender Systems. pp. 17–22. ACM (2016)
27. Tian, T., Zhu, J., Xia, F., Zhuang, X., Zhang, T.: Crowd fraud detection in internet advertising. In: International Conference on World Wide Web. pp. 1100–1110 (2015)
28. Tseng, V.S., Ying, J.C., Huang, C.W., Kao, Y., Chen, K.T.: Fraudetector: A graph-mining-based framework for fraudulent phone call detection. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 2157–2166 (2015)
29. Wu, S., Ren, W., Yu, C., Chen, G., Zhang, D., Zhu, J.: Personal recommendation using deep recurrent neural networks in NetEase. In: International Conference on Data Engineering. pp. 1218–1229 (2016)
30. Xu, H., Liu, D., Koehl, A., Wang, H., Stavrou, A.: Click fraud detection on the advertiser side. In: European Symposium on Research in Computer Security. pp. 419–438. Springer (2014)