Using Recurrent Neural Networks To Predict Online Browsing Behavior And Detect Anomalies

Abhishek Easwaran[1] and Sai Kanaka Harsha Praneeth Nooli[1]

Abstract— **In cybersecurity, one of the most common causes of breach is credential theft. When a hacker steals a users credentials through phishing or social engineering, and subsequently logs in, the only way to detect and differentiate the hacker from the real user is to leverage behavioral models of the real user based on prior activities. In online browsing, historical behavior is often a good predictor of subsequent behavior, especially in an enterprise situation where users perform their daily activities in a very consistent manner. We have two primary goals in this project : 1) To capture this normal user and peer-group behavior using Recurrent Neural Networks and apply it in real time to predict the next moves thereby detecting any significant deviation from the normal behavior and 2) Train generic model for multiple users and given a set of inputs, classify the user(s) that those inputs belong to.**

In the first case, since online browsing results in a well defined sequence of HTTP requests between the browser/client and the server, RNNs can be used effectively to model these sequences, capturing the normal user behavior. Once the models are built, they can be used to predict the next clicks based on actual prior clicks. The probabilities of predicted clicks can then be compared with those of actual clicks from the user, thereby detecting anomalies based on significant deviations beyond a threshold. The system can then take an action on these anomalies as appropriate.

In the second case, given a set of input rows, it would be possible to classify which user it belongs to. Thus, if we have a set of records that are inputted which should have been of a particular user but the model predicts otherwise, it indicates that this is an anomalous record with the potential of being an attacker thereby triggering an anomalous flag.

Our experiments have shown that for the user-specific behavior modeling, using RNNs has provided an accuracy of about 80 percent for most users on the test set while for the classification problem, we have obtained test scores of about 75 percent. Thus, using RNNs has proved to be an effective solution to our problem.

I. INTRODUCTION

Credential theft, the first stage of a credential-based attack, is the process of stealing credentials. Attackers commonly use phishing for credential theft, as it is fairly cheap and extremely efficient. Its effectiveness relies on human involvement in an attempt to deceive employees, unlike malware and exploits, which rely on weaknesses in security defenses.

Phishing is a type of social engineering attack often used to steal user data, including login credentials and credit card numbers. It occurs when an attacker, masquerading as a trusted entity, dupes a victim into opening an email, instant message, or text message. The recipient is then tricked into clicking a malicious link, which can lead to the installation of malware, the freezing of the system as part of a ransomware attack or the revealing of sensitive information. An attack can have devastating results. For individuals, this includes unauthorized purchases, the stealing of funds, or identify theft.

Malware, short for malicious software, can easily be described as unwanted software that is installed in one's system without their consent. Viruses, worms, and Trojan horses are examples of malicious software that are often grouped together and referred to as malware. With so many people routinely surfing the Web, malware is often unwittingly spread across the Internet. Once a person enters a compromised website, their system can easily be contaminated after clicking on malicious links present and malware can be installed on their computer without their knowledge. Malware is an easy method of mass infection; compromising one website is easier than sending several emails. It targets all visitors/victims that visit a site and click on desirable links, converting their systems to act on the attackers behalf. There are different types of malware (Maladvertising, SEO Poisoning, Typosquatting and Social Engineering) that can be distributed through code on a website. Once

someone's been exposed to an attack, a download is activated through an exploit kit. That download can then infect a users computer to corrupt it, or steal information.

Online browsing behaviour is a subject of study not only in cybersecurity, but also in e-commerce and banking where it plays an equally important role for various companies to identify sets of users exhibiting certain online behaviour and performing necessary changes/modifications to the sites in order to make their impact stronger. Our focus is going to be in cybersecurity, particularly in identifying a distinct user-specific behaviour.

Rather than focusing on building firewalls that protect users, this research intends to build a user-specific behaviour model that catches the attackers if their behaviour exhibits movement away from the norm. For example, if one visits a banking platform and moves their cursor in a certain pattern, or type at a certain speed, user-specific behaviour models will be able to determine, on future visits, whether or not the user with someone's login credentials is actually the person himself or an attacker. Account takeovers, remote access (RAT), and MitB malware attacks could all be potentially thwarted by this approach.

One implementation of this could be explained as follows - If someone has a credit card and visits another state that is an 'anomaly' or a 'deviation from the normal pattern', the bank company generally has a firewall that requests an additional layer of authentication or directly rejects the transaction from taking place.

User-specific behaviour modeling works the same way, except it uses atypical variations in parameters, like typing speed, mouse movement, keyboard strokes, tapping force and swipe patterns instead of geographical location. Take this practical example: After a few logins, this system will learn that the 'user' tends to browse slowly, tap icons hard and type at an average speed. If someone gets hold of that person's login information and browses quickly, with fast typing speed and weak taps, the system will trigger a fraudulent use, and the hacker will be forced to provide further authenticating details (or, more likely, give up the effort).

The possibility of an imposter impersonating this specific behaviour is likely to be low and thus, it is important and can prove to be a very valuable layer of authentication if an effective behaviour model is built.

II. LITERATURE REVIEW

Previous studies involving clickstream data an user behaviour modeling based on that data have been performed. [1] talks about building an unsupervised system to capture dominating user behaviors from clickstream data and attempts to visualize the clusters in an intuitive manner. This paper talks about using clickstream data to generalize users into clusters rather than build a user-specific model, with other objectives in mind. In another research, [2] tries to perform User Behavior Modeling with Large-Scale Graph Analysis. Here, their primary focus is in trying to understand how fraudsters work by performing user-behaviour analysis of fraudsters and thereby build a scalable machine learning model that models abnormal as well as normal behaviour. [3] attempts to build two systems : the first is to build a semi-supervised system to capture malicious accounts using clickstream data and the second is an unsupervised system that aims to capture and understand the engrained user behavior. [4] is a paper that tries to understand consumer behaviour using Recurrent Neural Networks. This paper intends to find out, given a user when they open a e-commerce website, the probability that they will end up shopping. The output variable is a probability score and the paper compares their results to vector based models like Logistic Regression. This paper provides a strong background for our research in understanding user-behaviour by using RNN, though our area of research is different. Finally, [5] focuses on the use of RNNs to classify positive (fraudulent) and negative (normal) users by using http connection data. The paper is a decent starter point for our research, but to our knowledge, not a lot of research has been done in identifying user-specific behaviour anomalies based on clickstream data using RNNs. Our area of research is thus in building a user-specific behaviour model that identifies potential intruders by using clickstream data and applying RNN. [6] gives a good started on using LSTMs for the detection of anomalies in

time-series data. [7] talks about using Recurrent Neural Networks to leverage sequence to sequence learning.

III. PROBLEM DESCRIPTION

In online browsing, historical behavior is often a good predictor of subsequent behavior, especially in an enterprise situation where users perform their daily activities in a very consistent manner.

We describe our two goals as follows:

1. The first goal is to capture the normal user and peer-group behavior using LSTM based Recurrent Neural Networks and apply it in real time to predict the next moves thereby detecting any significant deviation from the normal behavior. In this case, we make use of http exchanges of individual users and attempt to detect the 'anomaly' in a given user's http exchanges. In short, we build a RNN model for a user and use that model to predict the next sequence of events of that user. Thus, if a normal user's next sequences deviate significantly from the predicted model, an anomaly is flagged.

2. In the second case, we build a generic RNN model by using multiple users' data. We have an output variable which indicates which user the given row belongs to. Thus, building this generic model will enable us to identify, given http exchanges, which user it should ideally belong to. Thus, any deviation from normal for the data would trigger a flag. For example, if an attacker has logged into user A's system and his http exchanges are analyzed, the RNN model will not flag the user as user A which indicates that there is an anomaly. An important parameter here is the 'threshold' value we get for each user. A lower threshold value for all users will indicate lesser probability of the data belonging to any user and hence we flag it as not belonging to any user. The importance of doing this is that we need to be sure that the model classifies the user at a higher probability which indicates the confidence of the model's predictions. Simply choosing the highest probability would not be appropriate in this case.

IV. DATA DESCRIPTION

Online browsing behaviour data generally consists of what is shown in the figure 1 below. Our dataset consists of a user ID and the username which is an identifier of the user. The username gives us an opportunity to perform user-specific behaviour modeling and to classify the users. Another two columns namely download bytes and upload bytes contains information of the number of bytes that have been downloaded or uploaded at a click for the given user.

Clicks and exchanges are the number of clicks of a person at a website. This is a very good indicator of user-specific behaviour. Timestamp is the information of the person at that given particular time.

The columns GET, POST, PUT, HEAD, DELETE are binary variables containing and giving information if a person has had the attributes during the visits.

Response Type is an indicator of the type of response that is obtained. Request path and referrer path are other features of the input that contain the request and the referrer paths when moving from a particular web page to another web page. HTTP messages are how data is exchanged between a server and a client. There are two types of messages: requests sent by the client to trigger an action on the server, and responses, the answer from the server. HTTP messages are composed of textual information encoded in ASCII, and span over multiple lines. The HTTP referer is an HTTP header field that identifies the address of the webpage (i.e. the URI or IRI) that linked to the resource being requested. By checking the referrer, the new webpage can see where the request originated.

Response codes are when a search engine or website visitor makes a request to a web server, a three digit HTTP Response Status Code is returned. This code indicates what is about to happen. A response code of 200 means "OK, here is the content you were asking for." A 301 says, "That page has moved, so I'll send you there now." And so on.

HTTP defines a set of request methods to indicate the desired action to be performed for a given resource. Although they can also be nouns, these request methods are sometimes referred to as HTTP verbs. Each of them implements a different semantic, but some common features are shared by a group of them: e.g. a request method can be safe, idempotent, or cacheable.

	aegis_user_id	download_bytes	upload_bytes	GET	POST	PUT	HEAD	DELETE	clicks	exchanges	user_name	timestamp
0	1.694620e+15	25486	60	8	1	0	0	0	6	10	subrota_mondal	2016-07-11 03:54:04+0000
1	1.694620e+15	20	0	1	0	0	0	0	1	1	subrota_mondal	2016-07-11 04:51:20+0000
2	1.694620e+15	26163	60	11	1	0	0	0	7	12	subrota_mondal	2016-07-11 07:09:38+0000
3	1.694620e+15	281	0	2	0	0	0	0	2	2	subrota_mondal	2016-07-11 08:10:24+0000
4	1.694620e+15	26872	60	12	1	0	0	0	7	13	subrota_mondal	2016-07-11 09:15:57+0000

Fig. 1. Dataset containing user-specific http sequence information

V. DATA PREPROCESSING

In our analysis, data preprocessing is probably the most important stage of all. Having the right data is vital for the model to learn and be able to predict the next sequence of moves and/or detect the user that it belongs to. We have converted our entire dataset into a set of categorical features. Although converting data into categorical features tends to create a loss of information, we have converted the data in such a way that we do not lose a lot of information. We have done the following:

1. The response type is converted into a categorical feature by using one hot encoding. Response type consists of text like text/html and others and these texts are converted into categorical features by selecting the first split of the /.

2. An important feature called 'node numbers' is generated. This is obtained by creating a list of all unique request paths and referrer paths. Then, after obtaining this list, the request path is compared to this list and the index where it is present in this list is considered to be the 'node number' of that request path. This ensures that our request paths are all obtained without loss of information and also that we can convert it into a categorical feature.

3. The response codes are generally in the 200s, 300s and 400s. We have converted this into a set of categorical features by binning it into 4 categories. 0 to 200 and more than 500 belong to one category. 200 to 300, 300 to 400 and 400 to 500 belong to the remaining 3 categories. As known, response codes of 200-300 and others have specific purposes as described earlier.

4. Clicks are also an important feature. We have a number for each row that denotes the click information. If the number increments in the next row, it means that there has been a click in that particular row. We have thereby converted our click data into 0s and 1s by assigning a 0 if there has been no click and 1 if there has been a click.

5. Request Method is one-hot encoded to convert into a categorical feature. Also, the host name is also one-hot encoded into a categorical feature.

VI. REVIEW OF RNNs

1. Recurrent Neural Networks - The idea behind RNNs is to make use of sequential information. In a traditional neural network we assume that all inputs (and outputs) are independent of each other. But for many tasks thats a very bad idea. If you want to predict the next word in a sentence you better know which words came before it. RNNs are called recurrent because they perform the same task for every element of a sequence, with the output being depended on the previous computations. Another way to think about RNNs is that they have a memory which captures information about what has been calculated so far. In theory RNNs can make use of information in arbitrarily long sequences, but in practice they are limited to looking back only a few steps (more on this later). Figure 3. shows what a typical RNN looks like. The diagram shows a RNN being unrolled (or unfolded) into a full network. By unrolling we simply mean that we write out the network for the complete sequence. For example, if the sequence we care about is a sentence of 5 words, the network would be unrolled into a 5-layer neural network, one layer for each word. The formulas that govern the computation happening in a RNN are as follows:

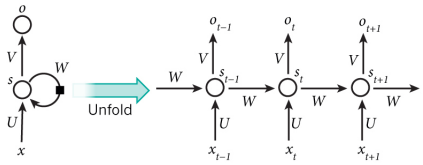

Fig. 2. RNN architecture

a. xt is the input at time step t. For example, x1 could be a one-hot vector corresponding to the second word of a sentence.

b. st is the hidden state at time step t. Its the memory of the network. st is calculated based on the previous hidden state and the input at the current step: st=f(Uxt + Wst-1). The function f

usually is a nonlinearity such as tanh or ReLU. s-1, which is required to calculate the first hidden state, is typically initialized to all zeroes.

c. ot is the output at step t. For example, if we wanted to predict the next word in a sentence it would be a vector of probabilities across our vocabulary.

2. LSTMs - In the mid-90s, a variation of recurrent net with so-called Long Short-Term Memory units, or LSTMs, was proposed by the German researchers Sepp Hochreiter and Juergen Schmidhuber as a solution to the vanishing gradient problem.

LSTMs help preserve the error that can be backpropagated through time and layers. By maintaining a more constant error, they allow recurrent nets to continue to learn over many time steps (over 1000), thereby opening a channel to link causes and effects remotely. This is one of the central challenges to machine learning and AI, since algorithms are frequently confronted by environments where reward signals are sparse and delayed, such as life itself. (Religious thinkers have tackled this same problem with ideas of karma or divine reward, theorizing invisible and distant consequences to our actions.)

LSTMs contain information outside the normal flow of the recurrent network in a gated cell. Information can be stored in, written to, or read from a cell, much like data in a computers memory. The cell makes decisions about what to store, and when to allow reads, writes and erasures, via gates that open and close. Unlike the digital storage on computers, however, these gates are analog, implemented with element-wise multiplication by sigmoids, which are all in the range of 0-1. Analog has the advantage over digital of being differentiable, and therefore suitable for backpropagation.

Those gates act on the signals they receive, and similar to the neural networks nodes, they block or pass on information based on its strength and import, which they filter with their own sets of weights. Those weights, like the weights that modulate input and hidden states, are adjusted via the recurrent networks learning process. That is, the cells learn when to allow data to enter, leave or be deleted through the iterative process of making guesses, backpropagating error, and adjusting weights via gradient descent.

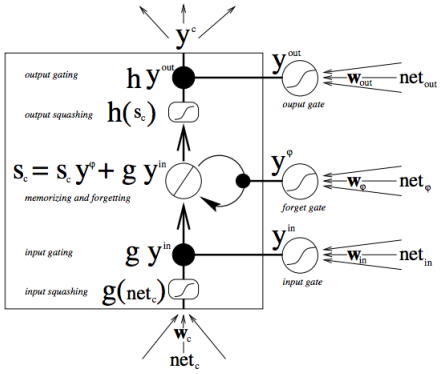

Fig. 3. LSTMs

Starting from the bottom, the triple arrows show where information flows into the cell at multiple points. That combination of present input and past cell state is fed not only to the cell itself, but also to each of its three gates, which will decide how the input will be handled.

The black dots are the gates themselves, which determine respectively whether to let new input in, erase the present cell state, and/or let that state impact the networks output at the present time step. Sc is the current state of the memory cell, and gyin is the current input to it. Remember that each gate can be open or shut, and they will recombine their open and shut states at each step. The cell can forget its state, or not; be written to, or not; and be read from, or not, at each time step, and those flows are represented here.

The large bold letters give us the result of each operation.

VII. OUR RNN ARCHITECTURE

Our RNN architectures for both the problems are as follows.

1. For the user-specific modeling of next sequences, we used various look backs (number of time steps we are going to look back in order to predict the next sequence) and a look back value of 12 provided us the most superior results. Also, our architecture here was a simpler one compared to the 2nd case. Here, we had a LSTM layer with 1024 neurons. The activation layer was reLu. This was followed by two more dense layers with 512 and 256 neurons in each layer and reLu as the activation function in both. The final layer is a

dense layer with the shape of the output variable y and sigmoid as the activation function. Adam optimizer with a learning rate of 0.001 was used and categorical cross entropy was the loss function. The batch size chosen was 64.

2. For the case of predicting the user to which a row belongs to, we have had many more rows of data. This called for a deeper and denser architecture. Here, we used a LSTM layer with 2048 neurons in the first layer. They were followed by 3 more dense layers with 1024, 512 and 256 neurons each. Each of the layers had the reLu activation layer. The final layer was a dense layer with the shape of the output variable y and sigmoid as the activation function. Adam optimizer with a learning rate of 0.001 was used and categorical cross entropy was the loss function. The batch size chosen was 64.

VIII. EVALUATION METRICS

We had two evaluation metrics for both the problems:

1. In the user-specific modeling where we predict the next sequences (rows) of data, our evaluation metrics was to compare the predicted values with the actual values. The actual values were obtained after performing a train-test split and the test data was used for prediction. The scoring was as follows - If for a given row, all the columns were correctly predicted, the score for that row was 1. If half or more than half but less than all the columns were correctly predicted, the score for that row was 0.5. Lastly, if less than half the rows were correctly predicted, the score for that row was 0.

2. In the case of prediction of the user given a row, our evaluation is as follows - The sigmoid layer in the final layer gives us a prediction probability for each user. For example if we have 5 users, it gives us a probability for each user (which need not sum up to 1). So, to evaluate our model, we have chosen a threshold of 0.5. That is, if the probability value is greater than 0.5, we select the highest probability and predict that that row's values belong to that particular user. In case the value is less than 0.5, we do not assign it to any user thereby effectively making that prediction's score 0. Thus finally, we calculate the accuracy by calculating the percentage of correct predictions.

IX. PROBLEMS WITH OUR DATASET AND HOW WE HANDLED IT

We created an artificial, biased, unbalanced dataset that helped us answer a few queries.

1. Is the model performing well with a biased dataset?

Although the performance of the actual data vs artificial data was expectedly going to be different, we felt it was important to train the architecture with a biased artificial data. Training with this biased data would give really high scores of accuracy thereby ensuring that the model is able to learn from data in the first place. We created a biased dataset by assigning certain high values of certain features to a particular user and using that similarly to create a dataset of multiple users with multiple feature, but a biased one. As was expected, in the case of classifying the user which the data belongs to, the model performed exceedingly well on that dataset.

2. Is the model performing well in an unbalanced dataset?

Another dataset consisted of 10 users with rows ranging from 200 to 2000 per user. This creation of an unbalanced dataset for each user was important in terms of validating the model and how it functions when there are different number of rows for each user. As was expected, in case where the row counts were different, given the same set of rows but of different counts, the model tended to be biased towards the user with more number of records.

Figure 4 shows the results for 5 users and the scores. As seen in the figure, we have tested the data using the node number parameter where we have nodes a, b and c. In this case, the input that has gone to each user is the same but the number of inputs are different. User 1 has 200 records, 2 has 400, 3 has 600, 4 has 800 and user 5 has 1000 records. Having these many records has shown that the final sigmoid output is biased towards users that have more records.

	node_c	node_a	node_b	usr-1_prob	usr-2_prob	usr-3_prob	usr-4_prob	usr-5-prob
0	0.0	1.0	0.0	0.158	0.310	0.473	0.622	0.785
1	0.0	0.0	1.0	0.158	0.312	0.473	0.624	0.786
2	1.0	0.0	0.0	0.392	0.443	0.493	0.534	0.590

Fig. 4. Figure showing the performance on the artificial dataset

X. EVALUATING THE MODEL USING ACTUAL DATA

A. Evaluating the user-specific behavior model

In our evaluation for the user-specific behaviour model, we have evaluated using the accuracy metric described earlier. Here, the performance of the model was for 30 users in all. A minimum of 1000 data points were considered to be needed to perform analysis.

For the user-specific behavior model, we trained the model using one LSTM layer which had 1024 neurons. This was followed by a 512 neuron dense fully connected layer. The final layer was the output layer with sigmoid as the activation function. The other layers had ReLu as the activation functions. The look back was varied widely and we came to a look back score of 12 for our analysis of individual user.

The reason for our model to be chosen is because of the graphs shown in Figures 5 and 6. Figure 5 shows the variation of batch size and how it affects the accuracy on the training and test set. As we can see, the greater the batch size, the lesser the scores.

Also, we can see from figure 6 the effect of dropout. As seen, the train accuracy is constantly reducing with increased values of dropout but the test accuracy is near about the same.

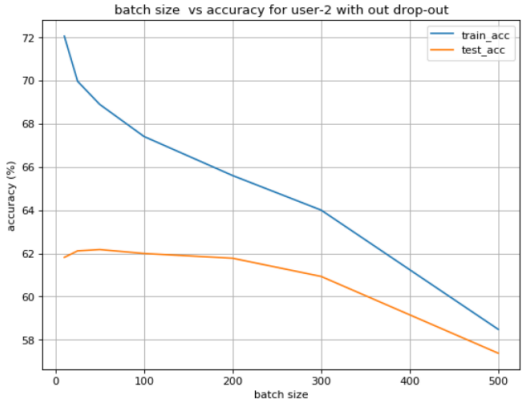

Fig. 5. Batch size vs accuracy for a particular user without dropout

We got a mean score of 68.9 percent on the test data set for the individual users. This was run for 30 users so it was run 30 times and on each case, the average train score was around 75 percent.

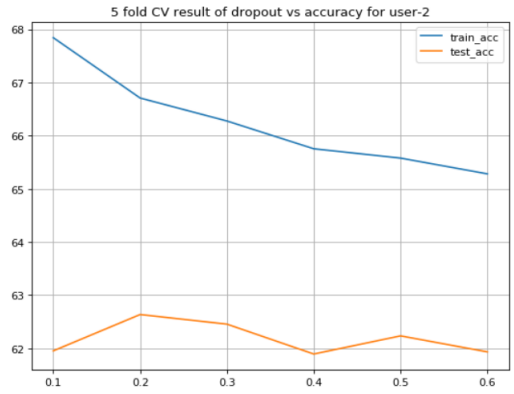

Fig. 6. 5 fold cross validation of dropout vs accuracy for a user

B. Evaluating the user classification model

In the evaluation for the user classification model, we have evaluated for different number of users. Firstly, we ran the model for 2 users. This was then incremented by 1 and we ran eventually for 30 users in all.

The graphs in figures 7 and 8 show us the learning rate (in adam optimizer) vs accuracy and the look back vs accuracy for a particular user. As seen, for the adam optimizer, if we increase the learning rate, then the model initially tends to perform better but the performance worsens as it goes. Also, the variation of look backs is shown and we've chosen the best one.

As expected, when the number of users is small, the model performed exceedingly well. For 2 users, we obtained a test score of about 80 percent. But as the number of users increased, the number of classes also increased and this reduced the test scores.

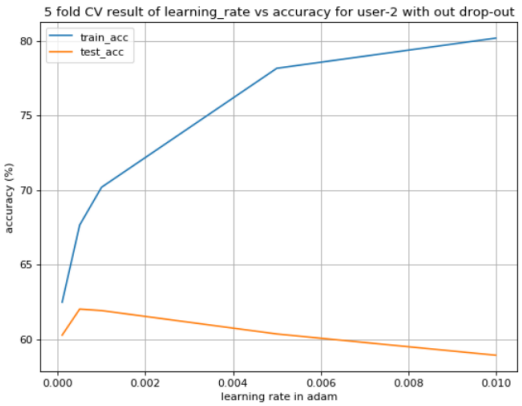

Fig. 7. Learning rate vs accuracy for a given user

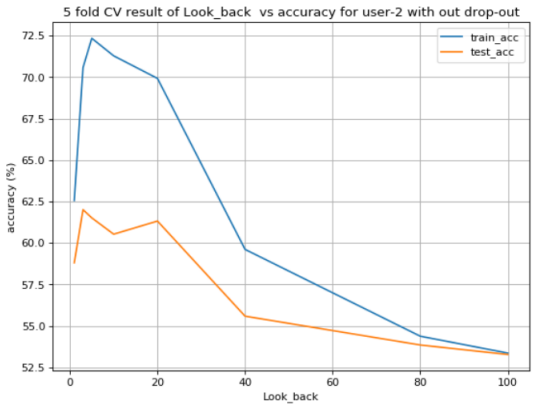

Fig. 8. Look back vs accuracy for a given user

This indicates that in order to build a better model for more number of users, our plan of approach has to change. This is also a good insight that the model complexity has to increase in order to accommodate for more number of users.

XI. LSTM ENCODER-DECODER BASED APPROACH

LSTM networks are recurrent models that have been used for many sequence learning tasks like handwriting recognition, speech recognition, and sentiment analysis. LSTM Encoder-Decoder models have been recently proposed for sequence-to-sequence learning tasks like machine translation. An LSTM-based encoder is used to map an input sequence to a vector representation of fixed dimensionality. The decoder is another LSTM network which uses this vector representation to produce the target sequence.

We propose a Long Short Term Memory Networks based Encoder-Decoder scheme for Anomaly Detection (EncDec-AD) that learns to reconstruct normal time-series behavior, and thereafter uses reconstruction error to detect anomalies. In this case an encoder learns a vector representation of the input time-series and the decoder uses this representation to reconstruct the time-series.

In order to solve our present problem we experimented with two different types of LSTM encoders 1) Sequence - Sequence model:- Here we try to reconstruct the entire time series sequences given the input time series sequences. eg:-[1,2,3,4,5] – [1,2,3,4,5] 2) Encoder-Decoder model:- This is used in case we want the length of the output sequences to be different from that of the length of the input sequences. eg:- [1,2,3,4,5] – [1,2] So in our case we train the LSTM encoders with sequences of features for each user separately with 70 percent of their data for training and in testing we have two folds- it should able to reconstruct the sequences properly if given the user testing data for whom it is trained for and it should not able to reconstruct or it should able to reconstruct less when given other user's data which is not used for training the model. Below are the results obtained using Sequence-Sequence model approach Fig.9 shows us the training and validation accuracy vs number of epochs with validation being done on the same user data which was trained. From this we can observe that we are having high validation accuracy which solved our one of the testing criteria. Fig.10 is about the rejection rate (1- accuracy) vs number of epochs when testing for a different user whose data is not being used for training. Ideally we should expect very high rejection rate in this case but we are achieving it around 60-70 percent. From both the graphs, we can observe that max. validation and rejection accuracy is obtained at number of epochs = 20.

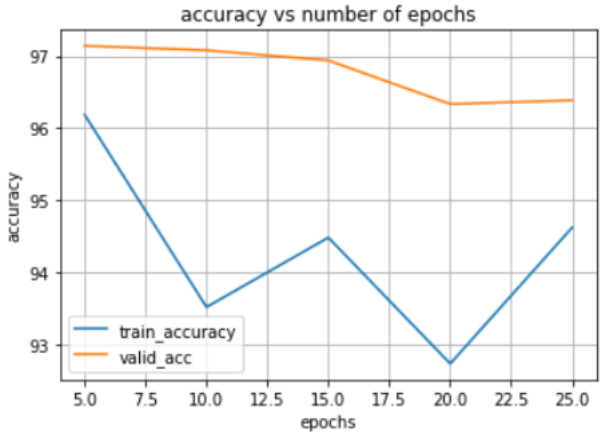

Fig. 9. Accuracy vs the number of epochs

XII. CONCLUSION

We set out to create a user-specific behavior model that understands the 'normal' user behavior using data and we tried to solve two problems with it. Firstly, given a set of time-series sequence data, we attempted to predict the next sequence of events and/or actions of that user. Secondly, given a set of

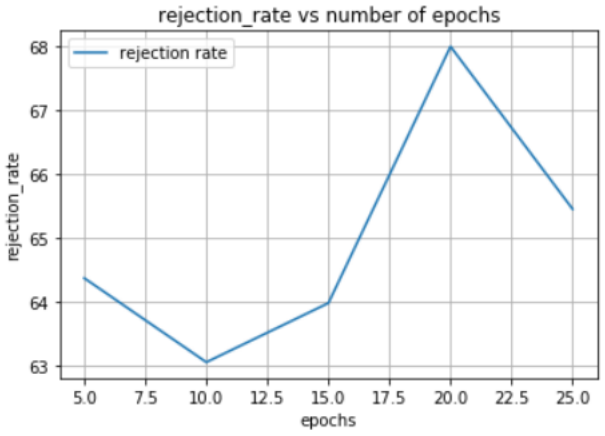

Fig. 10. Rejection rate vs the number of epochs

points, we tried to classify which user it belongs to. In the first case, we achieved a test accuracy of 70 percent for our data on average for users and in the second case, it was changing as the number of users being added into the model changed. A general consensus was that as we added more users, the scores kept decreasing. There is decent scope of improvement in the second problem and we are also able to detect the Anomaly time series sequences with the help of our proposed LSTM Encoder-Decoder based approach.

XIII. ACKNOWLEDGEMENTS

We would like to express our gratitude to our faculty advisor Prof. Carlos Morato for his guidance through our project and through the course. We'd also like to express sincere thanks to our internship advisor Mr. Ramesh Gupta whose valuable guidance has helped us in this project as well. Lastly, we'd like to thank WPI for providing us this wonderful platform.

REFERENCES

[1] Unsupervised Clickstream Clustering for User Behavior Analysis, Santa Barbara, CA, 93106.
[2] User Behavior Modeling with Large-Scale Graph Analysis.
[3] Clickstream User Behavior Models.
[4] Understanding Consumer Behavior with Recurrent Neural Networks.
[5] Detecting Fraudulent Behavior Using Recurrent Neural Networks
[6] Long Short Term Memory Networks for Anomaly Detection in Time Series
[7] Sequence to Sequence Learning with Neural Networks