VOL9 NO.1 March 2020 - ISSN 2233 – 1859

Operations Research Society in Bosnia and Herzegovina

Southeast Europe Journal of Soft Computing

Available online: http://scjournal.ius.edu.ba

IUS Soft Computing Research Group

Implementation Of Long Short-Term Memory (LSTM) For User Authentication Based On Keystroke Dynamics

Admir Ferhatovic[1*], Ali Abd Almisreb[1], Sherzod Turaev[2], Mohammed A. Saleh[3]

[1]Faculty of Engineering and Natural Sciences, International University of Sarajevo International University of Sarajevo, HrasnickaCesta 15, Ilidža 71210 Sarajevo, Bosnia and Herzegovina.

[2]Department of Computer Science & Software Engineering, College of Information Technology, United Arab Emirates University.

[3]School of Computing and Creative Media, University College of Technology Sarawak, Sarawak, Malaysia

admir.ferhatovic@gmail.com

Article Info

Article history:

Article received on 10 January 2020

Received in revised form 1 February2020

ABSTRACT: The paper reviews the usage, development and implementation of keystroke dynamics as a viable authentication system. AI methods are advancing, but we are still lacking biometric authentication systems in modern software which is being used daily. This paper shows the usage of long short-term memory layers for solving problems like keystroke dynamics and efficiently shows that with modern hardware, training and maintaining a small model is not taxing on the resources, as it may have been.

Keywords:

LSTM, AI, Biometric, Keystroke dynamic

1. INTRODUCTION

In this paper I want to provide a clear overview of a keystroke dynamics classifier implementation using long-short-term-memory layers and present an idea about how many test samples per person are needed for a secure and maintainable system [1]. This way I want to encourage the usage of keystroke dynamics as an authentication system in everyday applications [2], not to replace the simple password system, but to act as a support for it, this way maintaining a high level of security without needing the user to take additional actions which distract him from his everyday tasks [3]. Whether an authentication system should later in this paper, after we see the potential accuracy of such a system. One of the more common malware types that casual users get infected are keyloggers [4]. Once a password is compromised using a keylogger, the attacker will either use it directly and type it or get an authentication token using a request to an authentication service if it exists. Either way, if the service required a typing rhythm vector along the password, the account would be secured from this kind of attacker [5]. One could argue, if the keylogger records all keys typed, it could also record the typing rhythm. That is right, but it is not common and tedious work for the attacker, and this method would certainly make it harder to steal accounts on a large scale [6].

Another use case of keystroke recognition is to protect policies from services which forbid account sharing [7]. Account sharing has been an issue for some time now, and nothing has been done to try to counter it. Having users typing rhythms would finally help detect if an account is being shared and therefore help recognize if a policy is being broken. For this purpose, a publicly available dataset was used which contains 400 samples of time series vectors for 51 users [8].

II LITERATURE REVIEW

Most literature regarding the topic keystroke dynamics does not concern itself with the methods and techniques to implement such a system, but more with the necessary proof that keystroke dynamics is a valid biometric system that can stand its ground. In [2], it is argued that a typing rhythm is a behavioral trait instead of being a physiological characteristic, making it suitable as a biometric method. In [9], results from experiments have been reported that support the idea of typing rhythm being a unique behavioral pattern for every individual, as several tests have been performed and the results were promising. After analyzing the data that they collected, they reported a false alarm rate of only 5.5%, which is astonishing given the fact that these experiments have been performed in 1980. Other literature such as [10], concerns itself more with data acquisition and the advantages of keystroke dynamics in that regard. Their main argument for supporting keystroke dynamics as a biometric system is that data can be collected without the user being aware or having to perform anything. As users are typing, which is the main activity for most people that use a computer, their keystroke timings can be automatically recorded. Whilst, the paper [11] argues the viability of using keystroke dynamics as a system of authentication on its own versus using it only for additional verification. The main concern of this paper is providing proof of a viable neural network approach and its example usage in practice. In [1], it is stated that the main weakness of such an approach is having to retrain the network when a new user is added, but I want to show that this is not a huge burden anymore and can be easily overcome with modern technology and smart methods.

III DATASET

As proposed, the most suitable way to represent a typing rhythm for a given person would be a, so called time series. A time series is simply put, a vector containing time measurements of certain actions. In the case of keystroke dynamics, the possible actions can be broken down into key presses, and smaller components of key presses, and transition times between these components.

Basically, all these actions have some latency between them. A latency from action A to action B, therefore. Therefore, we can mark these latency using letters corresponding to actions, in this example A and B, to form a digraph AB. Having a classification of possible actions into digraphs makes our work easier when recording and reading latencies from existing datasets.

In most research the following actions are differentiated with their corresponding digraph symbols. We have the Key-Up timing (**KU**) which represents the time a key is being released and we have Key-Down timing (**KD**) which represents the time a key is being pressed. Next is Hold/Dwell time (**HT**) which represents the time between key release and a new key press. These are the at least necessary latencies. We can also derive other latencies from these such as Press-Press (**PP**) time between two key presses, Release-Release (**RR**) time between two key releases and Release-Press (**RP**) the time between the last key release and new key press.

Now that we cleared that our input is in the form of a simple vector containing floating point numbers, let us talk about its use in neural networks and its difficulties. Having a vector as input is probably the dream of anyone who ever put their hands onto data science. It is the lowest level and purest form an input can have. Usually, the hardest part is transforming your input into an input that resembles numbers, but here we already have that. Here it is clear, that keystroke dynamics as a form of biometrics is perfectly suited as a problem which can be solved using AI methods, ranging from techniques like SVMs to neural networks. Now, the only shortcoming is that depending on the password or text that users write the input vector is obviously different in length. But this is only a mild problem because there are simple techniques to overcome it like zero-padding, RNN layers and recursive NNs.

The dataset used throughout this paper is taken from https://www.cs.cmu.edu/~keystroke/[8]. It contains 51 subjects with 400 time series per subject. All subjects used the same password (*.tie5Roanl*). For each key pressed a timing is recorded in seconds. Also, for each key transition two times are recorded, the keydown-keydown time and the keyup-keydown time. Each record also has a session index which differentiates during which session a record was made. So, in total there are 31 input attributes and 51 possible outputs.

subject	sessionIn	rep	H.period	DD.period	UD.period	H.t	DD.t.i
s002	1	1	0.1491	0.3979	0.2488	0.1069	0.1674
s002	1	2	0.1111	0.3451	0.234	0.0694	0.1283
s002	1	3	0.1328	0.2072	0.0744	0.0731	0.1291
s002	1	4	0.1291	0.2515	0.1224	0.1059	0.2495
s002	1	5	0.1249	0.2317	0.1068	0.0895	0.1676
s002	1	6	0.1394	0.2343	0.0949	0.0813	0.1299
s002	1	7	0.1064	0.2069	0.1005	0.0866	0.1368

Figure 1: Example of dataset records

In Fig 1. we can see different records containing the subject number, the session number, the repetition number and digraphs for Hold Time (**H**) of period (.), Key-Down (**DD**) of period, Key-Up (**UD**) of period and Hold, Down time of T (t).

IV METHODS

There exists a variety of methods that can be used to implement classification of users through typing rhythms. One of the more successful ones is the usage of a Support Vector Machine, as in [4]. But the obvious shortcoming is that given N users, you need N support vectors to classify each time series. I am in favor of using neural networks, but as mentioned before neural networks also have a similar shortcoming. Once you train your neural network for a fixed number of possible users (possible outputs), you need to retrain it once another user is added. In this paper, I want to show that this is nothing, but a small hindrance with modern hardware and a creative approach, but first let us talk about the obvious advantage of neural networks in comparison to SVMs, neural networks have a fixed sized input, therefore they scale better.

They scale better only if you keep being smart about it. Adding users adds also outputs to the neural network, which increases the training cost, it takes longer to train it to a high percentage accuracy. Ironically, the idea to handle this comes as an analogy from SVMs. With SVM you keep adding support vectors for each possible class, in our case each new user.

To leverage the advantage of a neural network, the idea is to separate users into clusters of some fixed size and have a separate neural network for each cluster. The cluster size should be determined by the cost needed of retraining. One could say a compromise. But, after a cluster is full of users, it never needs to be retrained again and is in its final form, as each new user will be assigned to a cluster that has not reached its maximum size. In the use case of authentication, what allows for this to be efficient, is the fact that at user authentication we know beforehand as which user the person is authenticating so we can select the appropriate cluster and determine if the user is the user or is an imposter. Of course, we would like to know who the imposter is, but in case of authentication it is not a priority and can be determined at a later point, after the input is ran through all clusters. And of course, with more powerful resources, recognizing a person by keystroke dynamics is not a problem for a clustered system as the input can be ran concurrently through all clusters.

Now as for the details of what type of neural network and what layers to use.For a problem where the input is in the format of a time-series vector, using a Long short-term Memory model comes as a natural choice. [12] The hard part is to determine the number of layers and memory cells per layer in our model. Considering our small sample size, we need to constrict ourselves to as few layers as possible. In the model I implemented I used one LSTM layer for input, two LSTM layers as hidden layers and a dense 51 output layer. After testing, 100 memory cells per LSTM layer seems optimal for our sample size. The number 100 is calculated using a formula found in [13]. The model can be seen in *Fig 2.*

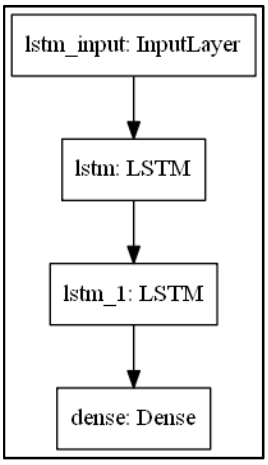

Figure 2: Visual representation of the model

V RESULTS

Let us talk about the goals of the tests and the hardware used for training. The goal of the tests is to show that it is not expensive anymore to train and retrain neural network models. When I say expensive, I am talking about the computational cost and the most important cost of all which is time needed to retrain a model.

Showing this, I hope to make usage of such models a routine in software development. Now about the other important cost, which is hardware. I did the training on a Nvidia GTX 1050ti[1]. For those unaccustomed with graphics card, this is a graphics card in the lower end of the user spectrum and therefore cheap, but still powerful enough for tasks such as e-sports oriented gaming and as we will see training of simpler neural network models. It allows us to train the model with a batch size of 2996, which is impressive for a cheaper graphics card.

For test, we implemented the model and ran the training using Keras[14], which is a deep learning library for python running on TensforFlow [15]. [1] We ran the training with different numbers of epochs in different instances on purpose, as to see how similar the results of the training process are between the same number of epochs. We started out with 500 epochs and the results can be seen in *Fig 3.*

[1] Nvidia GTX 1050ti - https://www.geforce.com/hardware/desktop-gpus/geforce-gtx-1050-ti/specifications

We see that it peaks at an accuracy of over 80% with the training data. This is of course not the accuracy we are looking for, but it is impressive considering that this was done in about 15 minutes with a relatively cheap graphics card.

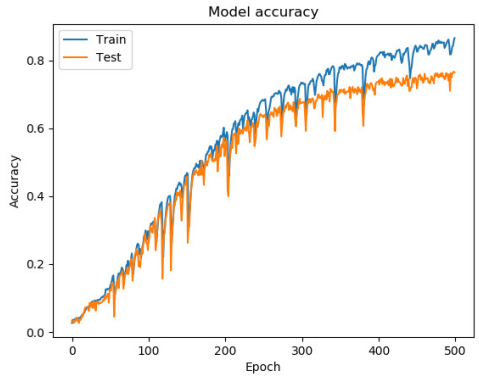

Figure 3: Training process over 500 epochs

Imagine doing it with a GPU intended for supervised learning. It would be done in under a minute, which is extremely practical and acceptable in modern multi-user software systems, which I will talk about again after we analyze further training instances.

In Fig 4. we can see the process happening over 800 epochs. As expected, nothing peculiar happens, the model gets more accurate for the test and train data, with the main importance being that the model reaches 80% accuracy for the training data. Still not accurate enough, but still impressive because it took around 20 minutes.

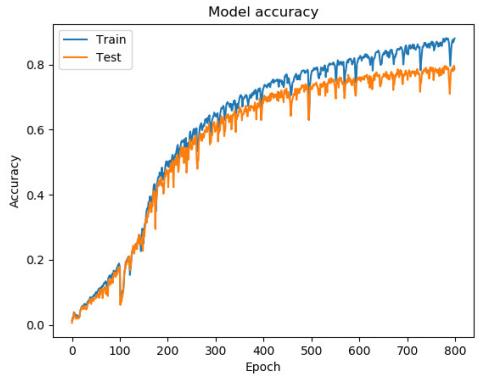

Figure 4: Training process over 800 epochs

In *Fig 5.* we see something important happening and that is overfitting of the model. Training of 1000 epochs was planned here, but the model started to overfit around 900 epochs. This means that one needs to be careful to not choose a fixed number of epochs when retraining such models in practice. The recommendation is to have some optimal fixed number of epochs, but to continue training in case of low accuracy, because it can be due to overfitting.

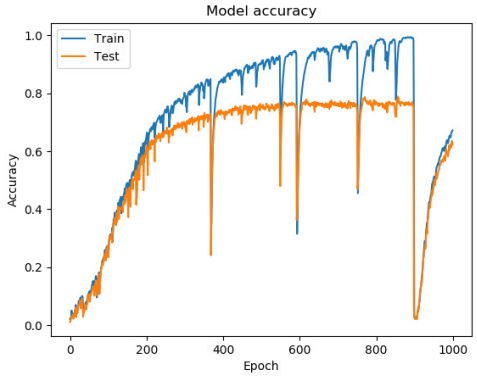

Figure 5: Training process over 1000 epochs

In the last training instance, which can be seen in *Fig 6.*the training was restarted with a goal of 2000 epochs.

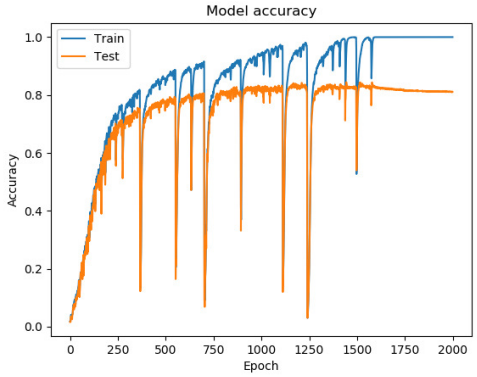

Figure 6: Training process over 2000 epochs

The model reached a 100% accuracy level on the training data, after 1500 epochs. We can also see that overfitting happened a couple of times, the most critical happened about the 1250st epoch, but the model recovers quickly from overfitting.

Now let us talk about the practical implications of the results. The last training instance took an hour on my configuration with the GTX 1050ti. Take into consideration that the model reached 100% accuracy around 1500 epochs and that we can put a stopping condition in the training process, let us assume that reaching 100% accuracy on training data takes about ~45 minutes. This is astonishing for a GTX 1050ti. The model has an accuracy of 80% on test data, which is not the best, but is acceptable considering the lack of data in the dataset used and the size of the model itself. Now, imagine using such a model in a multi-user application, more specifically in an e-commerce web application. Whoever has had any experience with e-commerce platforms, knows that they have certain times in a day, where the number of active users is minimal. This period usually lasts a couple of hours and is used by developers to run automatized maintenance tasks.

If one would be to use the strategy proposed earlier, which consists of clusters, all having a model themselves. One could use the period to retrain clusters which need retraining to keep their accuracy high. This would cost almost nothing, because the chance of errors happening is low at the specified time and retraining models does not result in a downtime for the e-commerce system.

VI CONCLUSION

In this paper, we showed the dataset and the implementation of keystroke dynamics in the area of deep learning. LSTM model is implemented for this purpose using Tensorflow and Keras libraries. Nvidia GTX 1050ti GPU is used for training and testing. For further development, the LSTM performance could be tested over several datasets to validate the reliability of the obtained results

REFERENCES

F. Chollet, Deep Learning mit Python und Keras, mitp, 2018.

F. Monrose and A. Rubin, "Authentication via Keystroke Dynamics".

Y. Zhong, Y. Deng and A. K. Jain, Keystroke Dynamics for User Authentication, 2012.

Y. Sang, Y. Sang and P. Fan, "Novel Impostors Detection in Keystroke Dynamics by," Ishikawa, Japan.

S. Bleha, C. Slivinsky and B. Hussien, Computer-Access Security Systems, 1990.

M. Karnana, M. Akila and N. Krishnaraj, "Biometric personal authentication using keystroke dynamics: A review," *Applied Soft Computing,* 2009.

S. Cho, C. Han, D. H. Han and H.-I. Kim, "Web-Based Keystroke Dynamics Identity," *Journal of Organizational Computing and,* 2014.

"cs.emu.edu," [Online]. Available: https://www.cs.cmu.edu/~keystroke/. [Accessed 12 October 2019].

W. L. S. P. a. S. N. R. Gaines, "Authentication by Keystroke Timing: some pre-eliminary results," 1980.

S. P. Banerjee and D. L. Woodard, "Biometric Authentication and Identification using Keystroke," *Journal of Pattern Recognition Research 7,* pp. 116-139, 212.

J. Ilonen, "Keystroke dynamics," Lappeenranta University of Technology, Lappeenranta.

S. Hochreiter and S. Jürgen, "Long Short-Term Memory," *MIT Press Journals,* 1997.

J. Brownlee, Long Short-Term Memory Networks With Python, Machine Learning Mastery.

"Keras," [Online]. Available: https://keras.io/.

"TensorFlow," [Online]. Available: https://www.tensorflow.org/.

F. Monrose, M. Reiter K. and W. Susanne, Password hardening based on keystroke dynamics, Springer-Verlag, 2001.

K. S. Killourhy and R. A. Maxion, Comparing Anomaly-Detection Algorithms for Keystroke Dynamics, 2009.

M. S. Obaidat and B. Sadoun, Verification of Computer Users Using Keystroke Dynamics, 1997.

R. Giot, M. El-Abed and C. Rosenberger, GREYC Keystroke: a Benchmark, 2009.

E. Yu and S. Cho, "Keystroke dynamics identity," *Computers & Security,* 2004.

J. A. Robinson, V. M. Liang, J. A. M. Chambers and C. L. MacKenzie, Computer User Verification Using, 1998.

S. Haider, A. Abbas and A. K. Zaidi, A Multi-Technique Approach for User Identification through K:eystroke Dynamics, 2000.

H. Crawford, Keystroke Dynamics: Characteristics and Opportunities, University of Glasgow, 2010.