Available online at www.sciencedirect.com

SciVerse ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

A second look at the performance of neural networks for keystroke dynamics using a publicly available dataset

Yasin Uzun [a,*], Kemal Bicakci [b]

[a] Middle East Technical University, Ankara, Turkey
[b] TOBB University of Economics and Technology, Ankara, Turkey

ARTICLE INFO

Article history:
Received 22 December 2011
Received in revised form
9 March 2012
Accepted 9 April 2012

Keywords:
Biometrics
Keystroke dynamics
Verification
Neural networks
Backpropagation

ABSTRACT

Keystroke Dynamics, which is a biometric characteristic that depends on typing style of users, could be a viable alternative or a complementary technique for user authentication if tolerable error rates are achieved. Most of the earlier studies on Keystroke Dynamics were conducted with irreproducible evaluation conditions therefore comparing their experimental results are difficult, if not impossible. One of the few exceptions is the work done by Killourhy and Maxion, which made a dataset publicly available, developed a repeatable evaluation procedure and evaluated the performance of different methods using the same methodology. In their study, the error rate of neural networks was found to be one of the worst-performing. In this study, we have a second look at the performance of neural networks using the evaluation procedure and dataset same as in Killourhy and Maxion's work. We find that performance of artificial neural networks can outperform all other methods by using negative examples. We conduct comparative tests of different algorithms for training neural networks and achieve an equal error rate of 7.73% with Levenberg–Marquardt backpropagation network, which is better than equal error rate of the best-performing method in Killourhy and Maxion's work.

1. Introduction

The fact that people show unique typing characteristics and the opportunity of using this information for security purposes was first discovered by telegraph operators (Bryan and Harter, 1994). Applying the same idea to computer keyboards provide the basis for Keystroke Dynamics, which is defined as "the process of analyzing the way user types on a terminal by monitoring the keyboard inputs" (Monrose and Rubin, 2000). Keystroke Dynamics provides a behavioral biometric feature which identifies individuals from their typing style. It depends on the observation that individuals show unique characteristics when they type on a keyboard.

A typical Keystroke Dynamics verification system works as follows. For a new user to be enrolled in the system, a profile is created from a set of typing samples collected from that particular user. There are different types of data available including typing pressure on the keys, finger temperature, etc. But because of practical reasons, the two most commonly used measurements are keystroke latency, which is the amount of time that passes between consecutive key press events and keystroke duration, which is the amount of time each key is being pressed. When there is a new access attempt to the system, the relevant keystroke data is compared to the stored user profile. The access is verified if the discrepancy between the data and profile is below the selected threshold, and rejected otherwise. Although it is possible to set a global

* Corresponding author.
E-mail addresses: yasin@ii.metu.edu.tr (Y. Uzun), bicakci@etu.edu.tr (K. Bicakci).
0167-4048/$ – see front matter © 2012 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cose.2012.04.002

threshold for all users, most implementations choose to specify a different optimized threshold value for each user, in order to adapt the difference in user behaviors.

Performance of a keystroke based identity verifier is generally evaluated as follows. Firstly, a set of typing samples is collected from users. For each user, a subset of typing samples is used to construct a profile and the remaining samples are used for testing. Additional typing samples may be collected to simulate impostor login attempts. For performance metrics related to algorithms, FMR (False Match Rate) – the ratio of wrongfully accepted impostor attempts and FNMR (False Nonmatch Rate) – the ratio of rejected legal attempts of an authorized user could be used (Jain et al., 2004) (We note that although the terms FAR (False Acceptance Rate) and FRR (False Rejection Rate) refer to error rates for systems, they are also used for describing error rates related to algorithms in many studies in the literature). Even computing both FNR and FNMR together may sometimes be insufficient to make reliable comparisons between performances of different systems since one method may have lower FMR while the other has lower FNMR or vice versa. The performance metric that overcomes this problem is EER (Equal Error Rate), which is measured by adjusting the acceptance threshold value so that FMR value is equalized to FNMR and is measured using "Receiver Operating Characteristics" (Green and Swets, 1996).

There are various methods proposed in the literature as keystroke identity verifiers. A long-standing problem in Keystroke Dynamics is discovering the best-performing method that achieves the lowest error rates. Although performance of many different methods were reported in the literature, as Killourhy and Maxion pointed out (2009), it is usually not possible to compare these different experimental results because of (i) the differences in the features used to train and test verifiers; (ii) diversity of the evaluation conditions (e.g., length of typing samples, number of typing repetitions, outlier-handling procedures, number of authentication attempts, the update of the model over time); (iii) inconsistent types of performance results reported (e.g., EER, FMR when FNMR = 0, etc.).

Motivated by the difficulty of comparing different verification methods using the evaluation results reported in the literature, Killourhy and Maxion decided to collect a dataset to be made publicly available and test an exhaustive list of methods using this dataset under the same conditions. They encourage other researchers to evaluate the performance of other methods using their dataset. With the same dataset and by applying the same methodology, other results can be compared with validity using the results in their work (2009). The equal error rates obtained using neural networks and the best method in Killourhy and Maxion's work are given in Table 1. It is interesting to see that EER value reported in the study for standard neural network is 82.8%, which is worse than even a random verifier with EER of 50%. Although the error rate for auto-associative neural network is much better (16.1%), it stills performs poorer than other types of detectors.

Table 1 – Selected equal error rates from Killourhy and Maxion's work.

Detector	Average EER (%)
The best detector (i.e., Manhattan (scaled))	9.6
Neural network (auto-assoc.)	16.1
Neural network (standard)	82.8

Because of the abnormally poor performance of neural networks reported in previous work, our aim in this study is to have a second look at their performance for Keystroke Dynamics. We investigate whether the high error rate reported was due to incompatibility of neural networks for the problem of Keystroke Dynamics or its imperfect usage by the earlier work. After performing a detailed study, we find that by taking advantage of incorporating negative examples into the training, the performance of artificial neural networks can be significantly improved. We conduct comparative tests of different algorithms for training neural networks and achieve an equal error rate of 7.73% by using Levenberg–Marquardt backpropagation network, which is better than 9.6% - the equal error rate of the best-performing method in Killourhy and Maxion's work (2009).[1,2]

The rest of the paper is organized as follows. Section 2 overviews the related work. Section 3 introduces the training algorithms for backpropagation neural networks used in our study. Section 4 presents and discusses our experimental results. Section 5 concludes the paper.

2. Related work

Verification of user identities based on their keystroke profiles has been extensively studied in the last two decades. Most proposed verification algorithms are based on statistical, neural network and other machine learning methods. In the statistical approach, the test samples are compared with the reference set of training samples while the neural network methods build a prediction model using the training samples. Since our aim in this paper is to gain new knowledge about the use of neural networks for Keystroke Dynamics, in this section our main focus is on earlier work which used neural network methods. Nevertheless, for the sake of completeness we also present the major results regarding the keystroke analysis using other methods (Killourhy and Maxion's work introduced in the previous section is discussed further in Section 4). A more general survey on this topic can be found elsewhere (e.g., Shanmugapriya and Padmavathi, 2009).

One of the initial efforts in the area was made by Bleha et al. (1990) using Bayes classification in which 32 users participated in the experiments and key latency was used as a metric for user validation. Among the participants, 10 were valid users of the system and 22 of them tried to imitate the valid users. The result of the study was promising. Rejection rate of valid users (FNMR) was 8.1% and acceptance rate of

[1] In line with the practice of shared data promoted in the mentioned study, we make the source code of all algorithms used in our tests available (Uzun, 2011).

[2] We note that there is another Keystroke Dynamics benchmark dataset developed by Giot et al. (2009a, b) which is available via request. There is also an application available online to extend this dataset.

invalid users (FMR) was 2.1%. The authors noted that the main reason of the errors was inconsistent typing style of inexperienced users.

A more complicated identity verifier was developed by Joyce and Gupta (1990), which recognize users by their "signatures", consisting of the typing latencies of four elements: user name, password, name, and surname. In their method, a similarity vector is calculated by comparing the latencies in training and test signatures. The user is positively identified if the elements in the similarity vector lie under specific thresholds, which is determined by mean and standard deviations of the reference (training) signature. Among 975 trials in total, FMR was reported as 0.25% and FNMR as 16.6%.

Obaidat and Macchairolo (1993) analyzed Keystroke Dynamics as a classification problem rather than verification. In data collection phase, 6 participants typed a predefined sequence of characters for 20 times and keystroke latency was used as the feature vector, which consists of 15 feature elements. Using the collected raw data, a hybrid sum-of-products neural network was trained with 15 inputs, 4 hidden units and 3 outputs. The classification system was designed to assign each test pattern to a predefined class and the reported classification success rate was 97.8%.

K-means algorithm was used for Keystroke Dynamics verification in the study of Kang et al. (2007). In the experiments, 150 samples (75 genuine and 75 impostor attempts) were collected from 21 participants. For each user, the authors divided the training sample set into three groups using k-means clustering. In testing phase, for an incoming sample, the minimum Euclidean distance between the sample and the nearest of the three cluster centers was accepted as the distance between the sample and the profile, to be used to determine the likeliness of the user as being genuine. The authors compared fixed, growing and moving window approaches for training the verifier and for these approaches they obtained the equal error rates of 4.8%, 3.8% and 3.8%, respectively.

An example of machine learning approach for keystroke verification problem is the work of Fan et al. (2004), p. 666–669, which employed Support Vector Machines as a solution. In this study, the authors recruited 10 subjects in total, who typed an alphabetic password of 10 characters and a numeric password of 8 characters and each user was imitated by five other users. Using combination of two passwords, the authors employed both one-class and two-class SVM to recognize the users. The error rates were 2% FMR and 10% FNMR, 10% FMR and 10% FNMR for one-class and two-class SVM respectively.

Another support vector machine based keystroke verification algorithm was developed by Giot et al. (2009). The participants typed the phrase "*greyc laboratory*" six times in the experiments. In the enrollment phase, two-class support vector machine is trained for each user using 5 training samples. The output of the machine was either $+1$ or -1 depending on the owner of the sample (genuine or impostor). The authors investigated two different issues in their study: the effect of keyboard change between enrollment and verification and the comparison of support vector machines with statistical, distance based and rhythm based algorithms. It was found that keyboard change had not led to significant degradation in verification accuracy. In the experiment results, the proposed SVM method outperformed all other methods with EER values varying between 10.30% and 11.76% for different keyboard combinations.

Besides other machine learning and statistical methods, many researchers have utilized different kinds of artificial neural networks for Keystroke Dynamics. One of the early efforts was the work of Cho et al. (2000), in which the authors used auto-associative neural network scheme, a kind of backpropagation neural network. In their experiments, a total of 25 subjects were asked to type passwords of their choice with 7 characters. Each subject typed his/her password 150–400 times, from which latest 75 samples were reserved for the test and their typing styles were imitated by 15 impostors for 5 times. The auto-associative multilayer perceptron, which is a three layer feed forward backpropagation network was trained for each user. The authors reported that they had achieved 1.0% FMR with 0 FNMR.

ARTMAP-FD neural network, an extension of the Fuzzy ARTMAP-also a type of neural network was proposed as a solution for keystroke based authentication by Loy et al. (2007). The collected dataset consists of 100 samples collected from 10 participants (10 from each). In their tests, using only keystroke latency, the best average EER (14.94%) was achieved using ARTMAP-FD. When the feature dataset was enlarged with keystroke pressure, the average error is decreased to 11.78%.

Lee and Cho trained a novelty detector for learning vector quantization network, another kind of neural network (2007). In their experiments, 21 users simulated legitimate users with individual passwords and 15 participants simulated impostor login attempts. For each user, 50 valid (positive) and 5 impostor (negative) keystroke timing vectors were used for training. For testing, 75 normal attempts and 70 impostor attempts were used. Initially, the authors trained the detectors using positive-only data and then negative examples were used together with positive examples. The average EER were reported as 0.59% when only positive examples were used. When negative examples (impostor attempts) were also used for training, the average EER was reduced to 0.43%.

In the studies described above, the authors proposed their approaches and presented experimental results independently. In contrast, Obaidat and Sadoun performed comprehensive tests for statistical and neural network approaches (1997). 15 users have participated in their experiments, in which each subject had a special user ID with different lengths having an average size of 7 characters. For eight weeks, each user provided 15 sequences daily. Each user also provided 15 samples for each of the other subjects. The dataset was partitioned into two halves: training and testing. Like test set, training set also includes impostor samples. The authors concluded that neural network approaches are superior to statistical approaches and achieve quite promising results (zero for both FMR and FNMR) when impostor samples are available during training.

Similar to Obaidat and Sadoun's work (1997) Haider et al. conducted a comparative study for fuzzy, statistical and neural network methods (2000). In their experiments, each user selected a password up to 7 characters length, typed it for 15 times and keystroke latency times were recorded. In the

test phase, users were given two chances for entry. The trained feed forward backpropagation neural network has 6 input nodes (size of the feature vector), 4 hidden nodes and a single output node. In the training phase, the output is set to 1.0 and the updated weight matrices are recorded as the profile. In the test phase, when a new typing sample arrives, the network is initialized with the recorded weight matrices and is run with the sample vector as the input. If the calculated output is close to the desired value (within thresholds) the sample is accepted as legitimate, otherwise it is accepted as invalid. The neural network has provided an accuracy of 20% FNMR and 22% FRR – worst among the three detectors tested. But when it is combined with fuzzy and statistical methods, FMR and FNMR were reduced to 2% and 6%, respectively.

Other than regular PC keyboards, keystroke verification can also be applied in cellular phones, as shown in the study by Clarke and Furnell (2007). In this study, to simulate mobile phone environment, the keypad of a mobile phone was connected to a PC in place of a keyboard. A total of 32 users were asked to type two types of numbers: a four digit PIN and an eleven digit telephone number. For each of the participants, 20 samples were used for training and 10 samples were used for testing. The classification tests were performed by comparing the samples of one valid user and the samples of other users acting as impostors. Three types of neural networks were employed in the experiments: Feed forward multilayer perceptron (FF-MLP), Radial Basis Function Network and Generalized Regression Neural Network (GRNN). For 4-digit PIN, GRNN performed the best with 13.3% EER while FF-MLP was the best for 11-digit phone number with 12.8% EER. In the second test, each of 30 subjects was asked to write 30 text messages using the numeric keypad. The subjects had to press one or more times to each key for a single alphabetical character. In this test, there were no predefined inputs and the subjects have written arbitrary messages. For alphabetical input, the best average EER (17.9%) was achieved with FF-MLP with gradual training, which is a neural network technique that utilizes changing number of epochs to improve generalization (Napier et al., 1995).

As discussed above, there have been numerous attempts to employ neural networks and other methods for keystroke dynamics. The summary of the major results is provided in Table 2. However, as we have discussed in the previous section it is not possible to make a sound comparison using these results because the tests are performed with different datasets and under various different assumptions. Specifically, for the use of neural networks in Keystroke Dynamics, making a performance comparison of different algorithms with a reproducible common methodology still remains an open problem.

3. Backpropagation neural networks

Neural networks are artificial learning systems modeled by simulating human brain. An artificial neural network consists of consecutive layers, which include self-computing neurons. Each neuron accepts a set of inputs; computes a weighted sum and applies a transfer function to the sum. The output is transmitted to other neurons or to the environment.

Learning problem in training neural networks can be considered as finding the optimal weight values that will minimize the error, which is calculated as in Equation (1) where t_u represents the target that the unit u is expected to show, o_u represents the output for the unit u. The squared sum of this difference gives the error for a single sample.

$$E = \sum (o_u - t_u)^2 \tag{1}$$

In the backpropagation algorithm, the error rate is computed for each output neuron first. For each output neuron, the gradient, which is the vector of partial derivatives with respect to weight values for inputs, is computed as in (2).

$$\Delta E = (\delta E/\delta w_1, \delta E/\delta w_2, \ldots, \delta E/\delta w_n) \tag{2}$$

Then for each output neuron, each input weight value is updated by adding the value Δw_i computed as in Equation (3), where Δw_i represents the update value for the ith input for the neuron, $\delta E/\delta w_i$ is the error gradient for the same input and μ is the learning rate.

$$\Delta w_i = \mu \times \delta E/\delta w_i \tag{3}$$

The process is repeated for each neuron in the output layer. Then, the update operation is carried for previous layers in reverse order. The crucial point here is that the activation function must be differentiable in order to be able to calculate the gradient.

Table 2 – Summary of the major experimental results of earlier work for keystroke dynamics.

Source study	Method	Text length	FMR (%)	FNMR (%)
Bleha et al. (1990)	Bayes classification	31	2.1	8.1
Joyce and Gupta (1990)	Statistical comparison	N/A	0.25	16.6
Obaidat and Macchairolo (1993)	Hybrid sum-of-products	15	97.8% classification success	
Obaidat and Sadoun (1997)	Backpropagation	7	0	0
Cho et al. (2000)	Auto-associative	7	1.0	0
Haider et al. (2000)	Backpropagation	7	22	20
Fan et al. (2004), p. 666–669	Support vector machine	10/8	2/10	10/10
Lee and Cho (2007)	1-LVQ	6–10	0.43	0.43
Loy et al. (2007)	ARTMAP-FD	N/A	14.94	14.94
Cho et al. (2000)	K-means	7–9	3.8	3.8
Clarke and Furnell (2007)	Backpropagation	11	12.8	12.8
Giot et al. (2009)	Support vector machine	16	10.30	10.30

There are different choices for training a neural network to minimize the error rate. In our study, we choose 12 different training algorithms proposed in the literature for back-propagation neural networks. In the following subsections we give brief descriptions of the algorithms. The detailed explanation of these algorithms can be found in (Beale et al., 2010).

3.1. Gradient descent backpropagation

Gradient descent backpropagation is the batch steepest descent algorithm. By using backpropagation, the error is passed through layers in reverse mode. At each layer, the gradient of the error is computed and for each weight value of each neuron the update rule is given in Equation (4) where Δw_{ij} is the update value for jth input of ith neuron, g_{ij} is the gradient for the input.

$$\Delta w_{ij} = \mu \times g_{ij} \tag{4}$$

3.2. Gradient descent with momentum backpropagations

In this algorithm, gradient descent is computed at each iteration. The weights are updated as in Equation (5) Δw_n represents latest weight update, Δw_{n-1} represents previous weight update, α is momentum constant and g is the gradient.

$$\Delta w_n = \alpha \times \Delta w_{n-1} + \mu \times (1-\alpha) \times g \tag{5}$$

This method makes a compromise between the latest gradient and previous search direction, therefore reducing the probability of getting stuck to local minima. Momentum constant takes value between 0 and 1. For the values close to 0, the method approximates gradient descent algorithm. For the constant values close to 1, the effect of the latest update gradually decreases, thereby reduces the training speed.

3.3. Gradient descent with adaptive learning rate backpropagations

In this algorithm, gradient descent is computed at each iteration. The weights are updated as in Equation (6), where Δw represents latest weight update.

$$\Delta w = \mu \times g \tag{6}$$

The learning rate changes adaptively. If error decreases toward the goal, learning rate is increased. If it increases more than the specified threshold, the update is canceled and learning rate is decreased.

3.4. Gradient descent with momentum and adaptive learning rate backpropagation

In this algorithm, gradient descent is computed at each iteration. The weights are updated as in Equation (7).

$$\Delta w_n = \alpha \times \Delta w_{n-1} + \mu \times \alpha \times g \tag{7}$$

If error decreases toward the goal, learning rate is increased. If it increases more than the specified threshold, the update is canceled and learning rate is decreased.

3.5. Conjugate gradient backpropagation with Polak-Ribiére updates

In this algorithm, gradient descent is computed at each iteration. The weights are updated as in Equation (8).

$$\Delta w_n = \beta_n \times \Delta w_{n-1} - g_n \tag{8}$$

The parameter β_n is calculated as in (9) where g_{n-1} represents the previous gradient.

$$\beta_n = \left((g_n - g_{n-1})^T \times g_n\right) \Big/ norm(g_{n-1}^2) \tag{9}$$

3.6. Conjugate gradient backpropagation with Powell-Beale restarts

In this algorithm (Powell, 1977), gradient descent is computed at each iteration. The weights are updated as in Equation (10).

$$\Delta w_n = \beta_n \times \Delta w_{n-1} - g_n \tag{10}$$

The parameter β_n is calculated as in (11).

$$\beta_n = \left((g_n - g_{n-1})^T \times g_n\right) \Big/ norm(g_{n-1}^2) \tag{11}$$

The search direction is reset to the negative of the gradient when the case (situation) as in Equation (12) exists.

$$\|g_{n-1}^T g_n\| >= 0.2 \|g_n\|^2 \tag{12}$$

3.7. Conjugate gradient backpropagation with Fletcher-Reeves updates

In this algorithm, gradient descent is computed at each iteration. The weights are updated as in Equation (13).

$$\Delta w_n = \beta_n \times \Delta w_{n-1} - g_n \tag{13}$$

The parameter β_n is computed as in Equation (14).

$$\beta_n = \mathrm{norm}(g_n^2) / \mathrm{norm}(g_{n-1}^2) \tag{14}$$

3.8. Scaled conjugate gradient backpropagations

Conjugate gradient algorithms require line search optimization at each iteration. Scaled conjugate algorithm is faster than other conjugate gradient algorithms since it avoids computationally expensive line search per iteration using a step size scaling mechanism (Moller, 1993).

3.9. BFGS quasi-Newton method

Newton's optimization method, which converges faster than conjugate gradient methods, requires Hessian matrix, which is a square matrix whose elements are second order partial derivatives of the activation function but is complex and expensive in terms computation. Family of algorithms that uses approximation of Hessian matrix is called quasi-Newton methods. BFGS algorithm (Broyden, 1970; Fletcher, 1970; Goldfarb, 1970; Shanno, 1970) which was independently developed by four scientists is one of them.

In this method the search direction is initialized to the negative of the gradient. In succeeding iterations, the weight

updates are calculated as in Equation (15) where H stands for the approximate Hessian matrix.

$$\Delta w = -H/g \tag{15}$$

3.10. One-step secant backpropagation

This algorithm (Battiti, 1992) can be considered as a hybrid combination of BFGS approach and conjugate gradient algorithms. The algorithm does not store Hessian matrix, instead, it assumes the identity matrix approximates the previous Hessian. At each iteration, weight values are updated as in Equation (16) where α and β are constants.

$$\Delta w_n = -g_n + \alpha \times \Delta w_{n-1} + \beta \times g_{n-1} \tag{16}$$

3.11. Resilient backpropagations

Resilient backpropagation is a batch learning method and is faster than gradient descent algorithms (Riedmiller and Braun, 1993). The main objective of the algorithm is to eliminate the harmful effect of the magnitude of the partial derivative. Therefore, instead of taking the partial derivative itself, only its sign is used to determine the step size for weight update. If the sign of the latest derivative is same as the derivative at the previous step, then the step size is increased. If the signs are different, meaning that the latest update is large and cause to jump over a local minimum, the update step size is reduced. Once, the step size is determined, the weight update takes the opposite of the sign of the derivative and is added to the weight values.

3.12. Levenberg–Marquardt backpropagation

Levenberg–Marquardt method is an iterative algorithm that approximates the global minimum for the error functions, which are expressed as sum of squares (Marquardt, 1963). The algorithm is successful for non-linear least squares problems, and hence ideal for training feed forward neural networks.

This algorithm stands between gradient descent methods and Newton optimization. But, instead of calculating Hessian matrix directly as in Newton method (which is complex in computation), it is approximated as in Equation (17) where J stands for Jacobian matrix containing the first order partial derivatives of the output errors with respect to weights and biases and is less computation expensive relative to Hessian (Hagan and Menhaj, 1944).

$$H = J^T J \tag{17}$$

The gradient is computed as in Equation (18) where e is the error vector.

$$g = J^T e \tag{18}$$

The weight updates are computed as in Equation (19) where I stands for identity matrix and μ is a scalar coefficient.

$$\Delta w = \left(J^T J + \mu I\right)^{-1} J^T e \tag{19}$$

4. Experiments

We evaluate the performance of different training algorithms of backpropagation neural networks introduced in Section 3 for keystroke based verification. In our experiments, we use the benchmark dataset that was collected in the study by Killhoury and Maxion (2009). The dataset includes 400 samples from 51 participants. A shared password (i.e., "**.tie5Roanl**") was assigned for all users, and together with the enter key, keystroke signature has 11 keystrokes. The feature set includes keystroke latency (consecutive keydown times, keyup–keydown times) and keystroke duration (hold times for each keystroke) and consists of 31 timing features in total.

We implement and test the algorithms using MATLAB Programming Environment (The MathWorks Inc, 2008). For our tests, we made a slight modification to the original dataset, containing alphabetic characters, which is problematic to place in a matrix in MATLAB environment. We remove the informative header and rewrite the first column containing the alphanumeric subject identifiers in numerical form. We remove the session and repetition number columns from the dataset and put two columns (usage and impostor flags) to discriminate training, test, and impostor samples in the dataset. We provide the MATLAB script used to process the dataset together with the modified dataset (Uzun, 2011) for full transparency of our test procedure.

In our experiments, first half of the samples is used for training and the second half is used for tests. In addition, first five samples in the second half of each user's dataset were used for simulating impostor login attempts for other users. By adopting the earlier evaluation methodology and by using the publicly accessible dataset, a valid comparison of our experimental results both with the preceding work (Killhoury and Maxion, 2009) as well as with potential future studies, is facilitated.

Each of the neural networks that we use consists of three layers. The input layer has 31 neurons corresponding to 31 features, hidden layer has 20 neurons, and the output layer consists of a single neuron in parallel to the configuration in (Killhoury and Maxion, 2009). Maximum number of epochs is set to 50 and learning rate is 0.1.

In order to perform the experiments, we develop a public MATLAB library (Uzun, 2011), which can also be used for any other biometric dataset as long as it conforms to the following requirements:

- The data is in tab separated text file (tsv) format.
- Each row represents a typing (feature) sample.
- Each row is in the following record structure:

User ID	Usage Flag	Impostor Flag	Feature 1	Feature 2	…	Feature N

Table 3 – The error results achieved with different weight update functions using only positive data. (bp: backpropagation).

Acronym	Training algorithm	Average EER with fix initial weights (%)	Average EER with random initial weights (%)
gdb	Gradient descent bp.	41.89	27.57
gdm	Gradient descent with momentum bp.	37.83	23.27
gda	Gradient descent with adaptive learning rate bp.	49.55	27.05
gdma	Gradient descent with momentum and adaptive learning rate bp.	34.99	23.76
cgpr	Conjugate gradient bp. with Polak-Ribiére updates	62.10	43.62
cgpb	Conjugate gradient bp. with Powell-Beale restarts	63.49	42.17
cgfr	Conjugate gradient bp. with Fletcher-Reeves updates	71.94	58.09
scg	Scaled conjugate gradient bp.	43.44	25.91
bfgs	BFGS quasi-Newton method	57.21	40.58
oss	One-step secant bp.	51.22	38.52
rbp	Resilient bp.	54.76	49.72
lmb	Levenberg–Marquardt bp.	54.00	45.70

- The description of record fields are presented as follows:
  - **User ID:** An integer identifier that is unique for each user or participant.
  - **Usage Flag:** An integer flag that is used to discriminate training and test samples. The value must be set to 1 for training samples and 2 for test samples.
  - **Impostor Flag:** An integer flag that is used for impostor test samples. The value must be set to 1 if the row is used as an impostor test sample for other users and 0 otherwise.
  - **Feature 1:** First feature value.
  - **Feature 2:** Second feature value.
  - **Feature N:** Last feature value.

In the first stage of our tests, for each user, the network is trained using only positive examples to produce a single, binary output, which is +1. Using receiver operating characteristic curves (Mayoue, 2007), EER is computed for each user. Then, we compute the average of the EER values for each weight update algorithm. We make experiments with two different initialization methods. First, we initialize all the weights to 0.1 as in (Killhoury and Maxion, 2009) and train the networks using 12 different backpropagation algorithms. Then, we perform the tests with the same parameters, with the exception that the initial weights were chosen randomly. In order to reduce the randomness in the results for random weight initialization, we run the second group of tests for 10 times for each algorithm and measure the average of the results of these test runs (we include all 51 users in the calculation of average EER values, none of the users are discarded as outlier in the calculation of average values). The calculated average equal error rates for the algorithms are listed in Table 3.

When only positive data is used for the training of neural networks with constant initial weights, the equal error rates are higher than 40 percent except two algorithms (*gdm* and *gdma*). Even for these two algorithms, the error rates are high (37.83% and 34.99%) and obviously unacceptable, considering that even a random verifier can make verification with 50% success. When random weight initialization scheme is preferred, gradient descent with momentum backpropagation performs the best with 23.27% equal error rate, which is much better than the rate with constant weights, but still much worse than the best algorithm (with 9.6% error rate) in (Killourhy and Maxion, 2009). Another observation is that scaled conjugate gradient backpropagation and family gradient descent of backpropagation algorithms perform better than the rest of the algorithms with equal error rates less than 30 percent with random initialization.

In the second stage of our tests, each network is trained using both positive and negative examples to produce a single binary output, which is +1 for the valid user and −1 for all other users. When training the neural network for a single user, we use the first half of the samples (the first 200) of that particular user as positive examples and the first halves (the first 200) of the keystroke samples of all the other (50) users as negative examples, making 200 positive samples versus 10000 negative examples in total. The weights were initialized randomly. We repeat this procedure for all the 51 users participated in the tests. As in the first stage, for each user we run the tests for ten times for each of the 12 training algorithms. We present the mean EER for each algorithm in Table 4.

Table 4 – The error results achieved with different weight update functions using both positive and negative data (bp: backpropagation).

Acronym	Training algorithm	Average EER (%)
gdb	Gradient descent bp.	83.06
gdm	Gradient descent with momentum bp.	51.45
gda	Gradient descent with adaptive learning rate bp.	49.65
gdma	Gradient descent with momentum and adaptive learning rate bp.	50.43
cgpr	Conjugate gradient bp. with Polak-Ribiére updates	25.34
cgpb	Conjugate gradient bp. with Powell-Beale restarts	20.30
cgfr	Conjugate gradient bp. with Fletcher-Reeves updates	21.13
scg	Scaled conjugate gradient bp.	18.88
bfgs	BFGS quasi-Newton method	10.97
oss	One-step secant bp.	20.55
rbp	Resilient bp.	27.54
lmb	Levenberg–Marquardt bp.	8.07

When we analyze the test results in Table 4, we see that family of gradient descent methods provide performance results no better than a random detector and conjugate gradient methods perform better than gradient descent algorithms with equal error rates ranging between 18.88% and 25.34%. The accuracy of resilient and one-step secant backpropagation algorithms is close to the accuracy of conjugate gradient family. BFGS quasi-Newton method and Levenberg–Marquardt backpropagation are the most promising algorithms with the average equal error rates of 8.07% and 10.97%, respectively.

The average EER for the neural network trained with the Levenberg–Marquardt algorithm (8.07%) is better than the best EER (9.6%) reported in (Killhoury and Maxion, 2009). As a result, we conclude that when negative examples are used for training, neural networks provide better performance results than the other methods for keystroke verification.

In order to make a sound comparison between the results of the algorithms, we compute 95% confidence intervals for differences of means for the Levenberg–Marquardt algorithm and the remaining training algorithms, which are listed in Table 5. Among these intervals, only one of them (BFGS quasi-Newton method) crosses over zero, therefore, we can state that Levenberg–Marquardt algorithm is significantly better than all the remaining 10 algorithms.

Distribution of errors for different participants is also a significant factor when evaluating the success of a system. The box plot diagram in Fig. 1 illustrates the distribution of EER values for 51 users. For each training algorithm, the diagram divides the EER values into four equal parts with three separating points: lower quartile, median and upper quartile. Horizontal lines in the middle of the boxes show the median EER and the box edges represent upper and lower quartiles. One fourth of the EER values lay below the lower quartile, another fourth is between lower quartile and the median, the third is between median and upper quartile and the last one is above the upper quartile. The whiskers extend

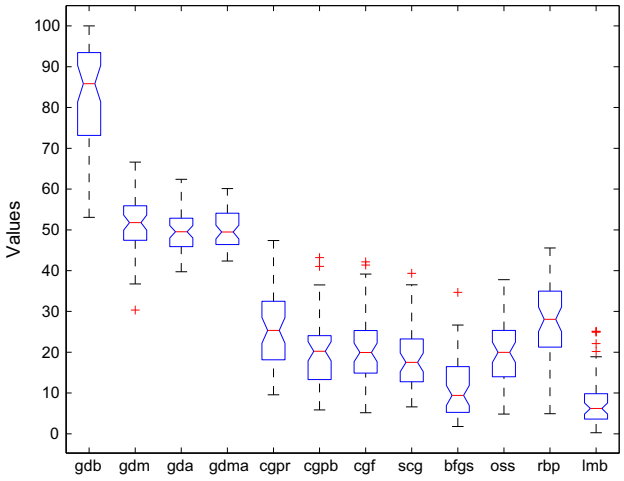

Fig. 1 – Box plot diagram showing the distribution of equal error rates in the sample population (Acronyms for the algorithms are explained in Table 3).

the boxes with 1.5 times the interquartile range from the ends of the boxes. The plus signs stand for the outlier elements which do not fit within boxes or whiskers.

In Fig. 1, the error rates for gradient descent algorithms are scattered almost randomly above or around 50%, confirming that they are no different than a random verifier. The upper quartiles for the conjugate gradient methods, one-step secant backpropagation and resilient backpropagation are significantly lower than 40%, which means that they provide better accuracy than random verifier at least 75% of the users. The error rate corresponding to the lower quartile is much lower for the BFGS quasi-Newton and Levenberg–Marquardt backpropagation than the other algorithms, as expected. Another observation from Fig. 1 is that for the neural network trained with Levenberg–Marquardt backpropagation, even for outlier participants (typically inconsistent typers), the error rates are below 30%.

Besides the type of training algorithm, there are other considerations for training a neural network. One such issue is the learning rate parameter, which is the proportion of the adjustment to the weight values updated by each learning step. The larger the learning rate, the greater the size of steps toward optimum, thereby enabling faster learning if the variation in the input data is small. However, if the variation in the training set is high, a large learning rate may lead to oscillations in the error thereby preventing the algorithm to find the global optimum. Another parameter of a neural network is the number of neurons in each layer. Since the number of neurons in the first layer must be equal to the number of inputs and there must be a single neuron in the last layer because of the nature of the problem, the only layer that can be altered is the hidden (middle) layer. In the experiments above, the learning rate was fixed to 0.1 and the number of neurons in the hidden layer was 20. As an extension to our work, we investigate the effects of changing these parameters on the EER value for Levenberg–Marquardt backpropagation algorithm. We present the results in Table 6. As seen from

Table 5 – Confidence intervals (95%) for differences between the mean of Levenberg–Marquardt algorithm and other training algorithms.

Algorithm	Confidence interval for differences of means
Gradient descent bp.	69.41 ↔ 80.57
Gradient descent with momentum bp.	37.79 ↔ 48.96
Gradient descent with adaptive learning rate bp.	36.00 ↔ 47.16
Gradient descent with momentum and adaptive learning rate bp.	36.78 ↔ 47.94
Conjugate gradient bp. with Polak-Ribiére updates	11.69 ↔ 22.85
Conjugate gradient bp. with Powell-Beale restarts	6.65 ↔ 17.81
Conjugate gradient bp. with Fletcher-Reeves updates	7.47 ↔ 18.64
Scaled conjugate gradient bp.	5.23 ↔ 16.39
BFGS quasi-Newton method	**−2.69 ↔ 8.47**
One-step secant bp.	6.89 ↔ 18.05
Resilient bp.	13.89 ↔ 25.05

Table 6 – Effects of changing the learning rate and number of hidden layers on the EER value for Levenberg–Marquardt backpropagation algorithm.

Number of neurons in the hidden layer	Learning rate			
	0.0001	0.001	0.01	0.1
10	8.26	8.52	8.46	8.21
20	8.75	7.73	8.62	8.07
30	7.87	7.76	8.04	7.99
40	8.56	7.90	8.19	7.74
50	8.42	7.95	8.18	8.28

Table 6, EER value can be reduced slightly by fine-tuning the parameters (The minimum EER, 7.73%, is achieved with 20 neurons in the hidden layer and with the learning rate of 0.001).

In the above tests, all available negative examples were used for the training. More precisely stated, when training the neural network for a single user, we used 200 training samples of that particular user as positive data and 10000 training samples (200 samples for each of 50 users) as negative data. In order to examine possible scenarios in which negative sample set is available for smaller impostor population sizes, as a third step of our tests we train the neural network with different impostor population sizes ranging from 5 to 50 users (with increments of 5) using the best-performing algorithm (Levenberg–Marquardt backpropagation). For each impostor population size p, we train the network using the samples of first p users in the original dataset (excluding the subject user) as negative samples. The corresponding change in EER value is shown in Fig. 2. The maximum EER is obtained as 16.63% when only 5 training impostors are used. EER reduces as impostor population size increases.

Lastly, we analyze the effect of changing the training set size on keystroke verification. In previous work, keystroke samples of each user were divided equally into two halves for training and test (200 for each) and we comply with this rule in our earlier analysis. Increasing the size of the training set may improve the learning capability of the network and thus reduce the error rate. To test this hypothesis, we repeat our tests using the first 300 samples of the keystroke set of each user (instead of 200) for training and the remaining 100 samples for testing. There is no other change in these tests (during the testing, we use the first five samples in the new testing dataset as impostor login attempts). For Levenberg–Marquardt backpropagation, the average equal error rate is reduced from 8.07% to 6.89%, which shows that greater number of samples used in training provides better prediction accuracy.

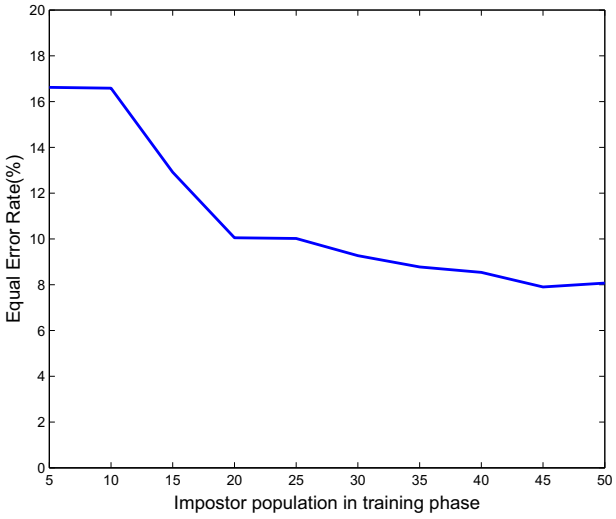

Fig. 2 – Equal error rates corresponding to impostor population sizes used in training.

5. Conclusion

To advance the field of Keystroke Dynamics, the importance of shared data and reproducible test results cannot be overemphasized. Killourhy and Maxion's evaluation work (2009) and accompanying dataset is seminal in this respect.

Motivated by their poor performance results reported in Killourhy and Maxion's work (2009), we revisit neural networks for the problem of keystroke dynamics. Using the same dataset and the evaluation methodology, we show that when negative examples are used to feed a backpropagation neural network and if a suitable training algorithm is applied, backpropagation neural networks can outperform all the other detectors evaluated in the aforementioned study. Our tests are made publicly available (see Uzun, 2011). Therefore, we conclude that backpropagation neural network is a viable alternative for identifying individuals based on Keystroke Dynamics data.

Another important result of our experiments is that for keystroke verification the performance of artificial neural networks significantly depends on the configured training algorithm. An inappropriate algorithm selection may give an error rate as high as a random verifier whereas with a proper algorithm the best performance result could be achieved. We should also note that there may still be room for further improvement.

There are several promising future directions in the field. We point out only one of them here. As smart phones are increasingly popular, security becomes a growing concern for these portable devices, which are subject to loss and theft. Although recent studies start investigating keystroke dynamics for these devices, to the best of our knowledge, currently there is no publicly available dataset collected for them. A public dataset could provide a base for making a comprehensive study for keystroke based verification on mobile devices.

Acknowledgment

We would like to thank TUBITAK (The Scientific and Technological Research Council of Turkey) for providing financial support to Yasin Uzun during his PhD study. We thank Musa Ataş, Fatih Kaya and anonymous reviewers for their valuable comments on the manuscript.

REFERENCES

Battiti R. First and second order methods for learning: between steepest descent and Newton's method. Neural Computation 1992;4(2):141–66.

Beale MH, Hagan MT, Demuth H. Neural network toolbox 7 user's guide. Natick, MA: The MathWorks; 2010.

Bleha S, Hussien B, Slivinsky C. Computer access security systems using keystroke dynamics. IEEE Transactions on Pattern Analysis and Machine Intelligence 1990;12(12):1217–22.

Broyden CG. The convergence of a class of double-rank minimization algorithms. Journal of the Institute of Mathematics and Its Applications 1970;6:76–90.

Bryan WL, Harter N. Studies in the physiology and psychology of the telegraphic language. Psychological Review 1994;4(1): 27–53.

Cho SC, Han CDH, Han DH, Kim HH. Web-based keystroke dynamics identity verification using neural network. Journal of Organizational Computing and Electronic Commerce 2000; 10(4):295–307.

Clarke NL, Furnell SM. Authenticating mobile phone users using keystroke analysis. International Journal of Information Security 2007;6:1–14.

Fan P, Sang Y, Shen H. Novel impostors detection in keystroke dynamics by support vector machine. In: Proceedings of parallel and distributed computing: applications and technologies; 2004. Singapore.

Fletcher RA. New approach to variable metric algorithms. Computer Journal 1970;13:317–22.

Giot R, El-Abed M, Rosenberger C. GREYC keystroke: a benchmark for keystroke dynamics biometric systems. In: IEEE third international conference on biometrics: theory, applications and systems (BTAS); 2009a. Washington DC, USA.

Giot R, El-Abed M, Rosenberger C. Keystroke dynamics with low constraints SVM based passphrase enrollment. In: IEEE third international conference on biometrics: theory, applications and systems (BTAS); 2009b. Washington DC USA.

Goldfarb DA. Family of variable metric updates derived by variational means. Mathematics of Computation 1970;24:23–6.

Green DM, Swets JA. Signal detection theory and psychophysics. New York: Wiley; 1996.

Hagan MT, Menhaj M. Training feed-forward networks with the Marquardt algorithm. IEEE Transactions on Neural Networks 1944;5(6):989–93.

Haider S, Abbas A, Zaidi AK. A multi-technique approach for user identification through keystroke dynamics. In: Proceedings of IEEE international conference on systems, man and cybernetics. Canada: Montréal; 2000. p. 1336–41.

Jain AK, Ross A, Prabhakar S. An introduction to biometric recognition. IEEE Transactions on Circuits and Systems for Video Technology 2004;14(1):4–20.

Joyce R, Gupta G. User authorization based on keystroke latencies. Communications of ACM 1990;33(2):168–76.

Kang P, Hwang S, Cho S. Continual retraining of keystroke dynamics based authenticator. Lecture Notes in Computer Science-Advances in Biometrics 2007;4642:1203–11.

Killourhy KS, Maxion RA. Comparing anomaly detectors for keystroke dynamics. In: Proceedings of the 39th annual international conference on dependable systems and networks; 2009. p. 125–34. Lisbon, Portugal.

Lee H, Cho S. Retraining a keystroke dynamics-based authenticator with impostor patterns. Computers & Security 2007;26(4):300–10.

Loy CC, Lai WK, Lim CP. Keystroke patterns classification using the ARTMAP-FD neural network. In: Proceedings of the third international conference on international information hiding and multimedia signal processing; 2007. p. 61–4. Splendor Kaohsiung, Taiwan.

Marquardt DW. An algorithm for least-squares estimation of nonlinear parameters. SIAM Journal on Applied Mathematics 1963;11(2):431–41.

Mayoue A. Performance evaluation of a biometric verification system. Open-source BioSecure tool, http://svnext.it-sudparis.eu/svnview2-eph/ref_syst/Tools/PerformanceEvaluation/; 2007.

Moller MF. A scaled conjugate gradient algorithm for fast supervised learning. Neural Networks 1993;6(4):525–33.

Monrose F, Rubin AD. Keystroke dynamics as a biometric for authentication. Future Generation Computer Systems 2000; 16(4):351–9.

Napier R, Laverty W, Mahar D, Henderson R, Hiron M, Wagner M. Keyboard user verification: toward an accurate, efficient and ecologically valid algorithm. International Journal of Human-Computer Studies 1995;43:213–22.

Obaidat MS, Macchairolo DT. An on-line neural network system for computer access security. IEEE Transactions on Industrial Electronics 1993;40:235–42.

Obaidat MS, Sadoun B. Verification of computer users using keystroke dynamics. IEEE Transaction on Systems, Man, and Cybernetics – PART B: Cybernetics 1997;27(2).

Powell MJD. Restart procedures for the conjugate gradient method. Mathematical Programming 1977;12:241–54.

Riedmiller M, Braun H. A direct adaptive method for faster backpropagation learning: the RPROP algorithm. In: Proceedings of the IEEE international conference on neural networks (ICNN); 1993. p. 586–91. San Francisco, Calif, USA.

Shanmugapriya D, Padmavathi G. A survey of biometric keystroke dynamics: approaches, security and challenges. International Journal of Computer Science and Information Security 2009;5(1).

Shanno DF. Conditioning of quasi-Newton methods for function minimization. Mathematics of Computation 1970;24: 647–56.

The MathWorks Inc.. MATLAB version 7.6.0. Natick: Massachusetts; 2008.

Uzun Y. Implementation of neural networks for keystroke dynamics, http://bicakci.etu.edu.tr/kd_nn/README.htm; 2011.

Yasin Uzun is a Ph.D. student at Department of Information Systems, Middle East Technical University in Turkey. He received his M.S degree from Department of Computer Science, Bilkent University in Turkey. He is currently researching the possible uses of Machine Learning methodologies for security applications.

Dr. Kemal Bıçakcı is currently an associate professor at Computer Engineering Department, TOBB (The Union of Chambers and Commodity Exchanges of Turkey) University of Economics and Technology. He has obtained his PhD degree from Middle East Technical University, Ankara, Turkey in 2003. Between 2004 and 2006, he was a postdoc researcher in Vrije Universiteit Amsterdam working with Prof. Tanenbaum in EU FP6 project named SecurE-Justice. His previous research experience includes several NSF funded security projects in which he participated as a research assistant during his MS studies in University of Southern California, Los Angeles, USA. Dr. Bıçakcı has directed a research project on usable security funded by Turkish Scientific and Technological Research Institute (TUBITAK).