Application of Neural Network in User Authentication for Smart Home System

A. Joseph, D.B.L. Bong, D.A.A. Mat

Abstract— Security has been an important issue and concern in the smart home systems. Smart home networks consist of a wide range of wired or wireless devices, there is possibility that illegal access to some restricted data or devices may happen. Password-based authentication is widely used to identify authorize users, because this method is cheap, easy and quite accurate. In this paper, a neural network is trained to store the passwords instead of using verification table. This method is useful in solving security problems that happened in some authentication system. The conventional way to train the network using Backpropagation (BPN) requires a long training time. Hence, a faster training algorithm, Resilient Backpropagation (RPROP) is embedded to the MLPs Neural Network to accelerate the training process. For the Data Part, 200 sets of UserID and Passwords were created and encoded into binary as the input. The simulation had been carried out to evaluate the performance for different number of hidden neurons and combination of transfer functions. Mean Square Error (MSE), training time and number of epochs are used to determine the network performance. From the results obtained, using Tansig and Purelin in hidden and output layer and 250 hidden neurons gave the better performance. As a result, a password-based user authentication system for smart home by using neural network had been developed successfully.

Keywords—Neural Network, User Authentication, Smart Home, Security

I. INTRODUCTION

RECENTLY there are a lot of criminals happening especially at residential areas. This shows that the security systems available in the market are not powerful enough. For an example, the security system can be easily hacked. Besides, the security system and the door lock are separated. Hence, the intruders can still break in without knowing the password for the security system. Therefore, a more powerful security system is required for the home safety.

Password-based user authentication is inexpensive and affordable. Currently most of the password-based user authentication systems are still using a table to keep the username and password of the authorized users. However, this password table has a potential threat that the passwords may be read or altered by an intruder.

The password-based user authentication using neural network which is introduced here is harder to be hacked. The neural network is used to train (generate and memorize) the identification parameters. One of the most well known types of neural network is the Multilayer Perceptrons Neural Network (MLPs). As a consequence of MLPs required hundred or even thousand of epochs to finish, even for a simple training since it is using Backpropagation technique, Resilient Backpropagation (Rprop) technique will be used to accelerate the training epochs in this paper. This was due to Backprogation Neural Network required a long time to train the nodes [1- 3].

By using the Neural Network system, it is safe enough for the user to combine the door lock with the security system because it is hard for the intruder to hack the system and get the UserID and password. Hence, the user does not need a key to open the door and no key lost or stolen will occur. Furthermore, this system can be applied as the authorization system before entering the smart home controlling system. Therefore, even the owner lost the hardware to remote access the smart home, the person who got it also difficult to access the smart home system because it is hard to crack the owner's User ID and password.

II. LITERATURE REVIEW

A. Resilient Backpropagation (RPROP)

MLP usually use sigmoid transfer functions in the hidden layers, these functions are often called squashing functions because they compress an infinite input range into a finite output range. Sigmoid functions are differentiated by the fact that their slopes must approach zero as the input gets large. Therefore, it causes a problem when a steepest descent is used to train a multilayer network with sigmoid functions, due to the gradient can have a very small magnitude and, hence, cause small changes in the weights and biases, even though the weights and biases are far from their optimal values.

In order to eliminate these harmful effects of the magnitudes of the partial derivatives, the Resilient backpropagation (Rprop) training algorithm is introduced. Only the sign of the derivative is used to determine the direction of the weight update; the magnitude of the derivative

A. Joseph is with the Universiti Malaysia Sarawak, Faculty of Engineering, Electronic department , 93400, Kota Samarahan, Sarawak, Malaysia(phone: 6082-583272; fax: 6082-583410; e-mail: jannie@feng.unimas.my).

D.B.L.Bong is with the Universiti Malaysia Sarawak, Faculty of Engineering, Electronic department , 93400, Kota Samarahan, Sarawak, Malaysia(phone: 6082-583313; fax: 6082-583410).

D.A.A.Mat is with the Universiti Malaysia Sarawak, Faculty of Engineering, Electronic department , 93400, Kota Samarahan, Sarawak, Malaysia(phone: 6082-583295; fax: 6082-583410).

gives no effect on the weight update. The size of the weight change is determined by a separate update value. The update value for each weight and bias is increased by a factor delt_inc whenever the derivative of the performance function with respect to that weight has the same sign for two successive iterations. The update value is decreased by a factor delt_dec whenever the derivative with respect to that weight changes sign from the previous iteration. If the derivative is zero, then the update value remains the same. Whenever the weights are oscillating, the weight change is reduced. If the weight continues to change in the same direction for several iterations, then the magnitude of the weight change increases.

Rprop is generally much faster than the standard steepest descent algorithm. It also has the nice property that it requires only a modest increase in memory requirements. The update values for each weight and bias are required to store, which is equivalent to storage of the gradient [4].

B. User Authentication

User Authentication is the process of determining that a user is who he/she claims to be. Usernames and passwords are the most common form of authentication in use today. Inspire of the improved mechanisms over which authentication information can be carried, most systems usually require a password as the token against which initial authorization is performed. Due to the conflicting goals that good password maintenance schemes must meet, passwords are often the weakest link in authentication architecture. This can be only partially addressed by technical remedies. The implementation of authentication systems should measure risks and benefits against an appropriate threat model and protection target.

Usernames have few requirements for security that some basic restriction should be implemented in the system. Usernames that are derivations of a real name or actual real names can clearly give personal detail clues to an attacker.

Password quality refers to the entropy of a password and is necessary to ensure the security of the users' accounts. A good password is a password that is impossible to guess by other people. The password is suggested should at least contain 8 characters, one alphanumeric, one mixed case and at least one special character (not A-Z or 0-9) [1- 3, 5].

III. PASSWORD-BASED USER AUTHENTICATION DEVELOPMENT

The development of password-based user authentication consists of three main parts which are user registration phase, user sign in phase, and user authentication phase. User registration phase is link to user sign in phase and user sign in phase is connected to user authentication phase. Training of the data is done after the users have registered the User ID and password, and before the users are able to login the system. The flowchart for developing the whole system is shown in Figure 1.
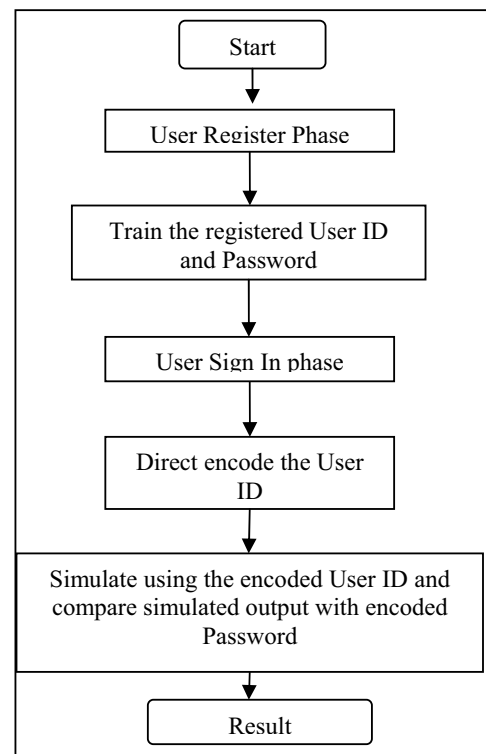

Fig. 1 Flowchart of Password-based User Authentication Development

A. Data Collection

Before the process of training the neural network model, the data of the users such as User ID and password are collected. There are 200 sets of User ID and Password were used as the training set while 85 sets were used as testing set to test the output of the simulation, where 55 sets were correct User ID and Password, 15 sets were correct User ID with wrong Password and 15 sets were Wrong User ID with correct Password.

A few factors are taken into consideration before collecting the data. First, the format of the data is taken into consideration. The data format is suggested to use alphanumeric and symbols because it is required in order to have a "strong password" which is harder to crack. However, considering the dimension of the hardware which is built to accommodate so many characters, it was decided that the input is limited to alphanumerical only. Hence, users can only either choose from the 26 alphabets (A-Z, a-z) or the 10 numbers.

Secondly, how the user is going to enter the data into the system is considered. In this paper, a basic Graphical User Interface (GUI) is built for User Registration Phase and User Login Phase. The interface consists of username, password, register and login.

Finally, the number of sets of UserID and passwords need to be determined (i.e. 200 sets of UserID and passwords). The numbers of data should be adequate for the training and testing purpose [1-3, 5].

B. Data Conversion

The information input by the user (User ID and Password) will be in the form of alphanumerical because this type of password is harder to be cracked. However, neural network can only recognize the numerical data; therefore, the alphanumerical data was required to convert into binary form.

The data keyed in by the users will first save to a text file. Then the data will be converted to binary since the range of input and output value for neural network is 0 to 1. The username and password had to be normalized before the BPN training. Every single alphabet, number and symbol was assigned to 7 bits binary code. For an example, the User ID key in by user is Ace. The binary code will be 1000001 1100011 1100101.

C. Neural Network Implementation

The first step for MLPs neural network implementation is to setup the feed-forward backpropagation neural network. Before training the network, the network is initialized. The network is ready for training once the network weights and biases are initialized. A set of examples of proper network behavior is required for the training process - network inputs p and target outputs t.

Furthermore, the network performance was measured using Mean Square Error (MSE). In the training phase, local adaptive technique called R-prop is embedded to the MLPs Neural Network to accelerate the training process since the conventional way to train the network using Backpropagation (BPN) requires a long training time.

D. User Authentication

For user authentication part, the server trained BPN to authenticate the login user whether is a legal user or intruder when it receives the login request. The authentication process is described as follows. The process of user login and authentication is shown in Figure 2.

1) The server encoded User ID and Password.
2) The User ID was entered the server products an output through the trained BPN.
3) The output was compared with the encoded password by the server. The login user was allowed to access (i.e. the door will be unlocked or enter the system) if the result was positive. On the other hand, if the result is negative, the login user will be treated as an intruder (i.e. cannot enter the door or system) [1- 3].
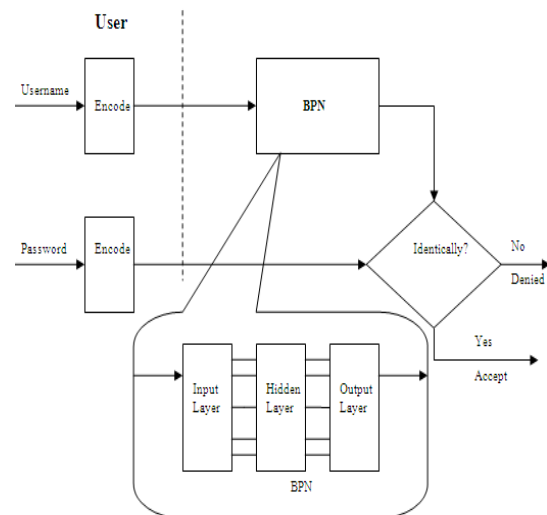

Fig. 2 The processes of login and user authentication phases

IV. RESULTS, ANALYSIS AND DISCUSSION

The performance of training time and numbers of epoch of RPROP learning algorithm are compared base on different number of samples, different hidden units and different transfer functions used for hidden layer and output layer. Two combinations of transfer function for hidden layer and output layer were carried out. First, Tansig was applied to hidden layer and Purelin was applied to output layer (Combination A). Secondly, Logsig and Purelin were applied to hidden and output layer respectively (Combination B).

Training performance graph and linear regression graph were observed each time after the program been executed. The simulation output was compared with the actual target. Besides, Mean Square Error (MSE) of the simulation was also observed in order to get the best performance.

Training process was standardized to execute for ten simulations in order to obtain the average result for the simulation. This was due to every training process would give vary performance and training time. The scenario happened because MLPs neural network used random weight initialization. The initial weights and bias are different for every training, hence, every training would have different results [6].

A. Compare the Performance of Different Hidden Neurons

The overall performance of different hidden neurons applied to the hidden layer was observed by using 150, 200, 250, 300, 350 and 400 hidden neurons. The simulations were carried out by using 200 training sets.

1. Mean Square Error (MSE)

The MSE results obtained from using different number of hidden neurons for Combination A and Combination B were shown in Table I and Table II respectively. The graph plotted for average MSE results with different number of hidden neurons were shown in Figure 3 (Combination A) and Figure 4 (Combination B).

TABLE I
MSE WITH DIFFERENT NUMBER OF HIDDEN LAYER
(COMBINATION A)

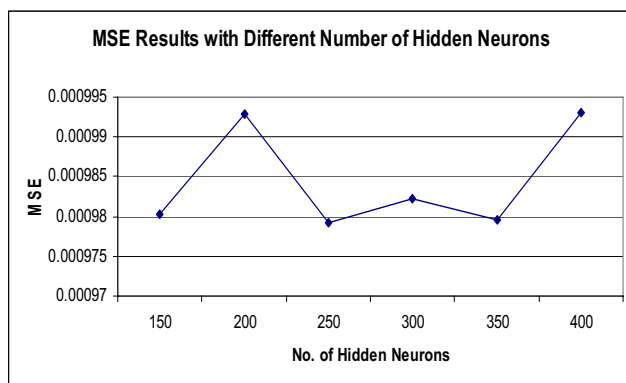| Simulation	No. of Hidden Neurons					
	150	200	250	300	350	400
1	0.001018	0.000975	0.000975	0.000983	0.000993	0.001017
2	0.000875	0.000985	0.000944	0.000960	0.000954	0.000934
3	0.000993	0.000954	0.001003	0.001000	0.000947	0.000985
4	0.001044	0.001017	0.000968	0.000984	0.000894	0.001013
5	0.001028	0.001008	0.001026	0.000967	0.001033	0.000991
6	0.000997	0.000988	0.000977	0.001094	0.000962	0.001015
7	0.001011	0.000989	0.001000	0.000948	0.000979	0.001060
8	0.000956	0.001031	0.000959	0.000917	0.001018	0.000974
9	0.000976	0.001022	0.000978	0.000963	0.001033	0.000981
10	0.000905	0.000960	0.000941	0.001008	0.000982	0.000960
Average	0.000980	0.000993	0.000979	0.000982	0.000980	0.000993

Fig. 3 MSE with Different Number of Hidden Neurons (Combination A)

TABLE II
MSE WITH DIFFERENT NUMBER OF HIDDEN LAYER
(COMBINATION B)

| Simulation	No. of Hidden Neurons					
	150	200	250	300	350	400
1	0.001005	0.001066	0.001040	0.000979	0.001065	0.000954
2	0.001045	0.000951	0.000996	0.001087	0.000996	0.000989
3	0.000983	0.000945	0.000972	0.000965	0.000970	0.001064
4	0.000978	0.001060	0.001042	0.000999	0.001087	0.000955
5	0.000949	0.000982	0.000992	0.000935	0.000968	0.001012
6	0.000996	0.001059	0.001006	0.001093	0.001084	0.001015
7	0.001003	0.000945	0.000965	0.001024	0.001066	0.001028
8	0.000995	0.001056	0.001003	0.000944	0.001081	0.000948
9	0.000976	0.000936	0.000933	0.001002	0.001010	0.001028
10	0.000974	0.001119	0.001042	0.000956	0.000958	0.000985
Average	0.000990	0.001012	0.000999	0.000998	0.001029	0.000998
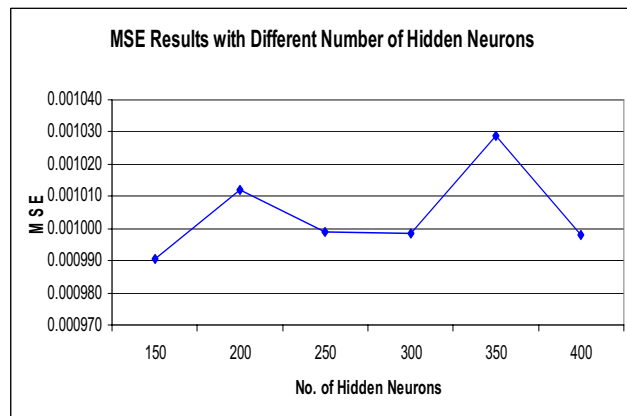
Fig. 4 MSE with Different Number of Hidden Neurons (Combination B)

The performance of training for Tansig and Purelin using different number of hidden neurons was considered good as they were able to reach the goal set for training. The difference of MSE using different number of hidden neurons to train the network was small. The average MSE results obtained were below 0.001 which the generated output would be identical to the target. 55 sets of trained UserID were chosen randomly and used as input for testing the simulated output; all the 55 sets were able to provide identical value as the target. Therefore, using Tansig and Purelin as the transfer functions for hidden and output layer were able to provide a very good output.

The performance of Logsig and Purelin as transfer function for hidden and output layer was consider good, the average MSE results were around the performance goal set. However, there were two average MSE results were slightly above 0.001, which were implementing 200 hidden neurons and 350 hidden neurons. The best MSE result was provided by using 150 hidden neurons which was 0.00099. By using 350 hidden neurons, it gave the highest MSE (0.001029). The overall MSE result still considered stable because increasing the number of hidden neurons did not cause fluctuation in the MSE result.

2. Training Time

Table III and Table IV showed the training time using different number of hidden neurons for Combination A and Combination B respectively. The average training time taken using different number of hidden neurons for Combination A and Combination B were shown in Figure 5 and Figure 6. The training time also stated in minute and second (min:sec).

TABLE III
TRAINING TIME WITH DIFFERENT NUMBER OF HIDDEN LAYER
(COMBINATION A)

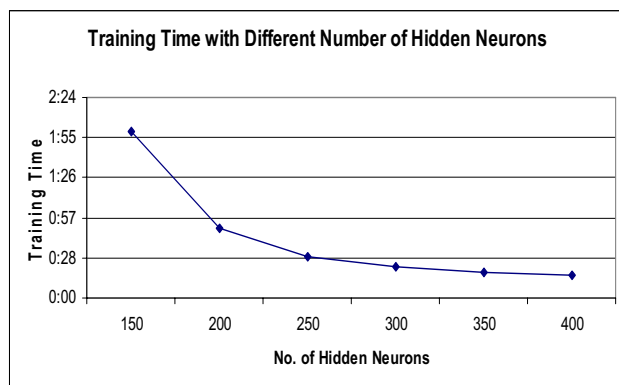| Simulation	No. of Hidden Neurons					
	150	200	250	300	350	400
1	1:57	0:50	0:34	0:22	0:18	0:17
2	2:04	0:50	0:30	0:23	0:19	0:16
3	1:40	0:51	0:30	0:22	0:18	0:16
4	1:52	0:52	0:30	0:24	0:19	0:15
5	1:52	0:51	0:29	0:23	0:18	0:16
6	2:04	0:51	0:31	0:24	0:18	0:16
7	1:54	0:49	0:30	0:23	0:19	0:16
8	2:09	0:49	0:30	0:24	0:18	0:16
9	2:11	0:49	0:29	0:22	0:19	0:16
10	2:14	0:46	0:28	0:21	0:19	0:16
Average	1:59	0:49	0:30	0:22	0:18	0:16

Fig. 5 Training Time with Different No. of Hidden Neurons
(Combination A)

TABLE IV
TRAINING TIME WITH DIFFERENT NUMBER OF HIDDEN LAYER
(COMBINATION B)

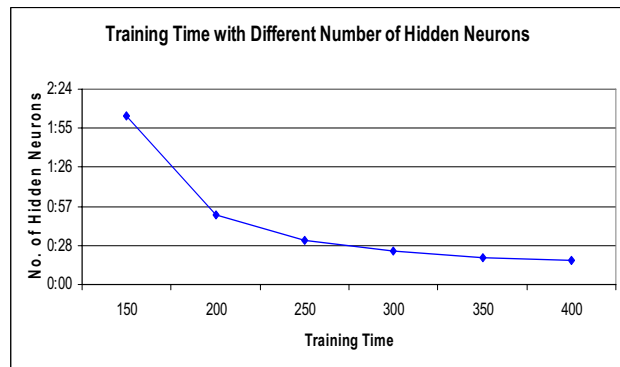| Simulation	No. of Hidden Neurons					
	150	200	250	300	350	400
1	2:07	0:55	0:31	0:24	0:20	0:18
2	1:58	0:53	0:34	0:26	0:19	0:18
3	2:11	0:49	0:33	0:26	0:20	0:18
4	2:01	0:49	0:33	0:24	0:20	0:17
5	2:03	0:49	0:34	0:23	0:20	0:17
6	1:50	0:50	0:33	0:25	0:20	0:18
7	2:09	0:57	0:33	0:24	0:20	0:18
8	2:03	0:47	0:30	0:25	0:20	0:18
9	2:04	0:47	0:32	0:25	0:19	0:18
10	2:12	0:53	0:32	0:26	0:21	0:17
Average	2:03	0:50	0:32	0:24	0:19	0:18

Fig. 6 Training Time with Different No. of Hidden Neurons
(Combination B)

For Combination A, the training time decreased as the number of hidden neurons increased. When the number of hidden neurons was increased from 150 to 200, the average training time reduced from 1 min 59 sec to 49 sec which was a difference of 1 min 10 sec. However, when the number of hidden neurons was increased from 300 to 350 and 350 to 400, the average training time only decreased 4 sec and 2 sec respectively.

The training time versus number of hidden neurons for Combination B was observed. The average training time also decreased as the number of hidden neurons increased. The biggest drop was when adding the hidden neurons from 150 to 200, the training time decreased from 2 min 3 sec to 50 sec (1 min 13 sec reduced). However, after 250 hidden neurons, the increase of number of hidden neurons did not reduce the training time much. When increased the hidden neurons from 350 to 400, the training time only reduced by 1 sec.

Although the difference of the training time for both Combination B and Combination C was not large, the training time for using Tansig and Purelin as transfer function for hidden layer and output layer was slightly shorter comparing to using Logsig and Purelin as transfer function for hidden layer and output layer.

3. Number of Epochs

The number of epochs used to train the network by using different number of hidden neurons was recorded into Table V and Table VI for Combination A and Combination B respectively. Figure 7 and Figure 8 were the average number of epochs put into graph for analyzing the trend of increasing number of hidden neurons for both Combination A and B.

TABLE V
NO OF EPOCHS WITH DIFFERENT NUMBER OF HIDDEN LAYER
(COMBINATION A)

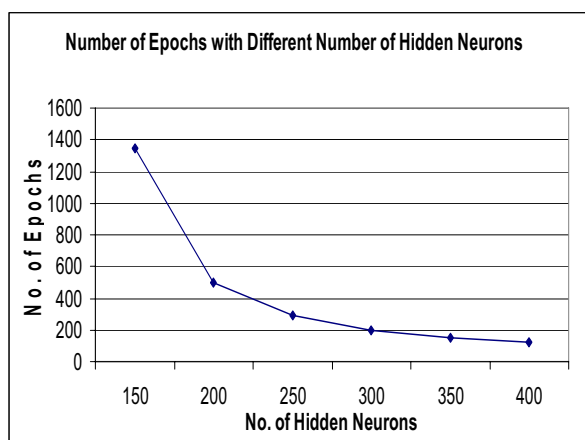| Simulation	No. of Hidden Neurons					
	150	200	250	300	350	400
1	1318	518	297	191	150	125
2	1412	504	296	202	160	124
3	1148	502	292	187	149	126
4	1254	517	289	195	155	120
5	1262	505	283	193	153	126
6	1417	507	297	198	151	126
7	1293	486	290	192	154	125
8	1447	486	286	205	149	124
9	1467	482	278	196	158	125
10	1478	459	276	187	157	125
Average	1349.6	496.6	288.4	194.6	153.6	124.6

Fig 7 No. of Epochs with Different Number of Hidden
Neurons(Combination A)

TABLE VI
NO OF EPOCHS WITH DIFFERENT NUMBER OF HIDDEN LAYER
(COMBINATION A)

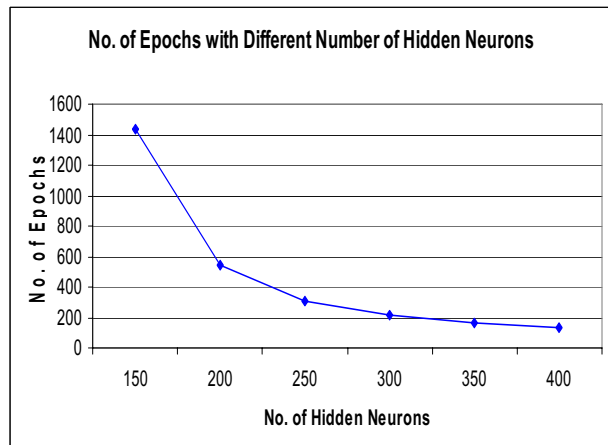| Simulation	No. of Hidden Neurons					
	150	200	250	300	350	400
1	1468	577	301	214	160	133
2	1420	562	318	224	155	133
3	1531	529	316	230	165	131
4	1406	530	308	213	165	127
5	1441	528	321	203	161	127
6	1286	537	307	221	160	136
7	1486	602	307	210	157	129
8	1421	510	282	221	161	130
9	1443	516	312	220	157	131
10	1504	583	307	226	170	129
Average	1440.6	547.4	307.9	218.2	161.1	130.6

Fig 8 No. of Epochs with Different Number of Hidden
Neurons(Combination B)

For Combination A, the number of epochs during the training decreased as the number of hidden neurons was increased. By using 150 hidden neurons, 1349.6 of epochs were required to train the network. The least number of epochs used to train the network was 124.6 of epochs by implementing 400 hidden neurons. The number of epochs dropped from 1349.6 to 496.6 when the number of hidden neurons increased from 150 to 200. However, after 250 hidden neurons, increasing the number of hidden neurons was only able to slightly reduce the number of epochs.

For Combination B, the number of epochs during the training also decreased as the number of hidden neurons was increased. Applying 150 hidden neurons gave the greatest number of epochs which was 1440.6. On the other hand, 400 hidden neurons gave the least number of epochs which was 130.6. The effect on the number of epochs by increasing number of hidden neurons was the same as the effect on the training time. The biggest drop was when increased the number of hidden neurons from 150 to 200, the number of epochs dropped from 1440.6 to 547.4. After that, the increased on number of hidden neurons reduced less and less number of epochs.

By comparing the overall results of MSE results, training time and number of epochs of the training, it was observed that by using Tansig and Purelin as the transfer function for hidden and output layer respectively provide the optimum training. Besides, using 250 hidden neurons gave the best MSE performance 0.000979 and the training time was 30 sec which was also acceptable. Applying 250 hidden neurons was chosen instead of applying 400 hidden neurons though 400 hidden neurons required only 16 sec to train the network because the difference was only 14 sec. Furthermore, using 400 hidden neurons might be too complex for the network and the required large memory to do the training.

B. The Overall Phases for Password-based User Authentication

The overall phases for the password-based user authentication consists of user registration phase, user sign in phase and user authentication phase. The three parts were further discussed in the sections below.

1. User Registration Phase

Figure 9 showed the interface for user registration. The user will register the User ID and password at the user registration interface.

Fig. 9 User Registration Interface

The User ID and password will be stored in a text file before all the registrations were completed. Every time when the user clicked the button "SUBMIT", the data stored in the text file was encoded into binary and being trained. The users were limited to choose a maximum of eight characters for their User ID and password. The reason for limiting the number of characters was to fix the dimension for the input and target.

2. User Sign In Phase

The User Sign In Interface was shown in Figure 10. User was required to key in User ID and password before login the system.

Figure. 10 User Sign in Interface

When the 'SUBMIT" button was clicked, the User ID and password were encoded and request were sent to user authentication phase to determine the legality of the user.

3. User Authentication Phase

At the user authentication phase, the legitimacy of the login user was determined. The encoded username was entered to the trained network and produced an output. The output would be compared with the encoded password. If the output matched with the encoded password, the user will be recognized and able to login the system. The result for a legal user was shown in Figure 11. On the other hand, if the output of the network did not match with the encoded password, the user will be rejected. The result for an illegal user was shown in Figure 12.
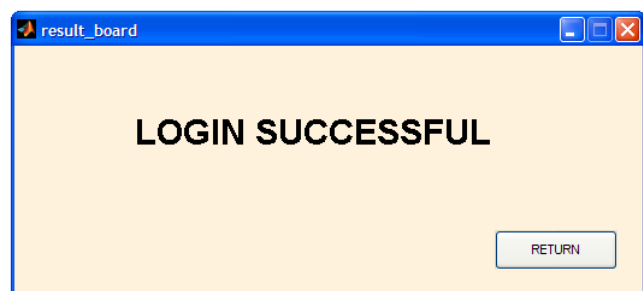
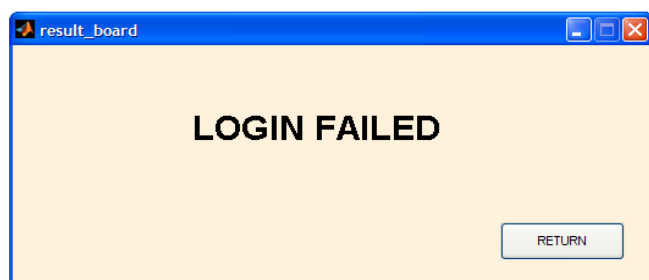

Fig. 11 User Login Successful Interface

Fig. 12 User Login Failed Interface

C. Discussions

From the result obtained, it showed that the trained feedforward backpropagation neural network using RPROP as training function generates outputs that are exactly identical to the targets. Hence, it was proved that the authentication system had provided a good accuracy. During the testing, the same training pattern was used to test the trained network. If the right User ID was input to the trained network, the output was the corresponding Password. This was due to the registered User ID and Password had been used in the training. Therefore, the error of the generated output would be very small which was very close to the actual Password. On the other hand, if User ID which was not trained (unauthorized) was input to the network, the generated output

would not be accepted as it would not be equal to the existing Password. Furthermore, if the User ID entered was an authorized User ID but the password entered was wrong, the user will not be able to enter the system too because the generated output does not match the entered password.

The difficulty of inverting the test pattern to the original password determined the security of the system. Hence, an intruder cannot easily derive the secret password from the test pattern even if the weights of the trained network were obtained by the intruder.

Furthermore, the training time required to train the data was much shorter comparing to the conventional BPN method [5, 7]. By embedding RPROP, the system only require 30 sec to train the network to train 200 training sets. Note that the input pattern of this system was also digital data (0 or 1).

Besides, unlike public key cryptography which required exponential computing, the process of this BPN only requires simple multiplication and addition to obtain the result [5]. Hence, when a user wants to enter the house or the smart home system, the system could response with the result either to accept or reject the user's request.

V. CONCLUSION

A password-based user authentication system for smart home by using neural network had been developed successfully. A local adaptive learning algorithm, Rprop, has been embedded to train the network. From the result obtained in section IV, it is observed that the implementing Tansig in the hidden layer and Purelin in the output layer give optimum results. Besides, it is observed that as the number of hidden neurons increased, the training time and number of epochs to train the network increased proportionally. However, the number of hidden neurons do not influence much on the MSE results because this is a pattern recall system.

The use of neural networks has been applied to eliminate the disadvantages of maintaining the conventional verification table for user authentication system. BPN is the method being employed to recall the relationship of User ID and Password that had been registered. The corresponding Password can be easily obtained according the input User ID by using this method. Hence, it could be used to replace the verification table stored in the system.

Furthermore, only simple and quick computation operations are required to produce the result instead of complex calculation as in public key cryptography. Therefore, users do not need to wait for a long time for the authentication system to response.

The advantage of embedding a local adaptive technique, Rprop, is that the training time is very short. Hence, the long training time of applying MLP – BPN in authentication system can be solved.

REFERENCES

[1] I.C.Lin,H.H. Ou, M.S. Hwang, "A user Authentication System using Back-propagation Network," Neural Comput & Applic, June 2005
[2] U. Manber, "*A Simple Scheme to make Passwords Based on One Way Functions Much Harder to Crack*",Nov 2000
[3] S.Z. Reyhani, M. Mahdavi, "*User Authetication Using Neural Network in Smart Home Networks,*" *International Journal of Smart Home, Vol 1 no 2,pp147, July 2007.*
[4] H. Demuth, M.Beale, M.Hagan, " Neural Network Toolbox™ User Guide: Faster Training," Natick: The Mathworks™ Inc, 2008.
[5] M. Curphey, A Guide to Building Secure Web Application, *The Open Web Application Security Project (OWAS*P), Boston, USA, 2002..
[6] A. Pavelka, A.Proch' azka, " Algorithm for Initialization of Neural Network weights, Institute of Chemical Technology, department of Computing and Control Engineering.