Applicability of Feedforward Networks in Extracted Keystroke Classification

Stanley Chen

Introduction to Parallel Distributed Processing

Dr. David Plaut

December 20, 2020

Introduction

Society nowadays operates on a fully internet-centric basis, with new innovations and breakthroughs happening from minute to minute. While rapid technological advances add efficiency and versatility to people's everyday lives, the hyper-dependency on the internet simultaneously increases. Specifically, the quantity of information that can be stored and the ease-of-access of this information is a feature of the internet that society simply cannot function without.

However, one of the greatest problems that the internet has raised is the issue of internet security. A great proportion of the information that is stored on the internet is actually Personally Identifiable Information (PII) or "any data that could potentially be used to identify a particular person", such as "a full name, Social Security number, driver's license number, bank account number, passport number, and email address."[4] It is not only undesirable for PII to be leaked to other people, but in the worst case it can potentially be life-threatening if leaked information is gained by a malevolent user.

One of the most well-known implementations of internet security is the use of a password or pin, which provides individuals with a way to access personal information by inputting a string of characters that only they themselves know - at least theoretically. In practice, passwords are far from secure. People often write down their passwords somewhere or choose passwords that are easily guessed, all which increase the likelihood of someone else gaining unwarranted access to their accounts. Even more, a data breach on Twitter in 2013 resulted in attackers gaining access to the passwords of 250,000 users. [1] For one of the biggest, most popular websites to be breached to such an extent just comes to show how passwords by themselves are nowhere near sufficient in providing perfect account security. Of course, over the last half a decade or so, significant developments in encryption algorithms have drastically improved the effectiveness of passwords. But on the other hand, hackers now have access to augmented decrypting and hacking algorithms that allow them to crack passwords more easily, especially if users choose short, predictable passwords. [1]

One approach to address the flaws of passwords has been the use of keystroke dynamics, or the temporal rhythms and tendencies associated with a user when they type on a keyboard. Research has shown that keystroke dynamics is actually a behavioral biometric, in the sense that it can be exploited to potentially identify a typing pattern with a specific individual. [2] A research study in 2018 by Muliono et al. involved 51 users each typing 400 instances of the same password. Researchers then trained both a support vector machine (SVM) and a deep learning network in classifying temporal data to the correct individual, in which the deep learning approach ended up performing the most effectively with 92.60% classification accuracy. [3] Although this accuracy shows that keystrokes can in fact be used to classify users, it is extremely far from an actual implementation of user authentication on the internet. In practice, not only do users have different passwords, but for a specific user’s password, there is no keystroke data from external individuals. In other words, even if a user types their password multiple times and provides the network personal keystroke data, the network does not have any keystroke data for the same password from other users that can be used to detect unauthorized access.

In this project, I wanted to address part of this problem by seeing if it's possible to take keystroke data from a longer string and extract keystroke data for a shorter encoded string. The first simulation will be a simple classification task similar to that of the aforementioned study, using a feedforward network with one hidden layer to classify keystroke data for a password. The second simulation will involve users typing a longer string of characters with the earlier password encoded inside. I will then attempt to extract keystroke data for the password and test the network to see if it can classify the extracted data on the same set of individuals.

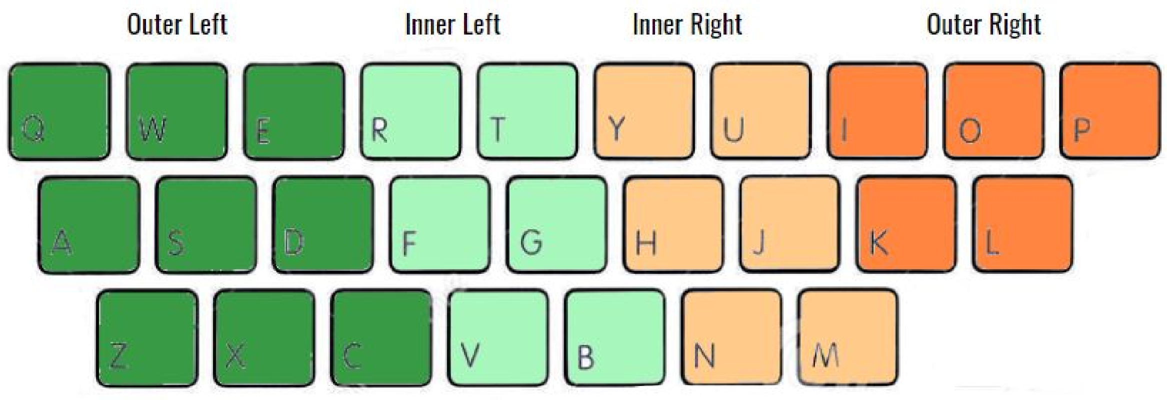
Simulation 1: Password Classification

Methods

Keystroke dynamics data is highly versatile, in the sense that when someone types a string on a keyboard, there are multiple features that can be examined. On the simplest level, there is the duration for which a key is held down. This is called *dwell time*. There is also the duration between releasing the previous key and pressing down the next key. This is called *flight time*.[2] In this simulation, I will be using only these two features as they constitute the entire duration of a single typing. Other features that could have been used are the times between key presses, the times between key releases, and the total time. However, all these features are essentially encoded within *dwell time* and *flight time*, so it would be redundant to use them as well. Another thing to note is that in practice, typing mistakes and use of the backspace key can actually provide additional information on keystroke tendencies that are unique to individuals, but that becomes a rather complex feature to encode into an input vector for a feedforward network. We will assume for simplicity that there are no typing mistakes when a user inputs a password that they are familiar with.

In this study, we found four participants over the age of 18 who use computers on a daily basis and do not have any type of physical condition that hinders typing performance. The choice of these characteristics was to have a group of individuals that do not differ drastically in typing speed or have idiosyncratic typing characteristics. Compared to the Muliono et al. study, which involved 51 participants, this choice of four participants may initially seem lacking in showing significant classification results. However, the main goal of this study is not mere keystroke classification, but rather to see if the trained network performs as well in classifying extracted keystroke data as it does in classifying actual keystroke data.

I aimed to choose a target password that could effectively capture individual keystroke characteristics by having a versatile collection of letter transitions. Consider the following color-coded letter section of a standard keyboard in which every letter is assigned to one of four groups: Outer Left, Inner Left, Inner Right, and Outer Right.



I evaluated choices of target passwords not only based on whether they contained a good balance of letters in each of the four groups, but also whether they contained a good balance of letter transitions between and within groups. The final password I ended up using was “hatyellow”. Below is a table that displays the keyboard group of each letter and the group transitions. The L/R (Left/Right) Transition column identifies if there is a transition between a left-side group and a right-side group (or vice versa). The I/O (Inner/Outer) Transition column identifies if there is a transition between an inner group to an outer group (or vice versa). For instance, the letter “h” is in the Inner Right group and the next letter “a” is in the Outer Left group, so there is both an L/R Transition and an I/O Transition. (Note that the last letter’s transition columns are grayed out since there is no next letter to transition to). The second table displays the overall count for each of the groups and transitions.

Letter	Group	L/R Transition	I/O Transition
h	Inner Right	Yes	Yes
a	Outer Left	No	Yes
t	Inner Left	Yes	No
y	Inner Right	Yes	Yes
e	Outer Left	Yes	No
l	Outer Right	No	No
l	Outer Right	No	No
o	Outer Right	Yes	No
w	Outer Left		

Left : Right	Inner : Outer	L/R Transition (Yes : No)	I/O Transition (Yes : No)
4 : 5	3 : 5	5 : 3	3 : 5

As shown in the second table, there is a relatively good balance in all of the categories, so “hatyellow” is versatile password that can capture a variety of aspects of individuals’ typing patterns.

Participants were asked to type “hatyellow” 90 times. Before actual data collection, each participant was instructed to practice typing the password for a minimum of 15 times until they felt familiar with typing it. This was done to replicate the fact that people are very familiar with typing their own passwords in real life. During data collection, participants were asked to take a 15 second break after every 20 trials before continuing again. This was to make sure that participants did not start typing the password in an automatic manner that would result in keystroke tendencies that deviated from their actual keystroke characteristics.

Each typing of a password was represented as a 17-dimensional input vector. Noting that “hatyellow” has length nine, the first nine features of the vector were the *dwell time* for each of the keys. The last eight features of the vector represent the *flight time* between each consecutive pair of keys. The actual durations were scaled by a factor of ten to normalize the features to a reasonable range.

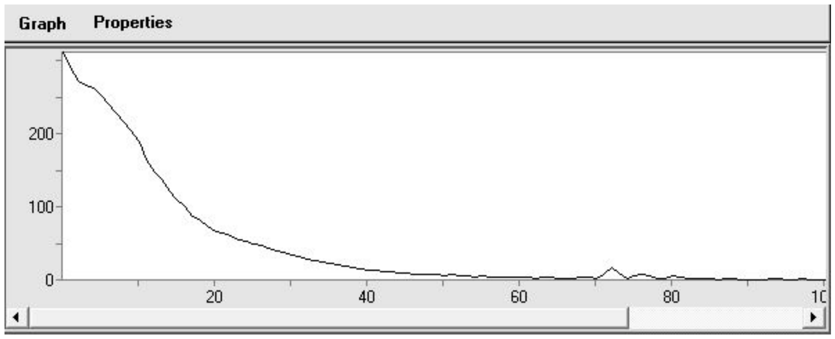
After collecting 90 instances of data from each of the four participants, I split the data randomly into training, cross-validation, and test sets, using a 50-20-20 split. The network parameters were optimized using the cross-validation set and then network training was ultimately evaluated based on classification accuracy on the test set.

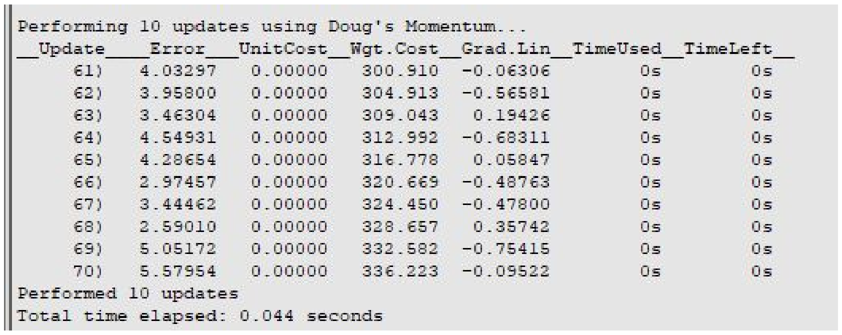
Fixed Network Parameters

The network being trained consisted of the aforementioned 17 input units, a hidden layer, and four output units. The activation of an output unit represented whether or not an input pattern belonged to a specific individual. As such, each example had a target output where exactly one output unit was 1, corresponding to the participant who actually typed the password. I trained the network with the SOFT_MAX constraint on the output group, which normalized the output activations to sum to 1. This was to essentially have the network generate a probability that a certain keystroke input belonged to each of the four individuals. The CRIT_MAX criterion was also added to the output group with a testGroupCrit value of 0.5. In other words, for the network to correctly classify an input pattern, not only does the target individual’s output node have to be the highest activation, but it also has to be above 0.5. Essentially, if the network is unconfident and generates partial activations for each output node, even if the target individual is the most activated, it should still be marked as a false classification. Lastly, the targetZeroScaling parameter was set to 2.0, which doubles the error on output nodes that have a target value of zero. This was done to penalize false positives more severely, as the main point of keystroke biometrics is to detect unauthorized user access. In other words, if an input does not belong to a certain individual and the network decides that it does, it is equivalent to allowing a hacker to access someone’s account.

Results

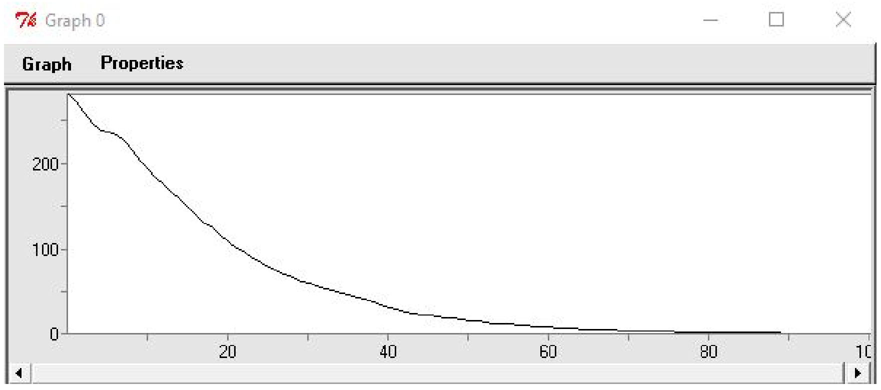
We started off training using 8 hidden units in order to fix a learning rate. At a learning rate of 0.5, although the network quickly dropped in training error after training 60 epochs, there was notable fluctuation before even reaching a stable configuration of weights. Below is the error graph and the training errors from epochs 60 to 70.

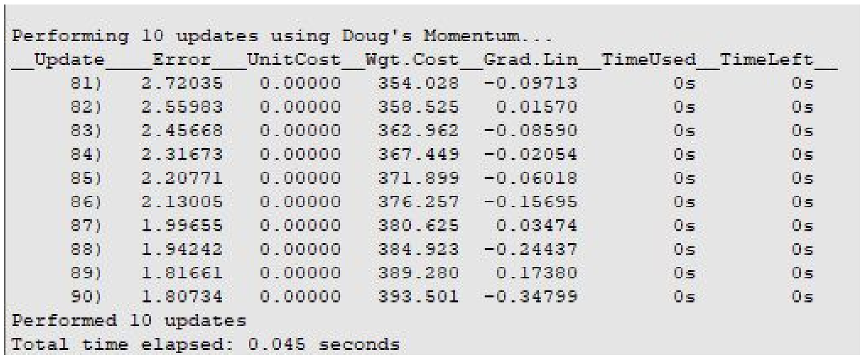




```
Performing 10 updates using Doug's Momentum...
__Update____Error___UnitCost__Wgt.Cost__Grad.Lin__TimeUsed__TimeLeft__
      61)    4.03297   0.00000   300.910  -0.06306        0s        0s
      62)    3.95800   0.00000   304.913  -0.56581        0s        0s
      63)    3.46304   0.00000   309.043   0.19426        0s        0s
      64)    4.54931   0.00000   312.992  -0.68311        0s        0s
      65)    4.28654   0.00000   316.778   0.05847        0s        0s
      66)    2.97457   0.00000   320.669  -0.48763        0s        0s
      67)    3.44462   0.00000   324.450  -0.47800        0s        0s
      68)    2.59010   0.00000   328.657   0.35742        0s        0s
      69)    5.05172   0.00000   332.582  -0.75415        0s        0s
      70)    5.57954   0.00000   336.223  -0.09522        0s        0s
Performed 10 updates
Total time elapsed: 0.044 seconds
```

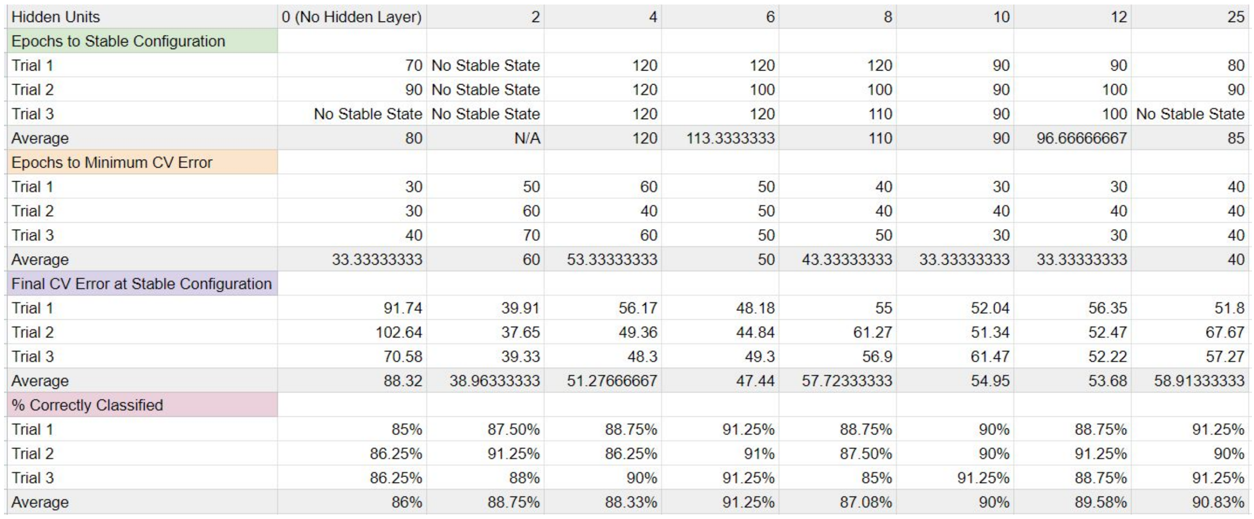
Reducing the learning rate to 0.2, there was definitely less fluctuation, but it still existed after 80 epochs of training. On reducing the learning rate to 0.1, we find that learning is relatively stable, with minor fluctuations in the first 30 or so epochs of training. However, the error curve descends smoothly afterwards and the network settles to a fairly stable state after around 90 epochs of training:





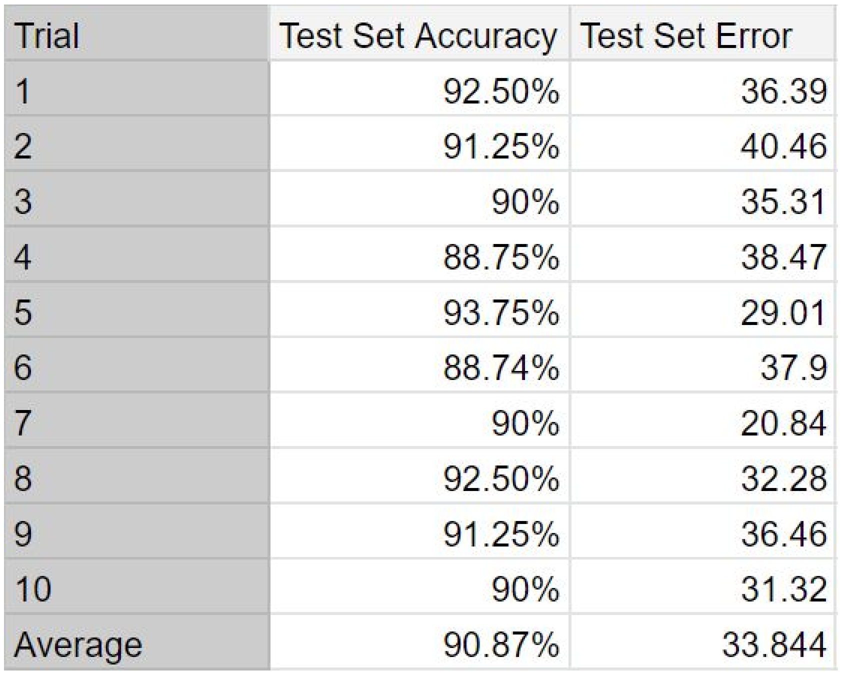
```
Performing 10 updates using Doug's Momentum...
__Update____Error___UnitCost__Wgt.Cost__Grad.Lin__TimeUsed__TimeLeft__
     81)    2.72035   0.00000   354.028  -0.09713        0s         0s
     82)    2.55983   0.00000   358.525   0.01570        0s         0s
     83)    2.45668   0.00000   362.962  -0.08590        0s         0s
     84)    2.31673   0.00000   367.449  -0.02054        0s         0s
     85)    2.20771   0.00000   371.899  -0.06018        0s         0s
     86)    2.13005   0.00000   376.257  -0.15695        0s         0s
     87)    1.99655   0.00000   380.625   0.03474        0s         0s
     88)    1.94242   0.00000   384.923  -0.24437        0s         0s
     89)    1.81661   0.00000   389.280   0.17380        0s         0s
     90)    1.80734   0.00000   393.501  -0.34799        0s         0s
Performed 10 updates
Total time elapsed: 0.045 seconds
```

As such, I decided to fix the learning rate at 0.1. To decide on how many hidden units the network should have, I trained the network with different numbers of hidden units for 3 trials each, including two extremes of 25 hidden units and also completely removing the hidden layer altogether. I recorded the number of epochs it took for the network to settle at a stable state by monitoring the decrement of training error and recording the cross-validation error and classification accuracy after reaching stable state. In some cases, the network did not settle at a stable state, but in all of these cases, the training error simply fluctuated within a range of less than 3. I recorded these fluctuations as “No Stable State” in the table below and recorded cross-validation error and accuracy 10 epochs after the fluctuations first began. Lastly, on each trial, we also examined the cross-validation error every 10 epochs and recorded the number of epochs it took this error to reach a minimum before increasing, which in every trial occurred before the network trained to a stable state. This is the “Epochs to Minimum CV error” in the table.



Hidden Units	0 (No Hidden Layer)	2	4	6	8	10	12	25
Epochs to Stable Configuration								
Trial 1	70	No Stable State	120	120	120	90	90	80
Trial 2	90	No Stable State	120	100	100	90	100	90
Trial 3	No Stable State	No Stable State	120	120	110	90	100	No Stable State
Average	80	N/A	120	113.3333333	110	90	96.66666667	85
Epochs to Minimum CV Error								
Trial 1	30	50	60	50	40	30	30	40
Trial 2	30	60	40	50	40	40	40	40
Trial 3	40	70	60	50	50	30	30	40
Average	33.33333333	60	53.33333333	50	43.33333333	33.33333333	33.33333333	40
Final CV Error at Stable Configuration								
Trial 1	91.74	39.91	56.17	48.18	55	52.04	56.35	51.8
Trial 2	102.64	37.65	49.36	44.84	61.27	51.34	52.47	67.67
Trial 3	70.58	39.33	48.3	49.3	56.9	61.47	52.22	57.27
Average	88.32	38.96333333	51.27666667	47.44	57.72333333	54.95	53.68	58.91333333
% Correctly Classified								
Trial 1	85%	87.50%	88.75%	91.25%	88.75%	90%	88.75%	91.25%
Trial 2	86.25%	91.25%	86.25%	91%	87.50%	90%	91.25%	90%
Trial 3	86.25%	88%	90%	91.25%	85%	91.25%	88.75%	91.25%
Average	86%	88.75%	88.33%	91.25%	87.08%	90%	89.58%	90.83%

Ultimately, I decided that it was most optimal to have 6 hidden units in the network, even though the final cross-validation error for 2 hidden units was notably lower. I will discuss this later. I then retrained the network for 10 trials, each for 120 epochs using 6 hidden units, and tested training using the 80 test examples. The table below displays the test set classification accuracy and error for the 10 trials:



Trial	Test Set Accuracy	Test Set Error
1	92.50%	36.39
2	91.25%	40.46
3	90%	35.31
4	88.75%	38.47
5	93.75%	29.01
6	88.74%	37.9
7	90%	20.84
8	92.50%	32.28
9	91.25%	36.46
10	90%	31.32
Average	90.87%	33.844

Discussion

During cross-validation optimization to determine how many hidden units to include, an initial thing to note is that classification accuracy was not affected by the number of hidden units (aside from removing the hidden layer). In fact, the percentage of cross-validation examples classified correctly was consistently around 90% for every single trial, even for the three trials using an extreme of 25 hidden units. This is partially expected for two reasons. First off, people tend to be fairly consistent in their own typing patterns. It is difficult for the network to overfit the data because the cross-validations examples are essentially embedded within the set of training examples in n-dimensional space. Secondly, by including the SOFT_MAX parameter, we forced the network to normalize the sum of outputs to 1. For a single cross-validation input, if the network would have originally generated low activations for each output node, these activations would then be scaled toward 1 so that the most likely output node would be highly activated. Essentially, since we are forcing the network to make a four-way probability distribution between the four individuals, overfitting training data will still result in the most probable outcome being activated.

The motivation behind training the network without a hidden layer is to see the effect of underfitting the data. If including many hidden units still results in a reasonable classification accuracy, would including none do the same? In the table, we see that the cross-validation error was significantly higher and accuracy was slightly lower when the hidden layer was removed. This means that the data is not linearly separable and we still need a hidden layer to give the network enough complexity to successfully train.

Although including 2 hidden units actually resulted in lower training error than 6 hidden units, I still ended up choosing to use 6 hidden units in the final network. The reason for this is that training with 2 hidden units never resulted in the network training to a stable state. This is a sign that when testing the network, there would have been a lot of variability in weight configurations. Furthermore, even though the cross-validation error is lower, the classification accuracy is slightly lower than when using 6 hidden units. For a classification oriented task, having accurate classification is more significant than minor differences in absolute error and stable training is more valuable for consistency.

In the 10 trials of testing on a 6 hidden unit network, classification accuracy again remained close to 90%, while testing error was significantly lower than that of the cross-validation set. It is probably by chance that the test set was more consistent with each individual’s keystroke than the cross-validation set. Although an average classification accuracy of 90.87% may seem rather low for practical applications of keystroke authentication, for the purposes of this study, it is high enough to say that the network correctly classifies most instances of keystroke data from the four individuals and has trained successfully. Moreover, compared to the 92.60% accuracy Muliono et al. achieved through a deep network, 90.87% seems to be an acceptable result for a network with only one hidden layer. The next simulation tests extracted keystroke data on this same exact network.

Simulation 2: Extracted Keystroke Classification

Methods

The same four participants were then asked to type a sentence with the previous password “hatyellow” encoded inside. More specifically, the target sentence was “halo yells at the party show”. Not only does this sentence contain every character that “hatyellow” contains, it also contains every letter transition as well. Below are some examples to clarify where these encoded transitions are:

Transition in “hatyellow”	Transition in “halo yells at the party show”
“**[ha]**tyellow”	“**[ha]**lo yells at the party show”
“h**[at]**yellow”	“halo yells **[at]** the party show”
“hatye**[ll]**ow”	“halo ye**[ll]**s at the party show”
“hatyell**[ow]**”	“halo yells at the party sh**[ow]**”

Note that the transitions in the sentence do not necessarily appear in the same order as they appear in the password (e.g. **[ll]** is before **[at]** in the sentence). However, the starting and ending transitions **[ha]** and **[ow]** of “hatyellow” are purposely the starting and ending transitions of “halo yells at the party show”. This was to account for variation in keystroke timing that potentially results from starting and ending typing tendencies. For instance, individuals might type a sequence of letters differently if the sequence starts off the string they are typing versus if the sequence is in the middle of the string.

Each participant was again asked to practice by typing the sentence, but for 3-5 times (as opposed to 15 or more times in the previous simulation). This was only to get the participants familiar with the sentence they were expected to type, rather than get them familiar with typing the sentence itself. The generalized idea is to be able to extract a user’s keystroke data for a password based on other strings they have typed, which they are not necessarily familiar with typing compared to a personal password.

To extract keystroke data for “hatyellow” from the sentence, we first collect keystroke data for the sentence in the same way as the first simulation. Since “halo yells at the party show” is 28 characters long, each typing of it corresponds to a 28 + 27 = 56 feature input vector, the first 28 which correspond to *dwell time* of individual letters and the last 27 which correspond to *flight time* between letters. We then reduce the dimension of this vector to 17 by only considering the features that correspond to features of the “hatyellow” input vector. For the last 8 features which are supposed to be *flight time* of letter transitions, we take the corresponding *flight time* from the typed sentence. For the first 9 features, which are letter *dwell time*, we had 2 choices for letters that were not the first or last. Consider the letter y in “hatyellow” and the transitions it is associated with in the sentence:

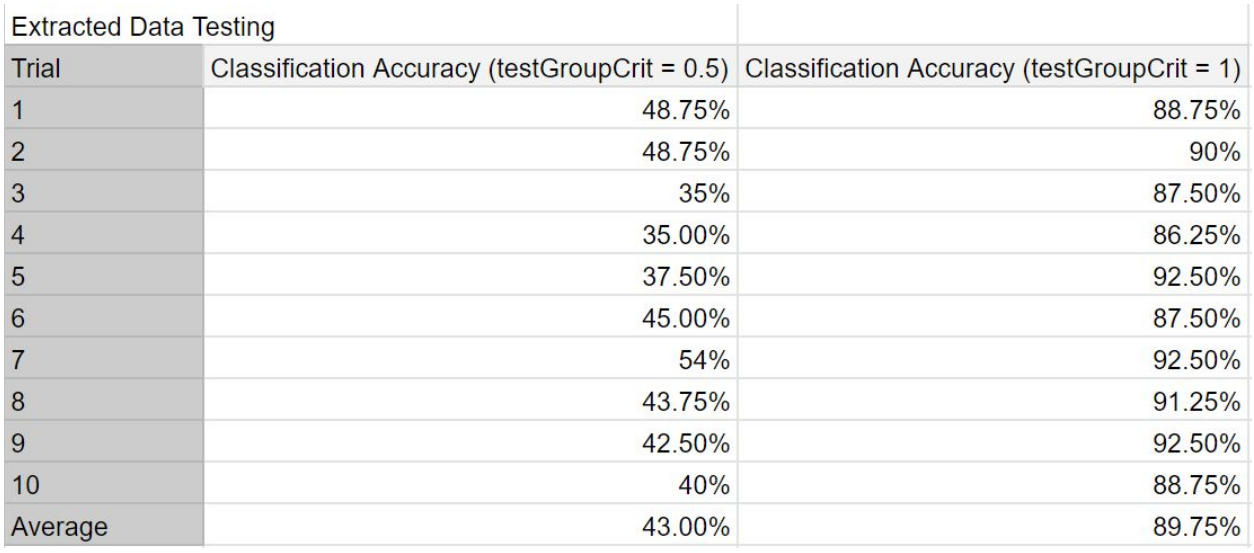
1) “halo yells at the par**[ty]** show” (ha**[ty]**yellow)
2) “halo **[ye]**lls at the party show” (hat**[ye]**llow)

As shown, the letter y appears once in each of the two letter transitions it is associated with, so we had two choices for which y to use for *dwell time*. We ended up using the second case or more generally, if we want to extract the *dwell* time for the letter b in “abc”, we take the *dwell time* from the b in the sentence that appears in the transition “bc”, not the one that appears in the transition “ab”. The reason for associating *dwell time* with the succeeding letter is because logically, the amount of time an individual spends holding a key is impacted heavily by how long they take to reach the next key on the keyboard. The preceding key is presumably rather independent of *dwell time*.

After collecting 20 instances of data from each individual, for a total of 80 extracted test examples, we tested classification accuracy using the network we trained in the first simulation.

Results

Using testGroupCrit = 0.5 and testGroupCrit = 1.0, we tested the network for 10 trials each. The table below shows the classification accuracies:



Extracted Data Testing		
Trial	Classification Accuracy (testGroupCrit = 0.5)	Classification Accuracy (testGroupCrit = 1)
1	48.75%	88.75%
2	48.75%	90%
3	35%	87.50%
4	35.00%	86.25%
5	37.50%	92.50%
6	45.00%	87.50%
7	54%	92.50%
8	43.75%	91.25%
9	42.50%	92.50%
10	40%	88.75%
Average	43.00%	89.75%

Discussion

When using testGroupCrit = 0.5 (the same value in simulation 1), the classification accuracy was drastically lower than the 80 test examples for the actual typing of “hatyellow”. This is expected, as individuals reasonably have different typing tendencies when typing a password they are familiar with as opposed to a longer, less-familiar sentence. However, when testing with testGroupCrit = 1, the classification accuracy increased to 89.75%, which rivals the 90.87% accuracy previously achieved. When testGroupCrit was set to 0.5, we constrained the network so that a correct classification also has to have the correct output activated above 0.5, but now, as long as the correct output node has the highest activation, it counts as a correct classification no matter how small the magnitude of activation is. What this shows is that if the network is forced to make a decision about which individual the extracted data belongs to, it essentially performs as well as classifying actual keystroke data for the password. However, the low classification accuracy for testGroupCrit = 0.5 shows that even though the network can make the correct decision, it is extremely unconfident in its decision and the generated probabilities are extremely dispersed among each of the four participants.

General Discussion

In training a feedforward network with one hidden layer, it becomes apparent that a deep network is not essential for a simple keystroke classification task. However, considering that only four participants were included in this study, it is likely that a greater number of individuals would require more hidden layers for successful classification. For the purpose of this study, the simple network that was trained was also sufficiently successful in testing sentence-extracted keystroke data in Simulation 2. Even though the 43% classification accuracy is extremely low, a completely random classification would result in 25% classification accuracy which means that the network is able to classify extracted input to a notable extent. The high classification accuracy of 89.75% when the threshold activation for the correct output was removed shows that a more complex network would likely be able to make more confident classifications.

The implications of these two simulations give insight to potential future developments in keystroke biometrics. To solve the problem of obtaining external user keystroke data for typing a specific password, it might be possible to extract keystroke data from different strings that other users type. If this type of data generation is successful, it can potentially make keystroke user authentication more powerful than any other security measure used today and completely revolutionize internet security.

References

[1] Keszthelyi, András. “About Passwords.” *Acta Polytechnica Hungarica*, vol. 10, no. 6, 2013, doi:10.12700/aph.10.06.2013.6.6.

[2] “Keystroke Dynamics.” *Biometrics*, 2016, www.biometric-solutions.com/keystroke-dynamics.html.

[3] Muliono, Yohan, et al. “Keystroke Dynamic Classification Using Machine Learning for Password Authorization.” *Procedia Computer Science*, vol. 135, 2018, pp. 564–569., doi:10.1016/j.procs.2018.08.209.

[4] Symanovich, Steve. “What Is Personally Identifiable Information (PII)?” *LifeLock Official Site*, 6 Sept. 2017, www.lifelock.com/learn-identity-theft-resources-what-is-personally-identifiable-information.html.