Contrastive Learning for free Keystroke dynamics

Alessandro Torri
torri.1835715@studenti.uniroma1.it

28/01/2023

Contents

Abstract

The way in which we type on a keyboard is believed to be unique; based on this belief, recent studies have started to focus on the keystroke dynamics as a behavioural biometric trait useful for verification or identification systems. In this paper, we will examine the capabilities of deep learning models to extract (using contrastive learning) discriminative features from free-text typed sequences.

1 Introduction

The main idea of this work is to exploit the power of deep architectures to enhance the discriminativeness of keystroke dynamics. To have a clear look at the work that has been done, it is important to explain the concepts and ideas applied.

1.1 A closer look at keystroke dynamics

Keystroke Dynamics is a behavioural biometric trait which uses the information about the user's typing style to recognize him/her.
In order to do that, we can consider three different kind of data that the user leaks while typing a sequence:

1. Timing data, metrics based on timestamps of pressing and releasing events;
2. Pressure data, the pressure that the user impresses when he/she types each key;
3. Coordinate data, the position where each key has been pressed.

Furthermore, if the user is not forced to type a specific sequence (both in the enrollment phase and in the recognition one) we talk about free keystroke dynamics. In particular, in this work I've used timing data obtained by free keystroke dynamics.

1.1.1 Keystroke Dynamics as a biometric trait

Let's consider the characteristics that a good biometric trait should have:

- Universality, everyone capable of typing on a keyboard can submit a sample; so, elder people (or babies) might not be able to expose this trait.
- Uniqueness, since the typing style is believed to be unique this is a quite good characteristic.
- Collectability, it is quite easy to acquire the timing data of the typing style of a person.
- Acceptability, it is not really a problem for people to write something on a keyboard (they already do that for simple passwords).
- Permanence, this is an issue for all behavioral biometric trait, since people's skills (including the typing style) might change due to variations like time and emotion.

It is also important to mention that keystroke dynamics is not a trait that can be used for scenarios like ”black lists”; it is indeed very easy for the user to alter his typing style in order to avoid being recognized as himself. So, a correct usage of this trait is in a scenario in which the user has an advantage of being recognized correctly (like in ”white lists”).
To summarize, keystroke dynamics is a good biometric trait (it is easy to collect and quite discriminative) but we must be aware that it may lack permanence if used for a quite large amount of time and, since it is weaker to camouflage than to spoofing, we should use it in a white list setting.

1.1.2 The timing features

For each key in the sequence, we can acquire different timing features:

- Hold Latency (HL or Dwell time), elapsed time between the press and release event on a key;
- Inter-key Latency (IL or Waiting time), elapsed time between previous key release and actual key press events;
- Press Latency (PL or Pressing interval), elapsed time between two consecutive press events;
- Release Latency (RL or Releasing interval), elapsed time between two consecutive release events;
- Double-typing time (DTT) , elapsed time between previous key press and actual key release events;

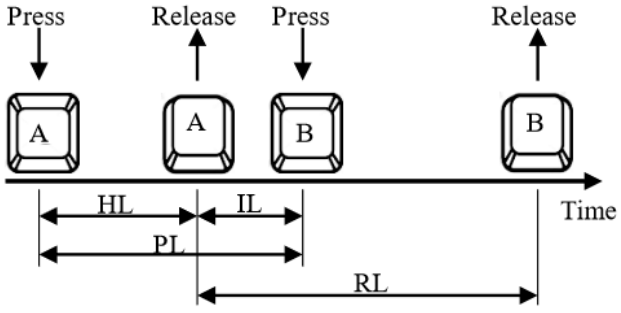

Figure 1: Figure showing the possible timing features

However, I decided to use only Hold Latency(HL) and Inter-key Latency(IL), since the other features are just a combinations of these two (as it can be clearly seen in Figure 1) and therefore a deep learning model can easily understand this dependence and simply ignore the other metrics.

1.2 Self-supervised and contrastive learning

Self-supervised learning(SLL) is a new technique for training neural networks that deals with the lack of sufficiently large labeled datasets. The main idea of SLL is to train a deep model on a so called pretext task, that is a very general task for which we have no need to label the data (the label can be obtained directly from the data itself); When the model is trained on the pretext task, it should learn a meaningful representation space for the data, and this space can then be used for a downstream task (e.g. Image classification).
Contrastive learning is a particular way of doing SSL in which you train your representation space to "attract" different instances of the same object and to "repel" different objects. In our context, what contrastive learning will do is to attract sequences typed by the same user and to repel sequences from different users. The hope is to obtain a space general enough that will allow new users to be enrolled in the system without having to re-train the model and without losing performances.

2 Related work

I became interested in keystroke dynamics about one year ago, where I trained some machine learning models (SVM,Random forest and so on) on fixed-text keystroke dynamics with a small dataset collected manually; however, after improving my knowledge in deep learning and approaches to deal with sequences (RNNs, Transformers), I decided to start a project for free-text keystroke dynamics and the work of this paper takes strongly inspiration from the paper about Typenet[1], which I wanted to replicate by adding some personal novelties. Thankfully, the dataset used in the paper is also publicly available [3] and it has a very huge amount of data to work with. Later on, a new paper about Typenet[2] has been published, with new datasets used in addition and other kinds of training strategies. For other related works I suggest to read the Typenet[2] paper.

3 The dataset

Let's now talk about the dataset[3]. The AAlto dataset has data from 168K different users and for each of them has 15 sequences registered; the data of each user is stored in a dedicated csv file that has one line per press-release event.

Row	PARTICIPANT_ID	TEST_SECTION_ID	SENTENCE	USER_INPUT	KEYSTROKE_ID	PRESS_TIME	RELEASE_TIME	LETTER	KEYCODE
	Int64	Int64	String	String	Int64	Int64	Int64	String15	Int64
1	2200	22307	What are the units?	What are the units?	1063868	1471961564282	1471961564656	SHIFT	16
2	2200	22307	What are the units?	What are the units?	1063875	1471961564552	1471961564657	W	87
3	2200	22307	What are the units?	What are the units?	1063882	1471961564912	1471961565000	h	72
4	2200	22307	What are the units?	What are the units?	1063888	1471961565001	1471961565120	a	65
5	2200	22307	What are the units?	What are the units?	1063891	1471961565272	1471961565336	t	84
6	2200	22307	What are the units?	What are the units?	1063897	1471961565337	1471961565424		32
7	2200	22307	What are the units?	What are the units?	1063931	1471961565480	1471961565584	a	65
8	2200	22307	What are the units?	What are the units?	1063937	1471961565656	1471961565720	r	82
9	2200	22307	What are the units?	What are the units?	1063943	1471961565696	1471961565808	e	69
10	2200	22307	What are the units?	What are the units?	1063951	1471961565809	1471961565872		32

Figure 2: Figure showing how a csv of a user looks like.

In order to use the dataset, a lot of data cleaning and pre-processing had to be applied and since this computation took a lot of time in Python, I did in the Julia programming language to speed up the process. What I did in this phase was to clean some files that had some "\n" characters not needed and to change the csv file to make it have a sequence per row; In each row, there is a column called TIMINGS that is basically a list of tuple $(char, keycode, HL, IL)$ that contains respectively the character pressed, the javascript keycode of the character (useful in case $char$ is missing), the hold latency and the inter-key latency (both in ms). The pre-processing on this phase can be seen in the following Colab notebook.

Row	PARTICIPANT_ID	TEST_SECTION_ID	SENTENCE	USER_INPUT	TIMINGS
	Int64	Int64	String	String	Any
1	107740	1175618	Jones executive vice president and chief operating officer.	Jones executive vice president and chief operating officer.	Any[(String7("SHIFT"), 16, 520, 0), (String7("J"), 74, 105, -81), (String7("o"), 79, 171, 108), (String7("n"), 78, 138, 13), (String7("e"), 69, 111, -6), (String7("s"), 83, 87, 102), (String7(" "), 32, 139, 65), (String7("e"), 69, 100, 47), (String7("x"), 88, 119, 468), (String7("e"), 69, 115, 122) … (String7("g"), 71, 108, -7), (String7(" "), 32, 148, 19), (String7("r"), 79, 140, 203), (String7("I"), 70, 136, 4), (String7("I"), 70, 76, 103), (String7("I"), 73, 152, 56), (String7("c"), 67, 148, 42), (String7("e"), 69, 239, 79), (String7("r"), 82, 144, -75), (String7("."), 190, 132, 519)]
2	107740	1175514	I think those are the right dates.	I think those are the right dates.	Any[(String7("SHIFT"), 16, 788, 0), (String7("I"), 73, 84, -88), (String7(" "), 32, 107, 135), (String7("t"), 84, 91, 39), (String7("h"), 72, 87, -23), (String7("i"), 73, 115, 156), (String7("n"), 78, 119, 44), (String7("k"), 75, 103, 38), (String7(" "), 32, 132, 76), (String7("t"), 84, 97, -8) … (String7("g"), 71, 108, 68), (String7("h"), 72, 139, 82), (String7("t"), 84, 107, 49), (String7(" "), 32, 172, 81), (String7("d"), 68, 164, 227), (String7("a"), 65, 164, 75), (String7("t"), 84, 135, 106), (String7("e"), 69, 76, 4), (String7("s"), 83, 128, 176), (String7("."), 190, 76, 219)]
3	107740	1175648	Don't forget the wood.	Don't forget the wood.	Any[(String7("SHIFT"), 16, 300, 0), (String7("D"), 68, 131, -103), (String7("o"), 79, 143, 81), (String7("n"), 78, 156, 13), (String7("'"), 84, 103, 1), (String7("."), 186, 199, 164), (String7("BKSP"), 8, 139, 324), (String7("BKSP"), 8, 97, 335), (String7("'"), 222, 108, 444), (String7("t"), 84, 103, -9) … (String7(" "), 32, 103, 43), (String7("f"), 84, 99, 17), (String7("h"), 72, 111, 9), (String7("e"), 69, 111, -27), (String7(" "), 32, 111, 26), (String7("w"), 87, 35, 89), (String7("o"), 79, 91, 152), (String7("o"), 79, 91, 100), (String7("d"), 68, 139, 193), (String7("."), 190, 103, 240)]

Figure 3: Figure showing the transformation after the Julia preprocessing.

In addition to the easier to manage structure of the dataframes, this transformation has also the good properties of making each file have exactly 15 rows (the sequences typed by the user) and the overall size of the dataset is way smaller (since we have compressed a lot of information). Switching back to Python, the timings column can be converted into a python list with the help of a regex and a dictionary to convert keycodes of missing characters into the right character.

	PARTICIPANT_ID	TEST_SECTION_ID	SENTENCE	USER_INPUT	TIMINGS
0	306610	3297345	I didn't hear from Ginger this week.	I didn't hear from Ginger this week.	[(shift, 16, 431.0, 0.0), (I, 76, 158.0, -164....
1	306610	3297487	We haven't made that decision here though.	We haven't made that decision here though.	[(shift, 16, 426.0, 0.0), (w, 87, 197.0, -157....
2	306610	3297580	The team mate says were not interested at this...	The team mate says were not interested at this...	[(shift, 16, 348.0, 0.0), (t, 84, 160.0, -119....
3	306610	3297600	But both reports were denied by the southern l...	But both reports were denied by the southern l...	[(shift, 16, 463.0, 0.0), (b, 66, 196.0, 151.0...
4	306610	3297318	Can you rough out a slide on rating agencies?	Can you rough out a slide on rating agencies?	[(shift, 16, 116.0, 0.0), (shift, 16, 456.0, 6...

Figure 4: The final pandas dataframe used in the Python colab notebook.

4 Experimental setup

4.1 Pytorch Dataset

As it has been done in [1], the training set is composed only by the first 68K users while the remaining 100K can be used for testing (so there is no overlap). For our contrastive purpose, the dataset has then been converted into a Pytorch dataset that is composed of all possible couples of samples and the ground truth is simply whether or not the couple comes from the same user. However, with $n = 68K$ users, we have:

$$\binom{n}{2} = \frac{n(n-1)}{2} = \frac{68000 * 67999}{2} = 2,3 * 10^9 \tag{1}$$

that is billions of couples. Since Colab (and my laptop too) have not enough memory to store this dataset, I had to implement it in a lazy way, meaning that the couple is loaded from disk whenever it is requested. However, there is still a time issue; training such big dataset took an endless time just to train one epoch; therefore, as it was also done in [1], I limited the training to 150 batches per epoch and, to avoid training always on the same data, I implemented the dataset to return a random couple that comes from the same user with probability ρ(hyperparameter).
Finally, a small percentage of users have been used as validation set and, since it contained a small enough number of samples, I implemented the validation set not in a lazy way.

4.2 Contrastive loss

Given two sequences (x_i, x_j), we will extract their features $(f(x_i), f(x_j))$ using our deep learning model f. We can define L_{ij} to be 1 if (x_i, x_j) are sequences from the same user, 0 otherwise; the contrastive loss will compute the euclidean distance from the features $d(x_i, x_j) = ||f(x_i) - f(x_j)||$ and then the loss will be:

$$L = L_{ij}\frac{d^2(x_i, x_j)}{2} + (1 - L_{ij})\frac{max^2(0, \alpha - d(x_i, x_j))}{2} \tag{2}$$

As we can see, the loss is basically divided in two complementary parts, one for the positive case (same user, genuine) and one for the negative one (different user, impostor). In the genuine case, the loss penalizes us the more the two sequences are distant in the representation space (and indeed we would like them to be near); conversely, in the impostor case, we have no penalization if the sequences are distant more than α (an hyperparameter of the loss) whereas we are penalized more as the sequences are near each other (notice that the maximum penalty for the impostor case is α^2, in case the distance is 0). It it clear that the role of α is fundamental since it is dictating how much to penalize for the impostor case; in [1] α was set to 1.5 while I've tried different values with their results that will be shown later.

5 The models

To obtain the features of the sequences I've basically implemented two different architectures: an LSTM[5] and a Transformer[6] Encoder. Both of them need as input some embedding of each element of the sequence that I've computed as follows:

1. First of all, I assigned a one-hot index to each character of the training set;
2. Given the triple $(char, HL, IL)$, we take the one-hot index of $char$ and we apply a linear projection to compute a "character embedding";
3. Then, another linear layer is used to project (HL, IL) to obtain a "timing embedding";
4. Both the embeddings are concatenated to obtain the final embedding.

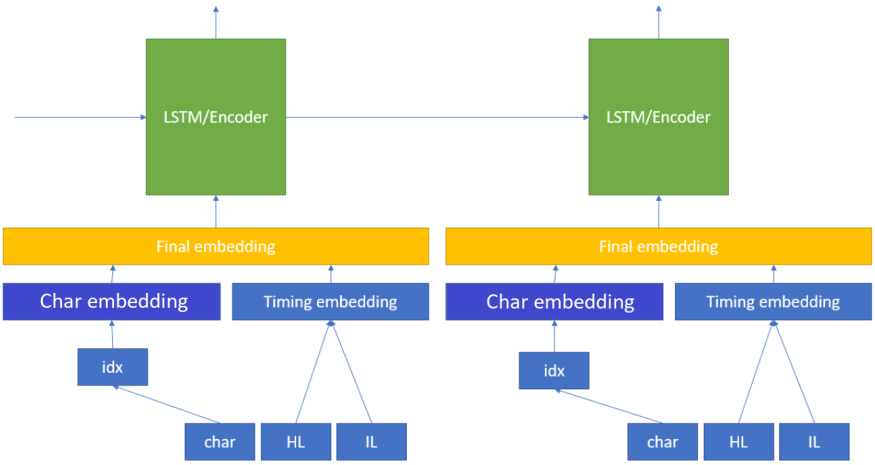

Figure 5: A visual representation of how the embedding of each key is computed.

6 Experiments & results

6.1 The role of α

As discussed in section 4.2, the role of α in the loss changes our training; indeed, the higher is α the higher is the loss; however, we should also remember that the constrastive loss is referring to our pretext task and not to our final objective of recognizing people. Therefore, having an higher loss is not necessary bad, since it may lead to a better space for recognition (our downstream task).

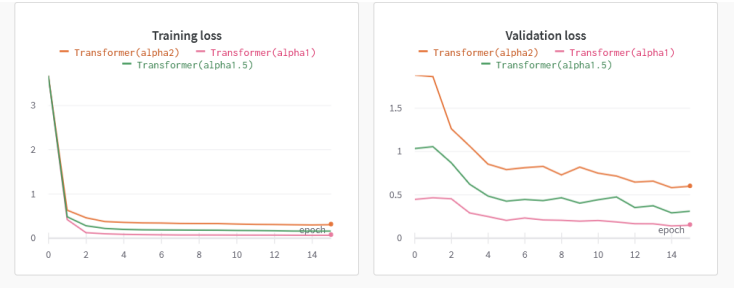

Figure 6: The changes in the loss when varying alpha

6.2 LSTM vs Transformer

Based on what we just said in section 6.1, when comparing two different architectures/set of parameters, it is crucial to do that having the same value for α to have a fair comparison. Even though I have not done an exhaustive number of experiments about that, Transformer architectures seem to have an higher loss (contrary to what I was expecting); however, as we will see later, they seem to produce a better representation space for the downstream task. The reasons for the higher loss may be found in the fact that Transformer architecture need an high number of parameters to perform at their best and they usually need long training times.

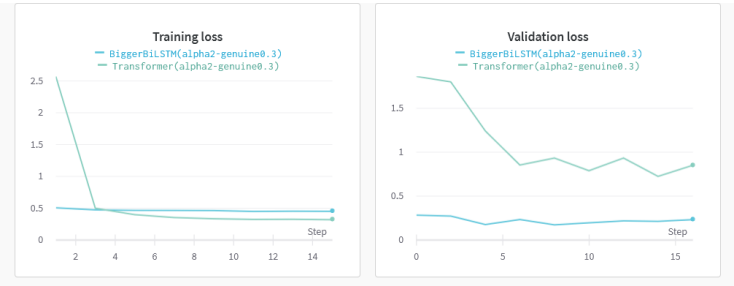

Figure 7: A comparison between LSTM and transformers (both with α = 2).

6.3 Representation space

If we take some test set users sample, extract the features and plot them (using PCA[4] to reduce the dimensionality), we can get an interesting visualization of the representation spaces:

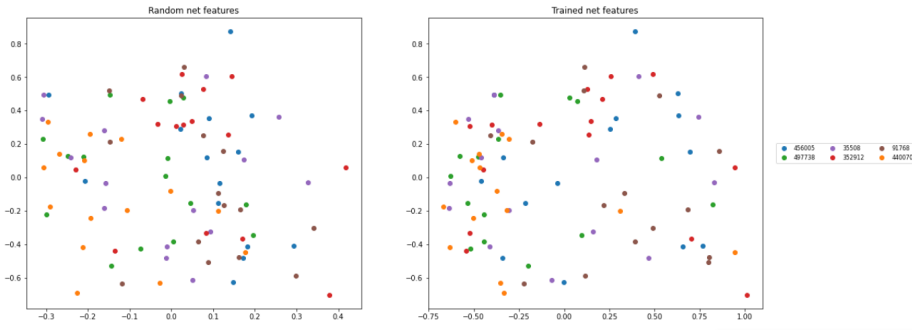

Figure 8: The representation space of the LSTM of Figure 7 (before training on the left and after training on the right).

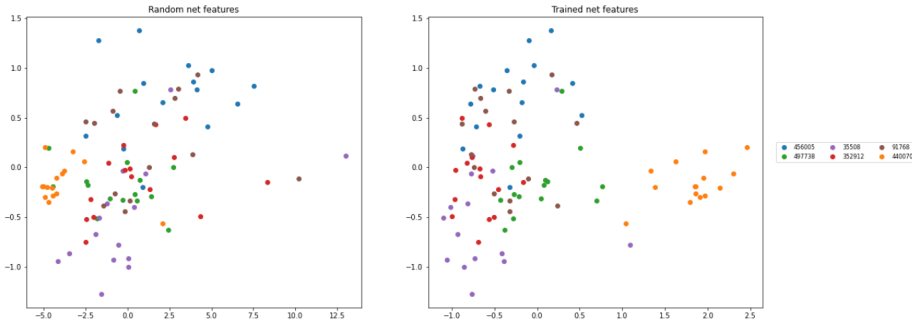

Figure 9: The representation space of the Transformer of Figure 7 (before training on the left and after training on the right).

As we can see in Figure 9, the orange points have been clearly separated from the others by the Transformer representation space (they also seem to be interestingly already near each other in the space of a transformer with random features); for the LSTM, instead, the disorder seem to persist, leading to a bad representation. However, in Figure 9 we can see how is still difficult to recognize other users (e.g. red points and green ones are too near each other).

7 Biometric Evaluation

Once we have our deep model trained, it will act as a feature extractor for a verification system. We will divide our testset into probe and gallery and then test the probe samples both in genuine cases (they claim their true identity) or impostor cases (they claim the wrong identity). For time reasons, I decided to use only 10K of the remaining 100K users for the testset; furthermore, in the impostor case a probe is considered only for one randomly chosen false claim (and not all the other 9999 users). In this way, if we decide to have k samples out of 15 as probes and the remaining (15-k) in the gallery, we will have $10^3k(15-k)$ genuine and impostor scores.

7.1 How the verification system works

The verification process works as follows:

1. Given a sample x_i and a claimed identity id_i, compute the features $f(x_i)$ from the model f
2. Compute the average euclidean distance from the samples with identity id_i in the gallery
$$avg(x_i, id_i) = \frac{\sum_{x_j \in Gallery | id_j = id_i} ||f(x_j) - f(x_i)||}{|\{x_j \in Gallery | id_j = id_i\}|} \tag{3}$$
3. If the average distance avg is higher than a threshold t reject, otherwise accept.

7.2 Evaluation

To evaluate the verification system we will use the standard metrics of Equal Error Rate(EER) and Area under the ROC Curve. To compute them, we have to define a set of thresholds and for each of them compute False Acceptance Rate (FAR) and False Rejection Rate (FRR). FAR is defined as the number of acceptances over all the impostor claims while FRR is defined as the number of rejection over all genuine claims; from what we said before, we have $10^3k(15-k)$ genuine and impostor claims. The set of threshold for which we compute the FARs and FRRs is defined as follows:

$$T = \{t | t = 0 + i\frac{max(distances)}{100} \wedge i \in [0, 100]\} \tag{4}$$

where $max(distances)$ is the maximum distance computed for all the probe/gallery pairs considered. Basically we are dividing the interval in 100 different points (to obtain a more fined grane result you can increase this value); for each of these values, we will compute FAR and FRR as follows:

- $FRR(t) = \frac{|\{p \in Probe | avg(p, id_p) > t\}|}{10^3k(15-k)}$ (id_p is the true identity of the sample p).
- $FRR(t) = \frac{|\{p \in Probe | avg(p, id_{cp}) <= t\}|}{10^3k(15-k)}$ (id_{cp} is the (false) identity claimed by p).

Once we have computed them, the ERR (that ideally is the value where FAR = FRR) can be approximated as:

$$t^* = t \in T | \forall t' \in T \wedge t' \neq t \wedge |FAR(t) - FRR(t)| < |FAR(t') - FRR(t')| \quad (5)$$

$$ERR \approx \frac{FRR(t^*) + FAR(t^*)}{2} \quad (6)$$

The Area under the ROC curve has been computed with sklearn and finally everything is plotted to visually assess the performances.

7.2.1 Transformer performances

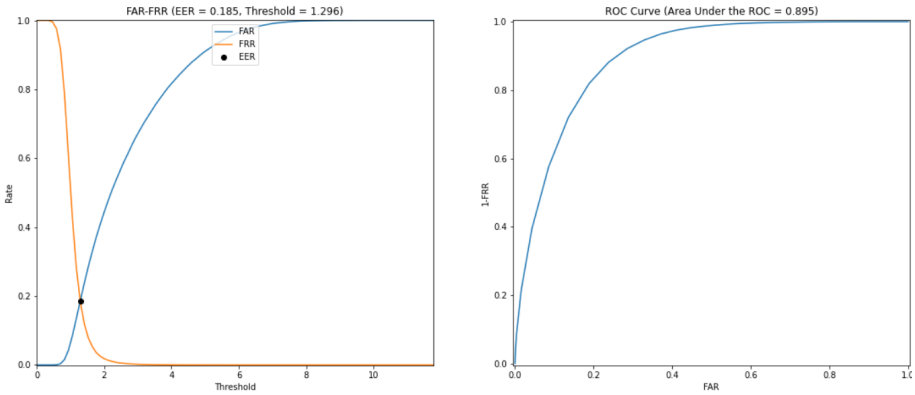

Figure 10: On the left, FAR/FRR plot of the transformer model; on the right the ROC curve.

7.2.2 LSTM performances

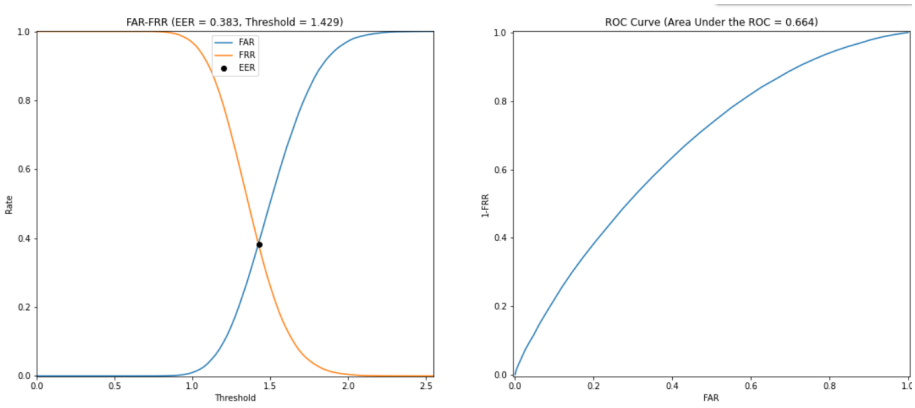

Figure 11: On the left, FAR/FRR plot of the LSTM model; on the right the ROC curve.

8 Conclusions

As we have seen, the Transformer architecture reached better performances in the verification system with an EER of 18.5%, compared to the 38.3% of the LSTM. Still, 18.5 is quite high as an error rate. However, it is anyway good enough to show the potential of this approach that still outperforms some previous works (as we can see in Figure 12 taken from [1]):

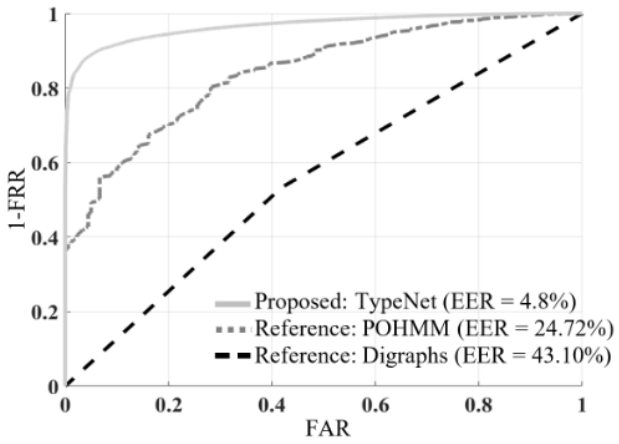

Figure 12: Previous works ROC Curves and EER.

The reason why my performances are lower than TypeNet ones may be:

1. The size of the model, that is unspecified in [1], since of course deeper models and longer training times may lead to better results;

2. Different handling of the sequences embeddings, since the keycodes and the features used are different (they used all the metrics seen in Figure 1

In conclusion, the results obtained are promising and can surely be improved and I reserve to do that in the future; It might also be interesting to work on a dataset that has data acquired in different moments (the time is not specified in the AAlto dataset) to assess how the performances vary with time (since permanence may be a weak point of behavioural traits).
All the code I implemented for preprocessing, training and testing/evaluation is publicly available.

References

[1] Alejandro Acien, John V. Monaco, Aythami Morales, Rubén Vera-Rodríguez, and Julian Fiérrez. Typenet: Scaling up keystroke biometrics. *CoRR*, abs/2004.03627, 2020.

[2] Alejandro Acien, Aythami Morales, John V. Monaco, Rubén Vera-Rodríguez, and Julian Fiérrez. Typenet: Deep learning keystroke biometrics. *CoRR*, abs/2101.05570, 2021.

[3] Vivek Dhakal, Anna Feit, Per Ola Kristensson, and Antti Oulasvirta. Observations on Typing from 136 Million Keystrokes. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*, 2018.

[4] Karl Pearson F.R.S. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[5] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.