MASTER'S THESIS

Interuniversity Master in Statistics and Operations Research UPC-UB

Title: Design, implementation and validation of ML models based on Adversarial Network for generating synthetic biometric behaviour

Author: Idoia Eizaguirre Peral

Advisor: Pedro Delicado Useros, Francesco Zola

Department: Department of Statistics and Operations Research

University: Universitat Politècnica de Catalunya – Universitat de Barcelona

Academic year: 2021-2022

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat de Matemàtiques i Estadística

UNIVERSITAT DE BARCELONA

Abstract

Keywords: Keystroke dynamics, Users Behaviour, Conditional Generative Adversarial Networks

MSC2000: 62M45 - Neural nets and related approaches

Digital security has become crucial in this new era of technology. Behavioral biometrics have been proved to be a robust user authentication system because it provides continuous user authentication. In recent years, keystroke dynamics, a type of behavioral biometric, is gaining popularity in user authentication systems due to the facility of being captured without any special hardware and an absence of the need for an active response of the user. However, its popularity has led hackers (and researchers) to investigate more deeply in how it is possible to fool authentication systems through a presentation attack. A presentation attack is a way to attack a system by using prank biometrics to impersonate a user.

To date, there is no literature on synthetic data generation on keystroke dynamics using Generative Adversarial Networks (GAN), which can be used for replicating presentation attacks. In this study, we pursue a new approach of keystroke dynamics data generation using Conditional Generative Adversarial Networks (cGAN), an extension of GANs. To that aim, two different architectures have been designed, implemented and validated. On the one hand, a "traditional" Vanilla-cGAN in which both subnetworks, the Generator (G) and the Discriminator (D), are Artificial Neural Networks (ANN) and on the other hand, a LSTM-cGAN in which both G and D are Recurrent Neural Networks (RNN) that make use of Long Short-Term Memory units (LSTM) in order to treat data sequentially. Two different real use cases have been considered, the first case, in which the attacker knows exactly the order of the word to be typed for replicating the user behavior, and the second case, in which the order of the words in the phrase is not known. Such use cases have been used for implementing and validating the models in real scenarios.

In order to evaluate the data generated by these models, the accuracy of the internal discriminator (D) has been measured during the training of the models and with a prefixed stopping criteria. A behavioral classifier trained with real data and based on a Siamese Network has been used to validate the synthetic behavioral data. This Siamese Network is trained to detect whether two keystroke dynamics belong to the same user or not.

In what keystroke dynamics data generation concerns, promising results have been obtained with a high performance at impersonating a user. However, the Vanilla-cGAN required much more training than the LSTM-cGAN to get realistic synthetic data. On the report of the results, the Vanilla-cGAN performed better than the LSTM-cGAN according to the external model. Nevertheless, when the stopping criteria is removed and the LSTM-cGAN is trained more, it is more successful than the Vanilla-cGAN. Consequently, if LSTM cells are used to treat keystroke dynamics as sequential data, even more realistic synthetic data can be obtained. In addition, architectures with LSTM units have shown to need less training to reach better results.

Abstract

Key words: Dinámica de tecleo, Comportamiento de usuarios, Redes Generativas Adversarias Condicionadas

MSC2000: 62M45 - Redes neuronales y enfoques relacionados

La seguridad digital se ha vuelto crucial en esta nueva era de la tecnología. La biometría de comportamiento ha demostrado ser un sistema de autenticación de usuarios muy robusto ya que proporciona una autenticación continua de los usuarios. En los últimos años, la dinámica de tecleo está ganando popularidad en la autenticación de usuarios gracias a la facilidad a la hora de capturar los datos al no necesitar de ningún tipo de hardware especial ni respuesta activa del usuario. Sin embargo, su popularidad ha llevado a hackers (e investigadores) a investigar más detenidamente cómo es posible engañar a los sistemas de autenticación de usuarios mediante un ataque de presentación. Un ataque de presentación es una forma de atacar un sistema utilizando biométricos falsos para suplantar la identidad de otro usuario.

Hasta la fecha, no hay bibliografía sobre la generación de datos sintéticos de dinámicas de tecleo mediante Redes Generativas Adversarias (GAN), los cuales pueden ser utilizados para replicar ataques de presentación. En este estudio, seguimos un enfoque de generación de datos de dinámica de tecleo utilizando Redes Generativas Adversarias Condicionadas (cGAN), una extensión de las GANs. Para ello, se han diseñado, implementado y validado dos arquitecturas distintas. Por una parte, un "tradicional" Vanilla-cGAN en el cual las dos subredes, el Generador (G) y el Discriminador (D), son Redes Neuronales Artificiales (ANN), y por otra parte, un LSTM-cGAN en el cual ambos, G y D, son Redes Neuronales Recurrentes (RNN) que utilizan unidades de gran memoria a corto plazo, también conocidas como LSTM. En este estudio, se han considerado dos casos de uso reales: en el primero, el atacante sabe exactamente el orden de las palabras que se escriben para replicar el comportamiento del usuario, en el segundo, el orden de las palabras de la frase no se conoce. Dichos casos se han utilizado para implementar y validar los modelos en un contexto real.

Para evaluar los datos generados por estos modelos, durante el entrenamiento, la precisión del discriminador interno (D) se ha medido con una condición de parada preestablecida. Por otra parte, se ha entrenado un modelo basado en una Red Siamesa con datos reales para clasificar comportamientos y validar los datos de comportamiento sintéticos. Esta Red Siamesa está entrenada para que detecte si dos secuencias de dinámica de tecleo pertenecen a un mismo usuario o no.

En lo que se refiere a la generación de datos de dinámica de tecleo, se han obtenido resultados prometedores con un alto rendimiento en imitar a un usuario. A pesar de ello, el Vanilla-cGAN ha necesitado mucho más entrenamiento que el LSTM-cGAN para obtener datos realistas. Según los resultados de este estudio, el Vanilla-cGAN genera mejores datos que el LSTM-cGAN en cuanto al modelo externo. Sin embargo, cuando la condición de parada se elimina, y el LSTM-cGAN se entrena más, se obtienen mejores resultados que con el Vanilla-cGAN. En definitiva, si las unidades de LSTM se usan para tratar datos de

dinámica de tecleo como datos secuenciales, se pueden obtener datos sintéticos aún más realistas. Además, las arquitecturas con unidades de LSTM han demostrado necesitar menos entrenamiento para llegar a mejores resultados.

Abstract

Key words: Dinàmica de tecleig, Comportament dels usuaris, Xarxes Generatives Adversàries Condicionades.

MSC2000: 62M45 - Xarxes Neuronals i enfocaments relacionats

La seguretat digital ha esdevingut de vital importància en aquesta nova era de la tecnologia. La biometria de comportament ha demostrat ser un sistema d'autenticació d'usuaris molt robust ja que proporciona una autenticació contínua dels usuaris. Els darrers anys, la dinàmica de tecleig està guanyant popularitat en l'autenticació d'usuaris gràcies a la facilitat a l'hora de capturar les dades en no necessitar cap tipus de hardware especial ni resposta activa de l'usuari. Tanmateix, la seva popularitat ha instigat hackers (i investigadors) a investigar de forma més profunda com és possible enganyar els sistemes d'autenticació d'usuaris mitjançant un atac de presentació. Un atac de presentació és una manera d'atacar un sistema utilitzant biomètrics falsos per suplantar la identitat d'un altre usuari.

A dia d'avui, no hi ha bibliografia sobre la generació de dades sintètiques de dinàmiques de tecleig mitjançant Xarxes Generatives Adversàries (GAN), les quals poden ser utilitzades per replicar atacs de presentació. En aquest estudi, seguim un enfocament de generació de dades de dinàmica de tecleig utilitzant Xarxes Generatives Adversàries Condicionades (cGAN), una extensió de les GANs. Amb això, s'han dissenyat, implementat i validat dues arquitectures diferents. D'una banda, un "tradicional" Vanilla-cGAN en què les dues subxarxes, el Generador (G) i el discriminador (D) són Xarxes Neuronals Artificials (ANN), i d'altra banda, un LSTM-cGAN en què ambdós, G i D, són Xarxes Neuronals Recurrents (RNN) que utilitzen unitats de gran memòria a curt termini, també conegudes com a LSTM. En aquest estudi, s'han considerat dos casos d'ús reals: en el primer, l'atacant sap exactament l'ordre de les paraules que s'escriuen per replicar el comportament de l'usuari; en el segon, l'ordre de les paraules de la frase és desconegut. Aquests casos s'han utilitzat per a implementar i validar els models en un context real.

Per evaluar les dades generades per aquests models, durant l'entrenament, la precisió del discriminador intern (D) s'ha mesurat amb una condició d'aturada preestablerta. Per altra banda, s'ha entrenat un model basat en una Xarxa Siamesa amb dades reals per a classificar comportaments i validar les dades de comportament sintètiques. Aquesta Xarxa Siamesa està entrenada per a que detecti si dues seqüències de dinàmica de tecleig pertanyen a un mateix usuari o no.

Pel que fa a la generació de dades de dinàmica de tecleig, s'han obtingut resultats prometedors amb un alt rendiment en imitar a un usuari. Malgrat això, el Vanilla-cGAN ha necessitat molt més entrenament que el LSTM-cGAN per obtenir dades realistes. Segons els resultats d'aquest estudi, el Vanilla-cGAN genera millors dades que el LSTM-cGAN pel que fa al model extern. Tanmateix, quan la condició d'aturada s'elimina, i el LSTM-cGAN s'entrena més, s'obtenen millors resultats que amb el Vanilla-cGAN. En definitiva, si les unitats de LSTM s'usen per a tractar dades de dinàmica de tecleig com a dades seqüencials, es poden obtenir dades sintètiques encara més realistes. A més a més, les

arquitectures amb unitats de LSTM han demostrat necessitar menys entrenament per arribar a millors resultats.

Aknowledgements

First of all, I would like to thank my advisor Francesco Zola for guiding me during the whole project at Vicomtech and for his help, motivation and support, you have incredibly helped me in this successful learning process. Also, I would like thank my director Pedro Delicado from university for his support, advice, for trusting me and for his help with my Catalan skills.

Next, special words for my friend and classmate Leire Agirretxe who has supported me in the last six years me throughout my university career. Same to my parents, to the investment they have made to give me the best for my professional career.

Last but not least, I am really really grateful for my colleagues at my department V5 at Vicomtech, thanks for welcoming me with open arms, making me feel part of you from day one, helping me so much with any inconvenience I could have had and showing me the best *tortilla de patatas* I have ever tried. Additionally, thanks to Vicomtech for giving me an opportunity to work on this project and providing all the equipment I needed.

Thank you.

Contents

List of Figures

List of Tables

Chapter 1
Introduction

In the last decade, with the development of technology and Internet communication, cybersecurity has become crucial for data protection and more and more security methods have arisen for user security, privacy and data protection. Therefore, cyber attackers have also grown significantly, and hence, cyber attacks have become more difficult to detect, face and protect users from.

In this new era, there are millions of users that continuously store personal and intransferable information in the cloud. This progress leads to the need of a user authentication system to enable users access information. The first steps of user authentication were given in the 1960s with the first PIN codes and passwords. However, this authentication has become outdated and with almost every computer it is possible to guess any password in a few hours.

Digital Security is the area that studies the protection of some internet account and files from an attack. Nowadays, internet users make use of security systems in their computers to keep them safe when navigating in the cloud. For businesses, this security is incredibly important because they store important and intransferable information about personal data, behavioural data, financial data etc.

Digital Security refers to a wide range of techniques and processes that are used to protect users from data subtraction, important information theft or cyber attacks. As it is a constantly evolving area, it is necessary to be always updated. However, the biggest threat of a digital security system is the cybercrime. Cybercrime happens when hackers can access and steal information about people or organizations and try to make profit of it. To face this problem, in the last years it has been proved that biometry is a robust mechanism against these type of attacks.

Biometry refers to the measurement taking of living beings or biological processes. The area of research that studies biometry for the unequivocal recognition of people based on one or more features is called biometrics. As mentioned above, biometry is used to register and authenticate users. The main advantage of using it to register and authenticate system users is its universality (all users have), singularity (each

user has its own), permanence in time and context and being measurable quantitatively. When registering a user, first, the measurements have to be captured, then, processed, and last, recorded in some adequate storage unit. To authenticate a user, that is, to verify its identity, first, the user has to be identified to see if the system recognizes the user, and then, the system, by means of biometry, needs to verify that the measurements correspond to the user that is requesting access.

There are several biometric measurements that can be used to authenticate users. These measurements are divided in two groups, on the one hand, there are technologies that use physiological biometrics such as face, iris, fingerprint recognition [**1**], [**2**], [**3**]. On the other hand, there are technologies that use behavioral biometrics like signature, voice, way of walking and keyboard typing among others [**4**] [**5**] [**6**].

Biometrics have come to be a impressively secure authentication system. Thanks to the development of artificial intelligence and machine learning, the use of fingerprints, face recognition and iris recognition are very popular user authentication techniques nowadays. On the contrary, voice recognition and specially keystroke dynamics, are not as popular as the techniques mentioned before but they can lead to promising results in what continuous user authentication systems concern. In this project, behavioural biometrics will be studied, particularly, keystroke dynamics.

Keystroke dynamics refer to how a user types on a keyboard. The rhythm, the speed, the common mistakes and the time spent pressing each key can be used for authentication. It is thought that each user has its own typing ID and it can be very difficult to immitate by another user or by a robot. Keystroke dynamics research is divided into two groups: fixed-text (a fixed input is expected, it is previously known what the user will write) and free-text (the user decides what to write, input is unknown) dynamics. Almost all the literature until the moment has been made with fixed text and to the best of our knowledge, free-text dynamics have not been studied at large scale yet.

In order to study keystroke dynamics, most studies use different models from SVM (Support Vector Machines) and MLP (MultiLayer Perceptrons) to RNN (Recurrent Neural Networks) models from machine learning. However, such models try to directly classify keystroke dynamics of each user. Nevertheless, in this study, our idea is to learn these user behavioural patterns for implementing a Presentation Attack. This kind of attack represents an important risk in biometrics authentication, since it allows hackers to impersonate “genuine” users. In this work, we will implement the keystroke presentation attack using Generative Adversarial Networks (GAN) introduced by Ian Goodfellow in 2014.

Vicomtech

The center in which this project has been developed is called Vicomtech[1], a non profit foundation with approximately 175 workers in which 4 out of 10 have a doctor's degree. Vicomtech is divided into 7 departments: Industry and Advanced Manufacturing, Intelligent Transport Systems and Engineering, Digital Health and Biomedical Technologies, Data Intelligence for Energy and Industrial Processes, Speech and Natural Language Technolgies, Digital Media and Digital Security. The mission of the foundation is to respond to the needs of Applied Research, Development and Innovation in Information Technology.

This work has been done in the Digital Security department in which there are two technological guidelines: Detection&Response and Protection&Trust. In particular, this work is related to the second technological guideline (Protection&Trust). More concretely, this study is supported by the EGIDA project, a national project funded by the CERVERA network of national excellence for information privacy technology research. Its purpose is to use the records of the actions that are captured from a person to verify their identity or detect attempts to impersonate them. Therefore this project, if needed, it can be later used to integrate it in some other project.

[1] `https://www.vicomtech.org/es/`

Chapter 2
Preliminaries

2.1. Presentation attack

In the last decade, user authentication has had a huge development and today, it is much more than password verification [**7**]. Nowadays, biometrics are used for user authentication in very diverse areas like banking, military access control, smartphones and many more [**8**], [**9**]. Biometric techniques that are being used for authentication are face, fingerprint, iris, voice, keystroke dynamics recognition among others. These techniques based on physical and behavioural measurements are proved to be more secure than conventional authentication with PIN codes and passwords. [**10**]. However, there are various ways to attack a biometric system. Presentation attack (PA) is a way to attack a system by using spoof biometrics to impersonate a genuine user (see Figure 1). [**11**].

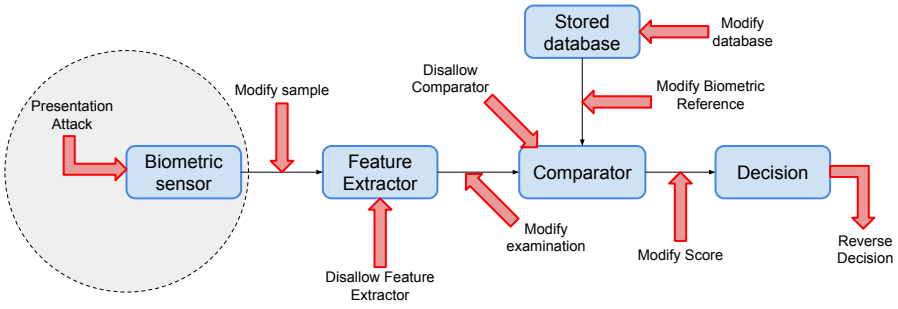

FIG. 1. Types of attacks to a Biometric system

A presentation attack tries to impersonate or obfuscate (hide the user's identity) a user that is authorized to access to the system. These presentation attacks are very common, for example, for a face recognition system the attack can be performed by using makeup [**12**], a mask, a picture or a video of the victim. In the case of fingerprint recognition, a silicon finger or a fingerprint stamp can be recreated

to try to trick the system [**13**]. For voice and keystroke dynamics presentation attacks, synthetic data can be used to impersonate the victim. All these tools used for presentation attack are called Presentation Attack Instruments (PAI) [**7**].

The techniques developed for mitigating presentation attacks are called Presentation Attack Detection (PAD). PAD systems proceed as follows: first, they identify whether the input is real or fake; then, if the system detects that the input is a presentation attack, access is rejected, and if not, the authentication of the user starts. To authenticate a user, the system verifies that the user is registered. If the user's identity is verified, access is accepted, on the contrary, access is denied. Researchers have developed many techniques to develop PAD systems, however, according to Abdullakutty et al. [**7**], it is not possible to know in advance all the attacks that can be performed to an authentication system.

For what concerns PAD in keystroke dynamics, Stefal et al. [**14**] studied the robustness of keystroke dynamics authentication against synthetic falsification attacks by means of a Principal Component Analysis (PCA), SVM, Markov Chains and a remote biometric authentication system called TUBA. However, according to Hazan et al. [**15**], keystroke dynamics do not help preventing replay attack, a type of attack that consists of collecting data and resending it to try to spoof the system by impersonating a user. On the contrary, Yu et al. [**16**] argued that keystroke behavioural biometrics is a feasible method to increase in user authentication system fiability. They obtained promising results when resolving the limitations of a neural network approach that Cho et al. proposed [**17**]. To that end, they used SVM, an extension of these using Genetic Algorithms (GA) and an ensemble creation.

In this project, we propose a new approach to generating keystroke dynamic behaviour to provoke presentation attacks. For the best of our knowledge, this is the first approach for generating keystroke biometric behaviour in an adversarial way.

2.2. Artificial Neural Networks

Artificial neural networks, also known as Neural Networks (ANN) are computational models that are able to reproduce and model nonlinear processes or behaviours. The idea of ANNs began in the 1950s when some researchers wanted to simulate brain behaviour. They belong to deep learning, a subfield of machine learning composed of several algorithms that try to model highly abstract data by means of multiple nonlinear transformations [**18**].

Artificial neural networks are composed of layers: an input layer, one or more hidden layers and an output layer. Each layer is formed by neurons (see figure 2) which combine their inputs in order to return an output. The architecture of ANNs can be changed by the user and it can be specified according to the problem to be addressed. In particular, the user can specify the number of hidden layers, the number of neurons at each layer, the activation function used by each layer, as well as the number of input and output neurons. The aim of ANNs is to obtain the

optimal weight values for each neuron in order to minimize a loss function which will help the model learn nonlinear behaviours of data.

Let us define an ANN with one hidden layer and j neurons that form this layer. At each neuron, information is received and after transforming it with some operations an output is returned. This transformation has three steps. First, the inputs $x_i \ \forall i \in \{1, ..., n\}$ from the input layer are multiplied by their corresponding weight $w_{ij} \in \theta$. Second, they are combined lineally with a transfer function and a bias or intercept x_0, then, $\sum_{i=0}^{n} w_{ij}x_i$. Last, the transformed output is once again transformed by an activation function f and the output of the neuron is obtained, that is, $f\left(\sum_{i=0}^{n} w_{ij}x_i\right)$.

Activation functions, are functions that help transform the output of the transfer function to some restricted space. The most common activation functions are Rectified Linear Unit $(ReLU)$ $f(z) = \max\{0, z\} \in [0, \infty)$, the sigmoid function $(sigmoid)$ $f(z) = \frac{1}{1+e^{-z}} \in (0, 1)$, the hyperbolic tangent function $(tanh)$ $f(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \in (-1, 1)$ and the linear function $(linear)$ $f(z) = z \in (-\infty, \infty)$. However, general recommendation is to use the $ReLU$ activation function due to its properties that help optimize neural networks with methods based on gradient computation. Note that the derivative of the *sigmoid* and the *tanh* activation functions can be computed by the functions themselves.

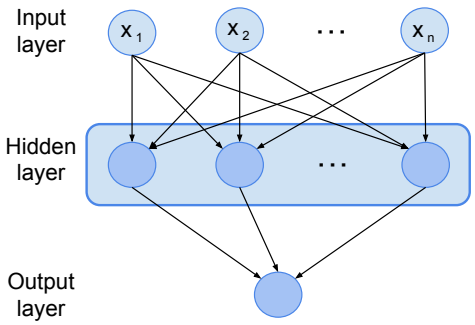

FIG. 2. A Neural Network with one hidden layer.

To optimize the model, the optimal weights need to be computed. These weights are obtained by minimizing a previously fixed loss function that evaluates the prediction error of the whole network. There are some commonly used loss functions already implemented in literature. For binary classification, one of the most common used techniques is the Binary Crossentropy. This loss function, also known as Log Loss, is the one used later in this work and is defined as follows:

$$L_p(\theta) = -\frac{1}{n}\sum_{i=1}^{n} y_i log(p_\theta(x_i)) + (1 - y_i) log(1 - p_\theta(x_i))$$

where y is each observed label and $p_\theta(y)$ is the predicted label for all n observations that depend on the estimated weights $w_{ij} \in \theta$. Note that the value of the loss function increases when the observed label and the predicted label for each observation differ a lot and decreases when observed and predicted labels are similar. Then, when minimizing this function, the similarity between the observed and the predicted labels is maximized.

To minimize the loss function, when the number of weights to optimize is large, the Stochastic Gradient Descent algorithm introduced by Cauchy in 1847 is commonly used [**19**].

This iterative method lies on the fact that the function to minimize is differentiable. Therefore, the partial derivatives with respect to the parameters to optimize can be computed. Let $\theta \in \mathbf{R}^p$ be the vector of the weights to optimize. At each iteration, the algorithm computes a new point by giving a step towards the optimum point as follows:

$$\theta^{k+1} = \theta^k + \alpha^k \delta^k \tag{1}$$

where α is the step-length or learning rate, δ is the direction of movement and k is the iteration number. The direction of movement is obtained from the gradient of the function to minimize, in this case, the loss function $L_p(\theta)$. Note that this gradient measures the change in the output if the input changes slightly.

Then, at each iteration, the partial derivatives of the loss function with respect to each element in the parameter vector θ need to be computed $\frac{\partial}{\partial w_j} L_p(\theta)\ \forall j \in \{1, ..., p\}$. The initial values of these weights are usually randomly chosen by means of a uniform distribution and the biases are initially set to 0.

To compute the direction of movement of each iteration, i.e. the gradient of the loss function, the backpropagation algorithm is used. As mentioned before, this algorithm computes the gradient of the loss function of the neural network with respect to the weights in all the previous layer's neurons in order to minimize the predicted error. This method works by computing the partial derivatives with respect to each weight by applying the chain rule. The most important fact of this method is that calculations are computed effectively to avoid repeated calculations and computing unecessary intermediate derivatives between hidden layers. In addition, the backpropagation algorithm can be written in terms of matrix multiplication and this fact fastens the computation of the new weights.

The Stochastic Gradient Descent algorithm's rate of convergence is linear, hence, it is a slow optimization algorithm. However, researchers have developed extensions of this method to speed up convergence that consist of adapting the step-length or learning rate as convenience, Adagrad, RMSProp and Adam among others ([**20**], [**21**] and [**22**]). The last optimizer is the one used in this work.

Training an ANN might not be trivial. The number of hidden layers, the number of neurons at each layer and the activation functions are a personal choice and the architecture is specific to each problem. Training is done by epochs, a term used in machine learning that makes reference to the number of passes that have been completed using the whole dataset for updating the model's weights. Epochs are divided into batches, subsets of the dataset of a fixed size. It is important to take into account that a very deep ANN can achieve better results but training will be hard due to the difficulties when optimizing the weights. However, according to the Universal Approximation Theorem of Neural Networks [**23**], almost any function can be represented by a Neural Network.

In what applications of ANNs concern, they have been implemented and successfully used in very diverse areas. ANNs have been used in cancer diagnosis [**24**], malware detection in Android apps [**25**], solving partial differential equations [**26**] and game playing [**27**].

ANNs try to predict an output value by analyzing its input information, however, they do not consider the temporality of the input data. What is more, ANNs do not mix information between executions, that is, they do not remember anything about previous executions. To face the limitations of ANNs, researchers developed the Recurrent Neural Networks (RNN).

2.2.1. Recurrent Neural Networks (RNN). Recurrent Neural Networks are another type of algorithm from deep learning that is characterized by treating sequential data efficiently. This efficiency is due to an internal memory unit that stores information about previous outputs and uses them as input. They are able to treat long sequences element-wise. With the rise of deep learning models, RNNs have become very important to analyze sequential data like time series [**28**] and text or speech data [**29**] among others. For example, they are used in voice recognition from large companies like Google Translate and Apple with Siri.

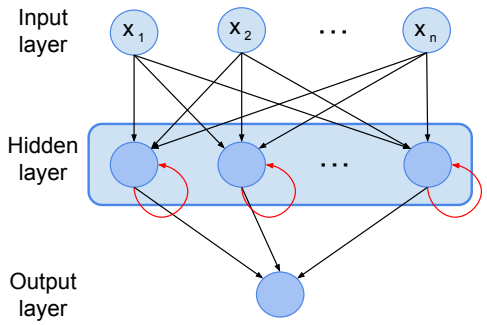

FIG. 3. Recurrent Neural Network

RNNs can be very accurate when making predictions about what is coming next in a sequence because, thanks to their internal memory, they store relevant information about previous inputs. As shown in Figure 3, these networks have got two inputs: the current input of the previous layer (black arrows) and the information about previous inputs (red arrows).

Training a RNN can be expensive because each neuron takes the elements from the input sequence element by element. This process can be simplified by unrolling the network in as many layers as time steps. The longer the temporal sequence the more layers are needed to unroll the network. As a consecuence, when optimizing the weights of the network using backpropagation, depending on the number of time steps the method can become very expensive and time consuming.

As explained previously in Section 2.2, to optimize weights, the gradient of the loss function is needed. If the values of the gradients decrease until they become very small, the network stops learning because the values of the weights do not move towards the optimum values. This problem is called the *vanishing gradient* problem. It consists of having problems when retaining the learned information from previous steps. When using backpropagation, at each neuron, the gradient must be computed to calibrate the internal weights according to Equation 1. However, if the calibrations are too small the next calibrations will be even smaller and therefore the gradient will reduce exponentially leading to a decrease in the long-term memory of the network.

To avoid the problem of the *vanishing gradient*, mechanisms known as Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) can be used. LSTMs and GRUs use a memory cell to store information from previous elements from the sequences.

2.2.2. Long Short-Term Memory unit (LSTM). LSTMs were presented by Hochreiter et al. in 1997 [**30**] in order to face the problem of the *vanishing gradient* efficiently. This gradient based method, uses gates which control the flow of information in the network. A gate is defined as a barrier that controls the entrance to a separated space, therefore, in this context, a gate is a threshold that decides when to store or discard information for the next layers. These gates select important information from each input and store it in the memory cell to later make predictions that take into account information form previous time steps in the sequence.

As mentioned above, the main advantage of LSTM units is that the relevant information from previous inputs from the sequence is stored and used once and again for prediction. To that end, the LSTM cells have a cell state and a hidden state, the part of the LSTM units that are in charge of storing the information, which are updated at each time step of the sequence and decide which information should be saved and which forgotten. The number of sequential LSTM cells can be fixed by the user.

The cell state and hidden state contain the information from previous inputs and at each time step it is joined with the input of that time step and they are updated by pointwise multiplication and addition (element-wise reduction of two vectors in one by multplying or adding respectively), *sigmoid* activation function and *tanh* activation function.

At each LSTM cell, that is, at each time step, the information passes throughout three gates: the forget gate, the input gate and the output gate, see Figure 4.

(1) Forget gate: The input from the corresponding time step joined with the information in the hidden state is transformed by means of the *sigmoid* activation function. In this gate, stored information from previous time steps is filtered in order to only save relevant information and discard the rest.
(2) Input gate: The main aim of this gate is to update the information from the cell state. It filters the information from the combination of the input and the hidden state using the *sigmoid* and the *tanh* activation function separately and combines the results by means of the pointwise multiplication. Last, the output of the pointwise multiplication is used to update the information coming from the cell state using the pointwise addition.
(3) Output gate: It decides which information will be taken to the hidden state of the next LSTM cell. To that end, on the one hand, the combined information from the hidden state and the input is once again filtered using the *sigmoid* activation function. On the other hand, the recently updated cell state is transformed by means of the *tanh* activation function. Last, both outputs are used to obtain the updated hidden state by means of the pointwise multiplication.

LSTM cells avoid the *vanishing gradient* to occur because they avoid the gradient of the loss function being flat and hence the weights will be updated at every iteration of the weight optimization process.

Since they were presented in 1997, LSTMs have been used in many studies, such as in flood forecasting [**31**] and in language modelling [**32**] among others.

2.3. Generative Adversarial Network

Generative Adversarial Networks (GAN) are a type of generative modelling from deep learning introduced by Ian Goodfellow [**33**]. They make use of two Neural Networks that compete with each other during their training: the generator (G) and the discriminator (D). The idea is that, through this adversarial training, both networks improve their performances. The generator's aim is to generate synthetic data departing from normally distributed random noise (z), that is, $G(z) = p(x)$, whereas the discriminator needs to be trained in order to distinguish between real or generated (synthetic) samples (x) (see Figure 5). These networks play a zero-sum or min-max game, that is, the two networks play an adversarial game, one looses when the other wins and the other way round. However, the aim of GANs is

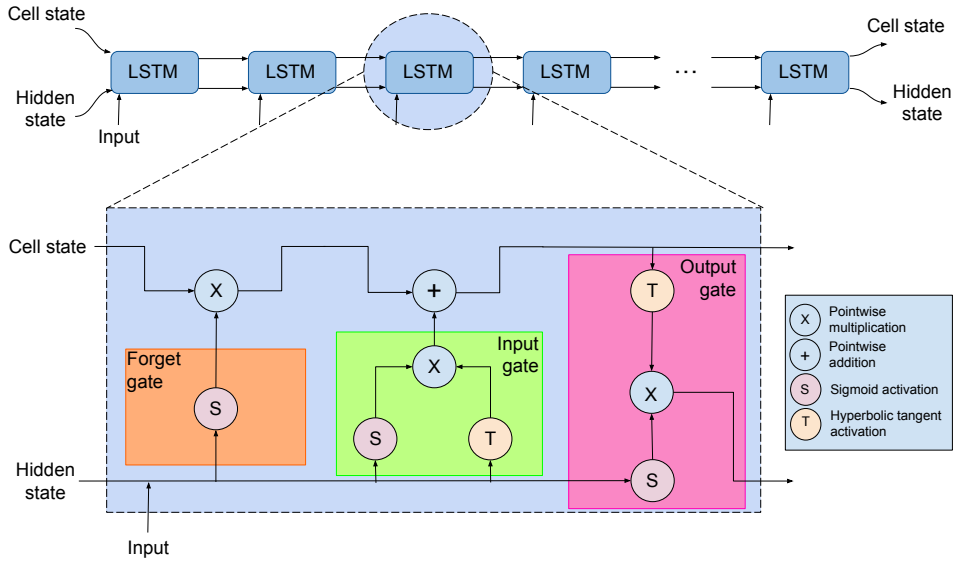

FIG. 4. LSTM cell visualization

to generate synthetic data, by modelling the distribution of the training data and generating new fake samples with a similar distribution.

For generative models, specifically for GANs, the training aim is to reach an equilibrium in the min-max game. This equilibrium, often called Nash equilibrium [**34**], means the whole network is stable, not the discriminator nor the generator wins the game or learns more than the other.

$$\min_{G\in\mathcal{G}} \max_{D\in\mathcal{D}} F(G,D), \text{ where } F \in \mathcal{C}^1 \tag{2}$$

Due to the difficulty of training GANs, it was thought that GANs may not always reach such equilibrium and Farnia et al. have recently obtained theoretical and numerical results that prove this thought [**35**]. In their paper, they propose a new approach, called proximal equilibrium to solve this problem which is based on an extension of GANs known as Wasserstein GAN.

Apart from the previously mentioned *non-convergence problem*, training a GAN may also result in *mode collapse* [**36**]. *Mode collapse* consists of having a generator that generates a subset of all possible outputs. This problem lies on the fact that the generator is unable to focus on the whole data distribution and considers only part of it. Finally, the *vanishing gradient* problem, as its name suggests, occurs when during an optimization of the weights from some model, the value of the gradient gets smaller and smaller and consequently the weights cannot be updated

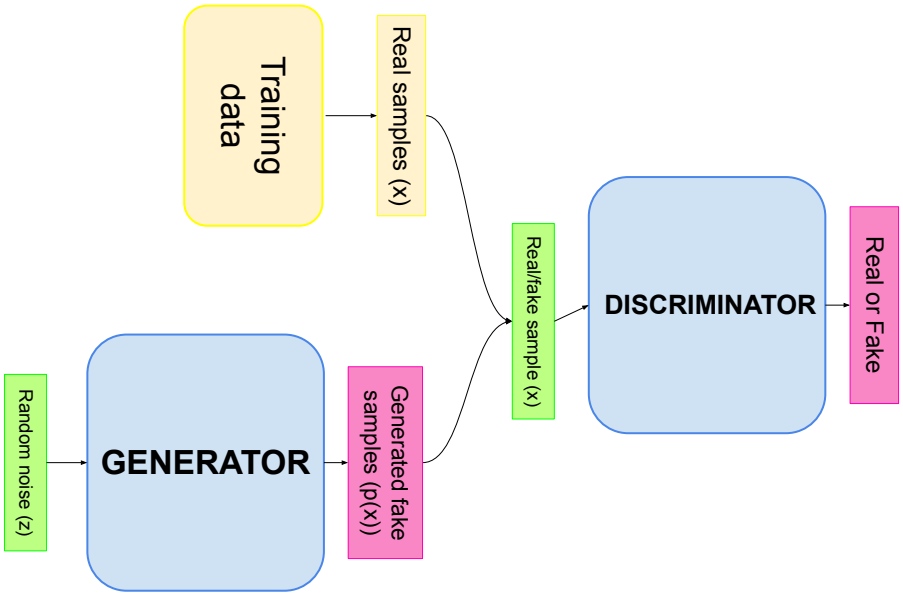

FIG. 5. General Adversarial Network (GAN).

to reach the optimum value. All these common GAN problems are summarized in [**37**].

GANs are widely used to create or generate new samples of data that are as realistic as the samples in the original dataset. Since this new framework was proposed, GANs have been used for very diverse applications and due to the difficulty when training, many extensions of GANs have been developed. They have been used for image creation [**38**] by means of Deep Convolutional Generative Adversarial Networks (DCGAN), data augmentation using Vanilla-GANs [**39**], image-to-image translation by means of Conditional Generative Adversarial Networks (cGAN) [**40**] and CycleGANs [**41**], melody generation from lyrics with Conditional LSTM-GANs [**42**]. What is more, in 2016, Zhang et al. proposed the StackGAN, another extension of GANs that transforms text descriptions to realistic images. In 2017, Karras et al. proposed a new methodology for GANs to improve in training stability and speed it up by means of a Progressive Growing of GANs [**43**].

In the last years, new GAN extensions have been also used to generate synthetic sequential data. In 2017, Esteban et al. proposed the Recurrent GAN and the Recurrent Conditional GAN to generate synthetic multi-dimensional time series [**44**]. The same year, Press et al. used the same GAN extension for language generation with Recurrent GANs that had one GRU layer [**45**]. In 2019, Mogren et al. proposed a way to train RNNs in an adversarial way to generate music that sounded better as the model was trained [**46**]. Among these GAN structures, in this study, we implement the Conditional GAN (or cGAN).

2.3.1. Most common unstable adversarial models. In the previous section the generative adversarial networks that have been trained have been explained in detail. However, as previously mentioned in section 2.3, training generative adversarial networks can be very difficult because convergence or stability might not always be reached. Note that during the training of GANs, both the generator and the discriminator compete against each other to optimize their weights and minimize the loss function. Up to date, there is no literature about any correlation between the loss of the discriminator or the loss of the generator with whether the GAN game is learning to produce realistic samples or not.

The failure of this training procedure can be due to different causes. However, in what solutions concern, there is no universal rule that fits every model. According to Ian Goodfellow [**37**], the tips that are given to solve GAN training problems can be useful for some and harmful for other models. In this section, the three most common problems of training stability will be explained.

1. *Convergence failure.* The failure of convergence is probably the most common failure type when training a GAN. This failure occurs when the generator and the discriminator do not find an equilibrium and the loss of one of them (usually discriminator's loss) decreases towards zero and the other increases nonstop throughout training.

This problem happens when the generator is not able to learn and simulate the training data distribution. The generator generates synthetic samples that can easily be detected as generated by the discriminator and consequently, discriminator's loss falls to zero. However, non convergence can also happen when the generator's loss falls to zero. This means that the generator is fooling the discriminator too easily by generating garbage samples. This non convergence problem might occur at any moment of the training process, however, it usually happens at the beginning of the training and continues diverging until some stopping criteria is satisfied.

Convergence failure might occur because of many different reasons such as, the model architecture is not able to learn the data distribution or the optimizer is too agressive (learning rate of one or both optimizers is too high).

Looking at Figure 6, we can observe that at first it seems that the GAN is converging, but when the training reaches epoch 4000 (note that during training, this plot has been updated every 50 epoch, hence, epoch 4000 is on the value 2000 of the axis of abscissas) we can observe that the generator's loss starts to increase nonstop and the accuracy of the discriminator is very high for both real and fake samples. In addition, it can be observed that the discriminator's loss is almost zero throughout the whole training process. Therefore, we can conclude that the generator is not generating realistic samples to fool the discriminator.

In what the architecture details of the GAN trained in Figure 6 concern, it is important to outline that this model differs on the final model in various aspects. Alike the architecture of the LSTM-cGAN (see Figure 16), the generator and the

discriminator have both two LSTM layers with 400 units each and *tanh* activation function. However, the example presented in this section, has got a dropout layer (rate of 0.4) between the two LSTM layers, the last layer of the generator has got 1575 units instead of 75. In addition, the random noise vector input of the generator receives an array with dimension 50 and it's distribution follows a standard normal distribution. It is important to outline that in this approach, the output of the generator has not the same dimension as the final model. This is due to the attempt to generate the typing times along with the *KEYCODE* and the introduced character embedding. Therefore, in this case, the shape of the generated samples is $(15, 105)$.

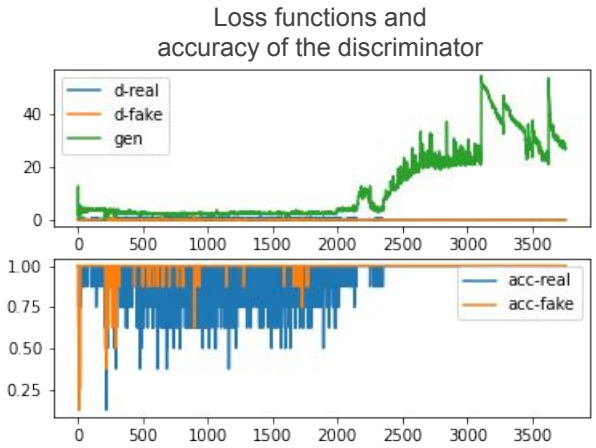

FIG. 6. Example of non convergence in training a GAN.

2. *Mode collapse.* Sometimes, the generator manages to prank the discriminator by generating realistic samples of a subset of all posible samples. Then, the discriminator is fooled and it seems that the adversarial network has reached an equilibrium, however, the generator is not capable of considering the whole data distribution. According to the tutorial presented by Ian Goodfellow in 2016 [**37**], "*mode collapse, also known as the scenario, is a problem that occurs when the generator learns to map several different input z values to the same output point*".

In order to realize that the training is on mode collapse the loss of the subnetworks in the model can be observed, specially the generator's loss. If throughout training time the generator's loss makes jumps or waves, it might mean that it is generating once and again a subset of realistic samples that have different loss values. As the majority of the literature for GANs generates images, another option to visualize if the model has this limitation is to generate a large sample of generated samples and observe whether there is a low variety of different samples or not. Note that in this project, as the generated samples are not samples that can be visually evaluated, in order to identify mode collapse, the plots of the subnetworks' losses are needed.

A factible solution for mode collapse is increasing the latent dimension, that is, increasing the dimension of the random noise vector in the generator's input. Mode collapse might occur because the model cannot generate a wide variety of different samples with a random noise input with a small dimension.

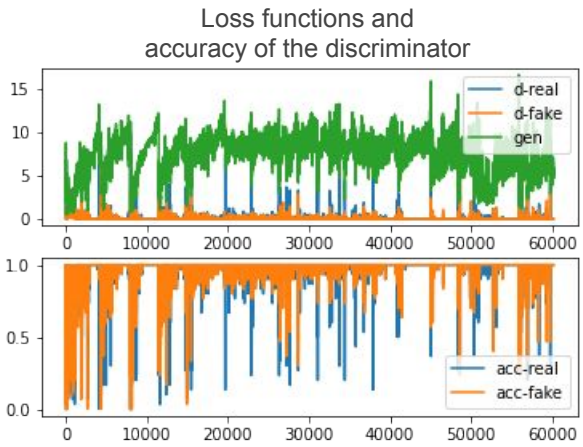

FIG. 7. Example of mode collapse in training a GAN. In this particular case, the model has been trained for 60,000 epochs, the x axis of the plot corresponds to the number of epochs.

Looking at Figure 7, we can observe that the behaviour of the generator's loss function is quite meaningful. It can be observed that the loss of the generator falls down rapidly quite often and then, returns to a higher loss. Whenever the loss of the discriminator increases and the accuracy of the discriminator decreases, the generator's loss decreases momentarily.

In what the architecture details of the presented example concern, this model is similar to the non convergence example presented in the previous failure example. It also has a dropout layer (rate of 0.4) between the two LSTM layers both in the generator and the discriminator with 400 units and *tanh* activation function. On the contrary, the generator receives 100 dimensional input vectors for random noise and the last layer in the generator is a dense layer with 42 units. Note that the output shape of the generator in this case is $(14, 3)$. This difference is based on the fact that, during the first attempts to build the final model, only 2 typing times were considered. Hence, the Conditional LSTM-GAN from this example generated sequences with 14 keystrokes with three features each, *KEYCODE*, *HL* and *IL*.

3. *Vanishing gradients.* In this section, the vanishing gradient problem is explained. Sometimes, the architecture of the discrimininator is optimal or too good, then the gradient value in the backpropagation is very low and the generator learns very slowly or even it does not learn anymore. If this occurs, then the weights of the layers in the generator do not change at all during training, and hence, equilibrium is not reached because the generator cannot learn to fool the discriminator.

Usually, in order to detect the vanishing gradient problem it can be very useful to check the weights. If the weights are decreasing to zero, then the gradient can be vanishing. Also, it can be useful to chech if the loss function's are constant. If so, it might be a sign of vanishing gradient because subnetworks are not learning anymore.

In order to solve this vanishing gradient problem, according to the recommendations made by Ian Goodfellow [**37**], the activation functions can be changed, or the loss function can be changed. There are two loss function changes that are commonly used when dealing with the vanishing gradient which are the Wasserstein loss and the modified minimax loss, both designed to deal with the vanishing gradient.

2.3.2. Tips and tricks to train a stable GAN. According to Soumith Chintala (AI researcher at Facebook) and Ian Goodfellow (ML director at Apple), there are some tips and tricks that can be used to solve the most common GAN training problems. They number several changes that can be made to the model in order to increase the chances to reach nash equilibrium. Among others, they mention the tips listed below. However, it is important to outline, these points are only recommendations that might not work for every model and do not have any theoretical proved background.

- Normalize inputs: Normalize inputs before giving samples to the discriminator between -1 and 1 and use *tanh* in the generator's output layer. To normalize the inputs, it is enough to substract the mean and divide by the standard deviation.
- Flip the target labels. During training, the labels in the discriminator for real samples are 1, and 0 for the synthetic samples. However, to train the generator it is recommended to flip the target of the generated samples, that is, to train the discriminator with a batch of generated samples with label 1.
- Use spherical random noise. To generate random noise input for the generator, generate random samples of a normal distribution instead of a random uniform distribution.
- Train with batches. When training the discriminator, do not mix generated and real samples. It is recommended to train the discriminator with a whole batch of samples from the same group (real or synthetic).
- Avoid sparse gradients. Use softer activation functions such as *LeakyReLu* instead of *ReLu* or *MaxPool*.
- Use soft or noisy labels. When training the discriminator, instead of using 0 and 1 for synthetic and real samples, use some random value between $[0, 0.1]$ and $[0.9, 1]$, and occasionally, flip the label values. This change neither will mislead the discriminator nor decrease discriminator's accuracy (it will only decrease the confidence of the discriminator).
- Use Adam optimizer. According to Radford et al. [**38**] the most popular and stable optimizer for GANs is Adam with a learning rate of 0.002 and the rest of the parameters in their default value.
- Different learning rates. Use a larger learning rate (step-length) for the discriminator than for the generator, for example a step-length of 0.0004 for the discriminator and 0.0001 for the generator.

- Do not balance depending on loss values. It is not recommended to try to invent some heuristics that train the discriminator or the generator depending on which of them has got the highest loss value to try to reach an equilibrium.
- If possible, use labels. Conditional GANs are demonstrated to be more stable (Section 2.4). They use some extra information that can be a class label, or for text data, an embedding. If embeddings are used, it is recommended on the one hand, that both the generator and the discriminator share the same embedding, and on the other hand, to keep dimensionality of the embeddings low.
- Track failure early. Usually if discriminator's loss decreases to zero, the norm of the gradient is too big (bigger than 100 for example) or the generator's loss decreases too fast, it probably means that the GAN is not stable or will not converge. Usually the discriminator's loss function has got lower variance when the GAN is training properly and it decreases with epochs.

2.4. Conditional Generative Adversarial Network (cGAN)

Conditional Generative Adversarial Networks (cGAN) are an extension of the GANs presented by Mehdi Mirza in 2014, [**47**]. The main difference is that cGANs can generate data from a previously specified category or class. In contrast with GANs, the user has control of the type of data that will be obtained from the generation process. Consequently, cGANs are, according to Mirza et al., the most influential GAN innovation for targeted data generation.

The generator in cGANs has got two inputs: random noise (z) and the extra information (y) that will condition the generated sample (see Figure 8). This condition can be a class label, a numerical value or an embedding vector. Therefore, the generator will try to generate samples from the corresponding data distribution of that class, that is, $G(z, y) = p(x|y)$ where the function p represents the generated probability distribution of the synthetic data.

The discriminator also receives two inputs: real (x, y) or generated data samples and their corresponding additional information that matches the sample $(p(x|y), y)$. The discriminator will learn to decide whether the given pair (sample and label) is a matching real sample or has been artificially generated by the generator. Hence, the discriminator returns a probability of how realistic is the given pair.

Since, in our implementation, the cGAN starts the generation from embedding values extracted from phrases. In the next section, we introduce several concepts related to embeddings.

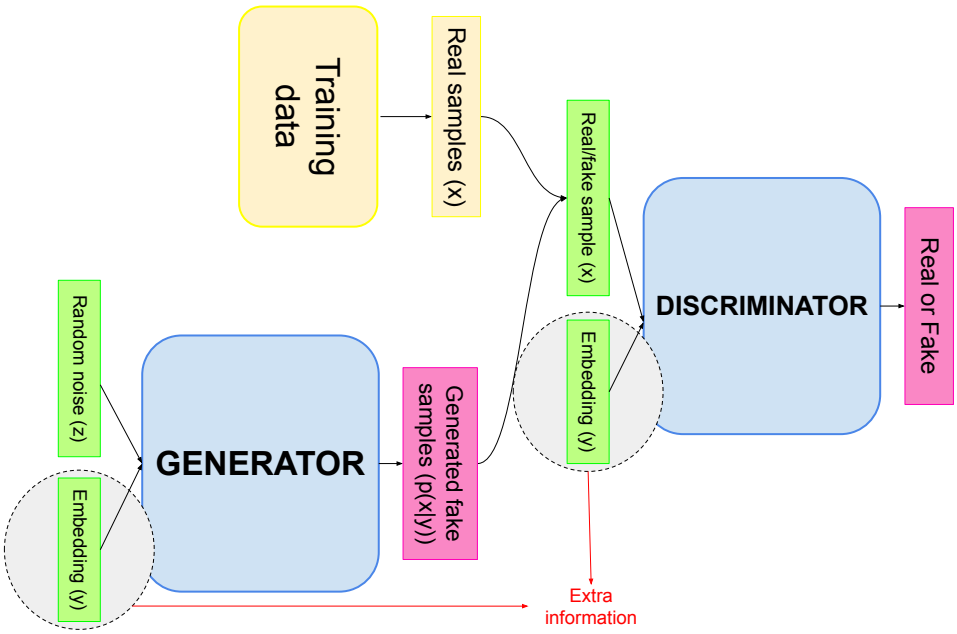

FIG. 8. Conditional Generative Adversarial Network

2.5. Embeddings

Embeddings are numerical representations in relatively low-dimensional spaces that are used to translate high-dimensional vectors (or text data). They are widely used in Machine Learning models because they make training easier and more efficient.

It is known that ANNs do not work with text, they only work with numerical values and vectors. Therefore, when working with text, strings or characters, they have to be converted into numbers in order to be analyzed by means of ANNs or other deep learning models. It is important to outline that these type of models do not understand text as humans, they only consider the statistical structure of the given data.

The technique of converting text into numerical representation is known as text vectorization. This can be done by dividing text into words, characters or n-grams (sequences of text with a fixed length of characters, n). These short sequences of text are called tokens. Hence, after dividing text into tokens, those are converted into numerical vectors known as embeddings. There are several techniques to transform words into embeddings such as one-hot-encoding, word embedding and character embedding. They mainly differ in the fact that word and character embeddings usually take semantic or spelling information into account whereas one-hot encoding does not.

- One-hot encoding: They are sparse high-dimensional vectors that are hard-coded, in other words, they are assigned by the software and are not obtained by learning from data.
- Word embeddings: This embeddings take semantic information from text and convert it into numeric vectors. Therefore, if two words have a similar meaning, they will be converted to two numerical vectors that are near from each other with a fixed distance. Word embeddings are dense lower dimensional vectors that are obtained by learning from data. They can be trained for each specific model by adding an embedding layer to the ANN. There are previously trained and successful word embeddings that are widely used, the most common are Word2Vec by Thomas Mikolov at Google in 2013 [**48**] [**49**] and GloVe by Standford researchers in 2014 [**50**] .
- Character embedding: They are very similar to word embeddings but unlike word embeddings, character embeddings take word spelling into account. Therefore, two words with similar spelling will differ less in distance than words with different spelling no matter the meaning of the words. In 2016, Cao et al. presented a pre-trained character embedding model called *Char2Vec* [**51**]

In our study, we use character embeddings (*Char2Vec*) for converting the information in the phrases typed by each user, to information that can be used and processed by our cGAN. This way, the adversarial network learns the keystroke dynamics of the users.

2.6. Keystroke dynamics

Keystroke biometric typing dynamics refer to the speed, rhythm, common errors and typing pattern information of a specific user. It is a biometric measurement that analyzes the behaviour of a user when typing. In computer science, researchers have found that thanks to the uniqueness of the typing features of each user, keystroke dynamics can be used to verify the identity of the person behind the machine. For example, it is thought that a user will type its name faster than a different name due to the habit of typing it many more times. Despite being unique, keystroke dynamics are inimitable. It is very difficult to type using other users rhythms, speeds and typing patterns to fool authentication systems for keystroke biometrics. However, it is necessary to take into account that the behaviour of a user strongly depends on the ergonomics, mood and tiredness of the moment [**52**].

Keystroke dynamics market is growing very fast in the last years due to the rise of online fraud and it is expected to continue growing in the following years. It seems to be a very effective method to authenticate users while typing in their machines without them being aware that the machine is continuously verifying their identity [**53**]. The main advantage of keystroke dynamics is that a standard computer with a keyboard is enough to collect data and no special hardware is needed to capture these typing times [**54**].

The typing times can be measured in some previously fixed-text or in free-text, that is, the text the user is typing can be unknown and random or previously fixed. However, the majority of the literature in the research area has been done with fixed-text data and free-text dynamics has not been studied at large scale yet, only with hundreds of user datasets.

2.6.1. TypeNet. As regards the latter quote, in a study called TypeNet, a large scale free-text database from Aalto University was used for keystroke behaviour analysis [**6**]. This database is the largest free-text keystroke database captured with more than 136M keystrokes from 168K users. In particular, in order to find out the typing patterns of a user, at each press, two times have to be registered apart from the pressed keycode: the press time and the release time. However Acient et al. used four variables to describe each typing pattern. These are: Hold Latency (HL), Press Latency (PL), Release Latency (RL) and Inter-Key Latency (IL), see Figure 9.

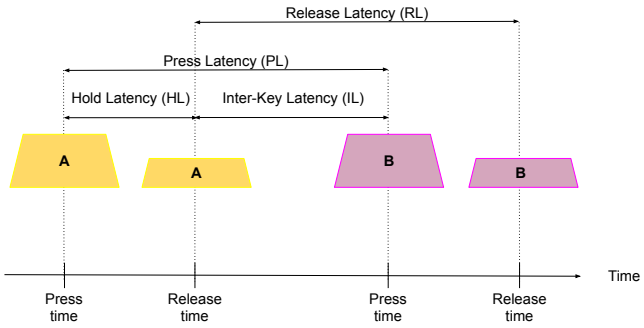

FIG. 9. Keystroke dynamics times representation.

As previously mentioned, Ancien et al. presented the model that has been used to validate the results in [**6**]. This model has been newly trained and improved for this study. The model corresponds to a Siamese Network. Siamese Networks are a type of ANN that are characterized by containing at least two subnetworks in it that have exactly the same architecture. In this case, each subnetwork consists of a RNN.

The input of each subnetwork corresponds to a keystroke sequence of length 15. For each keystroke, 5 features were taken into account: HL (hold latency), PL (press latency), RL (release latency), IL (inter-key latency) and $KEYCODE$. Therefore, this model receives two inputs of size 15×5 and after several layers in the architecture of each subnetwork, a distance is returned as output. The threshold of the distance for classifying the two given sequences is 0.5. In other words, if distance is larger than the threshold, it is assumed that the sequences correspond to two different users and if lower, the sequences are considered to belong to the same user.

The architecture of the model has two subnetworks that are identical. It is shown below in Figure 10. It can be observed that each subnetwork uses a LSTM layer to treat keystroke sequences sequentially. Once batch normalization and dropout are computed to avoid overfitting, it has got another LSTM layer to obtain two embedding vectors of size 1×128. Note that both LSTM layers have got an additional dropout of 0.2. Finally, in the subtract and euclidean distance layers, the distance from one embedding to the other is obtained.

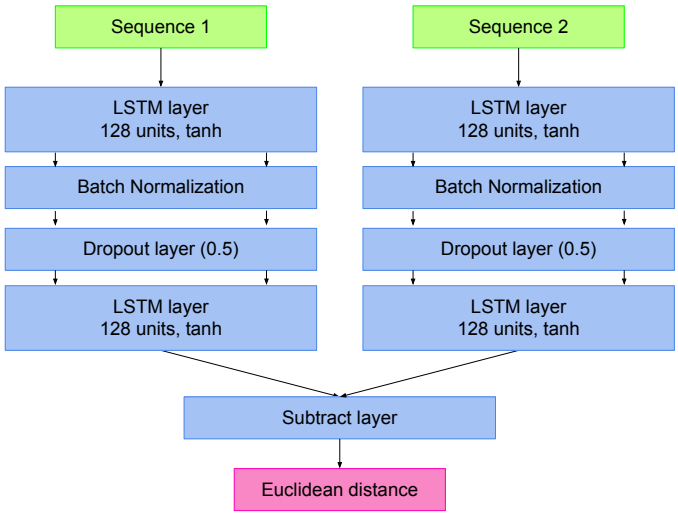

FIG. 10. TypeNet architecture.

In this study to validate the results, the generated sequences have been given to this validation model as input sequences. The aim was to detect if this model was capable to distinguish between generated and real data.

This model has been trained with 25 different users including Alice. It has been observed that this model does not behave accurately enough to validate users that were not in the training data and consequently, only users that were previously known by this model have been used in the study.

Chapter 3
Experimental Design

3.1. Context

To position this project, let us assume it is wanted to access a service which is protected by an identity provider called, Keyrock. To access this service, the user's identity is doubly verified. On the one hand, the Keyrock verifies the correctness of the password and on the other hand, it validates the way in which such password has been typed (keystroke pattern).

Alice, a registered user in the Keyrock protection system, wants to access the service with her credentials. Hence, Alice types the password and the identity provider verifies the password and keystroke pattern and enables access. See Step A in Figure 11 and Figure 12.

However, Bob, a non registered user, wants to access the service by impersonating Alice by means of a presentation attack. To that end, let us assume Bob already knows Alice's password. However, due to the double verification of the keyrock, Bob needs to learn Alice's typing behaviour spoofing the communication. In this context, the project is divided in two use cases.

- Use case 1: In this use case, it is assumed that Bob captures data directly, without noise. That is, he captures the typing patterns and knows exactly the order of all the words Alice has typed. See step B on Figure 11.
- Use case 2: In this use case, it is assumed that Bob captures data that contains noise. That is, he captures the typing patterns but does not know the order of the captured keystrokes. See step B on Figure 12.

In this framework, the aim of this project is to help Bob to develop the presentation attack. See step C in Figure 11 and Figure 12. In particular, the idea is to implement a generative model able to learn Alice's behaviour and then generate realistic synthetic data to fool the identity provider. In particular, in our study, we assume that the TypeNet model introduced in [**6**] (see section 2.6.1) is used by the Keyrock for verifying the user behaviour and allow the user to access to a certain

service. So, in this sense, the idea is that the implemented cGANs can learn Alice's behaviour for cheating directly the TypeNet model.

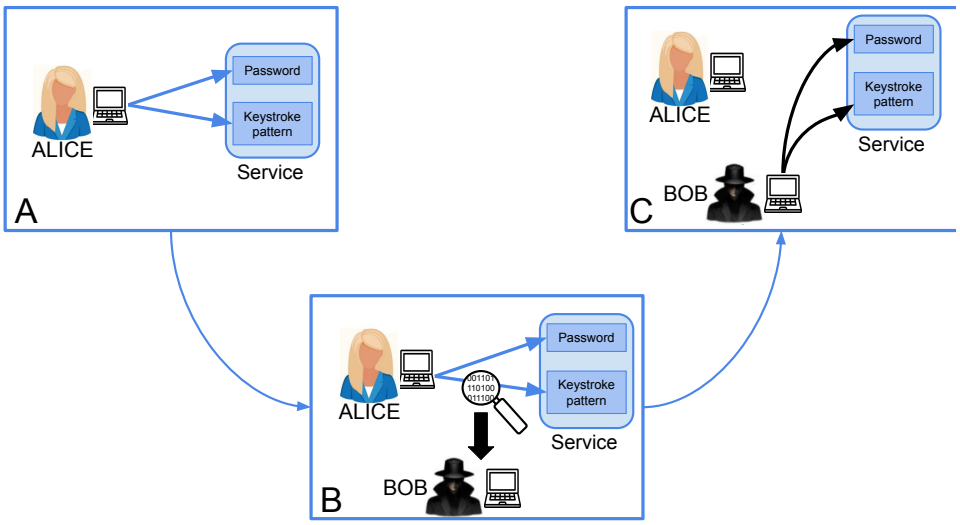

FIG. 11. Use case 1.

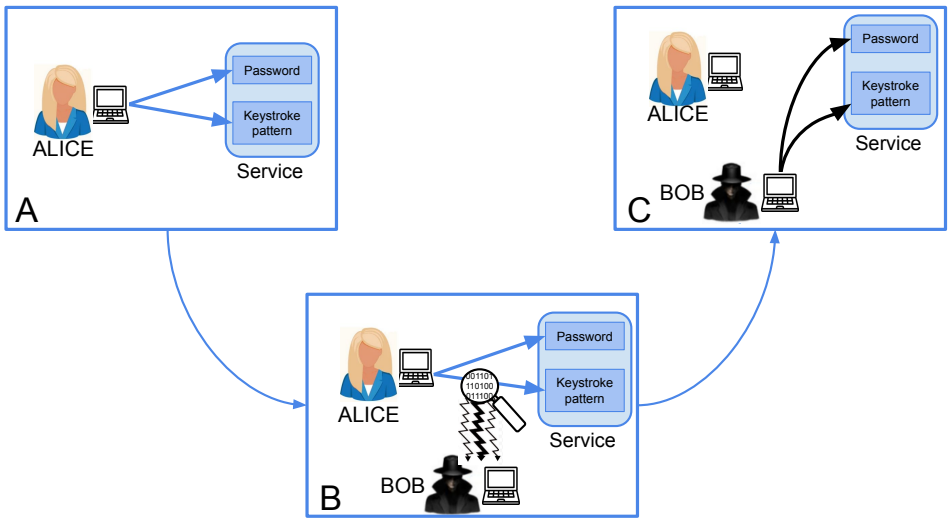

FIG. 12. Use case 2.

3.2. Data

This study has been carried out with the dataset provided by the Aalto University in Finland [**55**]. They collected data from 168K volunteers in an online study. It is a large scale dataset that contains information of the keystroke patterns of each of the volunteers. Each volunteer typed 15 sentences in English that were shown in their screen and had to memorize and type as quickly and accurately as possible. The displayed sentences were chosen randomly from a sample of $1,525$ sentences and contained more than 3 words and less than 70 characters, 5 special characters at most and only simple punctuation marks.

For all users, for each keystroke, each time the user pressed any keycode they recorded information about they key that was pressed, the time they pressed it and the time they released it. This times were registered via JavaScript's `date.now()` which its expected precision is $10 - 15$ milliseconds. Hence, they recorded more than 136M keystrokes.

The dataset contains the following 9 variables:

- *Participant_ID*: Identification number of each participant in the study.
- *Test_section_ID*: Identification number of each sentence.
- *Sentence*: String that was displayed in the screen for typing.
- *User_input*: String that the user actually typed.
- *Keystroke_ID*: Identification number of each keystroke.
- *Press_time*: Timestamps in which each keystroke has been pressed in milliseconds.
- *Release_time*: Timestamps in which each keystroke has been released in milliseconds.
- *Letter*: The letter that corresponds to each keystroke. They can be special characters or punctuation marks.
- *Keycode*: The keycode that corresponds to each keystroke according to the ASCII code.

Of course, not the entire dataset was used in our implementation, in fact, since we are only interested in learning and replicating Alice's typing behaviour, we have used only one user for training our cGANs. More precisely, the dataset used for training the models, has got 825 keystrokes that belong to 215 words grouped in 15 sentences. In order to choose this specific user, the performance of each user with the identity provider has been measured. For the final validation, other users have been used.

Below, a histogram of the keycodes registered by this user is shown, Figure 13. The histogram shows how frecuently was each keycode pressed in the data corresponding to user $Participant\_ID = 100182$. Looking at Figure 13, the most frequent keycode is the one that corresponds to the space in the keyboard (according to the ASCII code the space corresponds to the number 32). Note however that the keycodes 8, 69 and 84 are pressed more than 50 times (blue horizontal line). According to the

ASCII code, those keycodes correspond to the backspace and the letters E and T respectively.

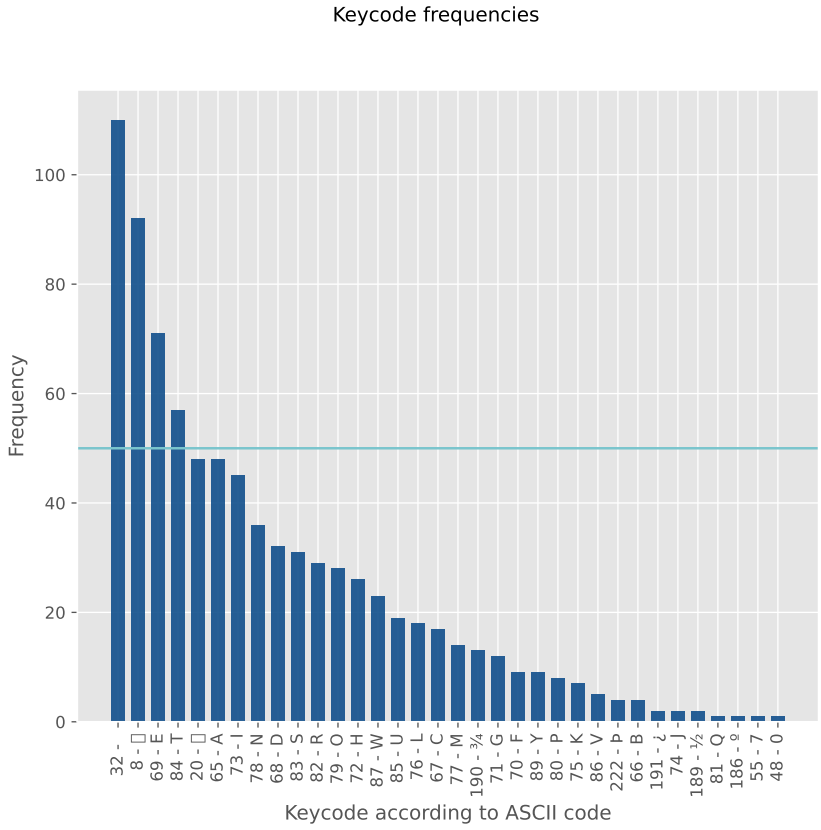

FIG. 13. Keycode frequencies of Alice (*Participant_ID* = 100182).

	HL	***PL***	***RL***	***IL***
Mean	92	270	269	179
Median	88	184	184	96
Standard deviation	29	285	282	283
Min	8	0	-8	-64
Max	224	4032	4032	3960
Range	216	4032	4040	4024

TABLE 1. Statistics of keystroke dynamics times of the user *Participant_ID* = 100182

In Table 1, the most important statistics of the collected times are shown for each keystroke. Looking at Table 1, taking into account the difference between the mean

and the median, we can conclude that the four variables are not biased. However, the variables *PL*, *RL* and *IL* have values that range in a wide interval because they have some extreme values. In contrast, the variable *HL* does not have extreme values and has a narrower range. Also, it can be observed that the variables *RL* and *IL* have negative values. This negative values are obtained when either the second keycode is released before releasing the first one (negative *RL*) or when the second keycode is pressed before releasing the first one (negative *IL*).

It is important to outline that although the original dataset is composed of sentences, sequences of 15 keystrokes have been used to validate and try to fool TypeNet. What is more, the training of the cGANs has been done by words, that is, the cGANs learn to generate the typing behaviour of words instead of sequences or sentences. Once the synthetic word typing behaviours have been obtained, sequences have been reconstructed in order to validate the results.

After preprocessing the original dataset, as mentioned before, a dataset composed of 215 words has been used to train the models. In Figure 14, the histogram of the frequencies of the lenghts of the words in the dataset is shown. By looking at Figure 14, it can be observed that the most abundant words are the words with only one keystroke, they are more than half of the words (53% of the words) in the dataset. Note that the space between two words has been considered a word. What is more, in English the words "*I*" and "*a*" are also one keystroke words. Also, according to the histogram, it can be concluded that the majority of the words length less than 7 keystrokes, only 9.8% of the words length more than 6 keystrokes.

3.3. The Models

To solve the problem presented in Section 3.1, different architectures of cGANs have been used. Two models have been built: a Conditional Vanilla-GAN (i.e. a cGAN composed of two ANN for G and D) and a Conditional LSTM-GAN (i.e. a cGAN composed of two RNNs with LSTM units for G and D). Both models have been designed, implemented, validated and improved. They both generate synthetic data in an adversarial way about the typing behaviour of a user, hence, the output of the three approaches are the typing times of the user. However, the main difference lies on the fact that unlike the Vanilla-cGAN, the LSTM-cGAN considers the temporal relation of the provided information (in this case the characters in a word, words in a phrase, and so on). It is important to outline that they have been trained for one specific user and therefore, they learn to generate data that corresponds to the typing behaviour of that user.

Both models (Vanilla-cGAN and LSTM-cGAN), for both subnetworks receive a 100 dimensional character embedding that corresponds to a word in the training dataset. These embeddings have been obtained from the pre-trained model *Char2Vec* which is already included in a `Python` package called `char2vec`. To that end, both subnetworks learn how should Alice's typing times and *KEYCODE* sequences be for that exact word. In the case of the discriminator, it learns whether

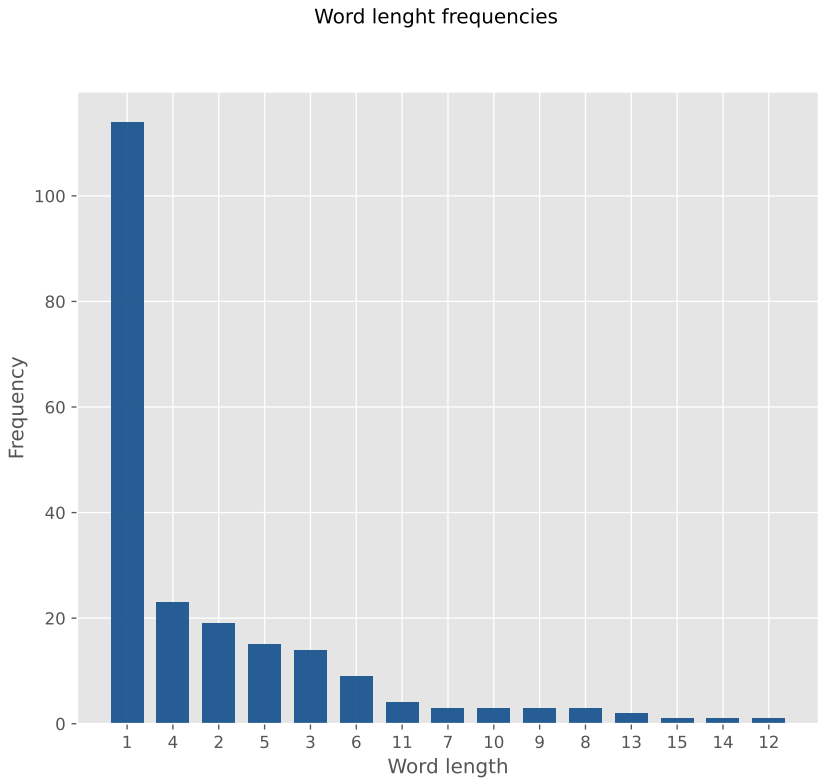

FIG. 14. Word length frequencies of Alice ($Participant\_ID = 100182$).

the introduced sample corresponds to that word and if this sequence is realistic enough to be real or it has been generated by G. However, G learns to generate synthetic samples for each embedding. In what the generator of both models concern, it is important to outline that in both cases the random noise vector, corresponds to a 500 dimensional vector that follows a $N(0, 0.1)$ distribution.

3.3.1. Conditional Vanilla-GAN. This cGAN is composed of two ANNs with two inputs and one single output (see Figure 15). On the one hand, the generator receives as input a 500 dimensional random vector and a 100 dimensional vector that corresponds to a character embedding from a word. These two vectors are concatenated and transformed three times with three dense hidden layers with 256, 512 and 256 neurons respectively. The three hidden layers use the $LeakyReLU$ activation function with $\alpha = 0.2$ (a modified function of the $ReLU$ activation function that has a small slope in the negative values). Last, the output layer is a dense layer with 75 neurons (it returns a 75 dimensional vector) and the $tanh$ activation function to range the elements in the output between -1 and 1.

On the other hand, the discriminator also receives two vectors as input to the model: a 75 dimensional array that corresponds to a real or synthetic sample and a 100 dimensional character embedding mentioned above. This input is transformed by means of 4 dense layers of which three are hidden layers and the last one is the output layer. The hidden layers are composed of 512, 256 and 128 neurons respectively and all use the *LeakyReLU* activation function with $\alpha = 0.2$. However, after the last hidden layer a dropout with a rate of 0.4 is applied before transferring the information to the output layer. Last, a one neuron dense layer is used with the *sigmoid* activation function to return a value that ranges between 0 and 1.

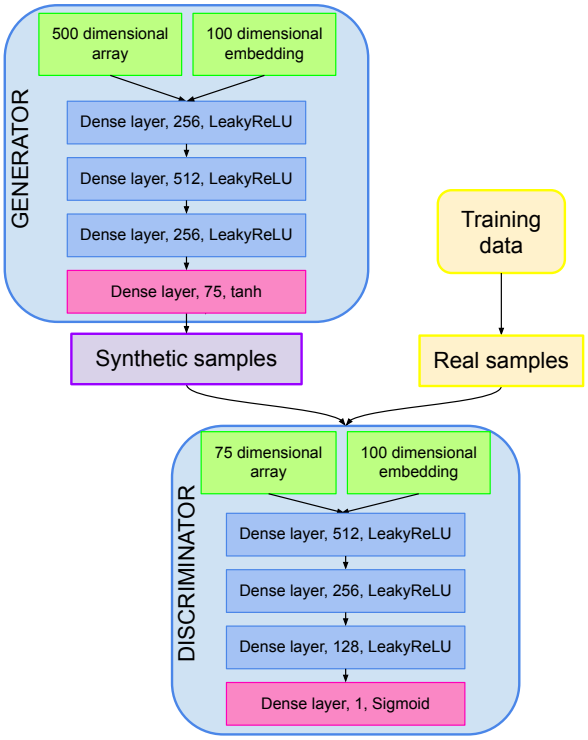

FIG. 15. Architecture of the Vanilla-cGAN.

3.3.2. Conditional LSTM-GAN. This model is composed of two RNNs with two inputs and one output each (see Figure 16). The generator receives a 500 dimensional random noise vector and a 100 dimensional vector that corresponds to a character embedding of a word as input. This information is first transformed by a dense layer with 400 neurons and *ReLU* activation function. Two LSTM layers receive the output of the dense layer and transform it by means of 400 units each using the *tanh* activation with a dropout with a rate of 0.4 between the layers. Last, output is obtained by another dense layer with 75 neurons and *linear* activation function to be reshaped to a 15×5 dimensional array.

In what the discriminator concerns, the architecture is very similar to the generator's architecture. It only has two LSTM layers separated by a dropout layer in between, followed by a dense output layer with one neuron with the *sigmoid* activation function.

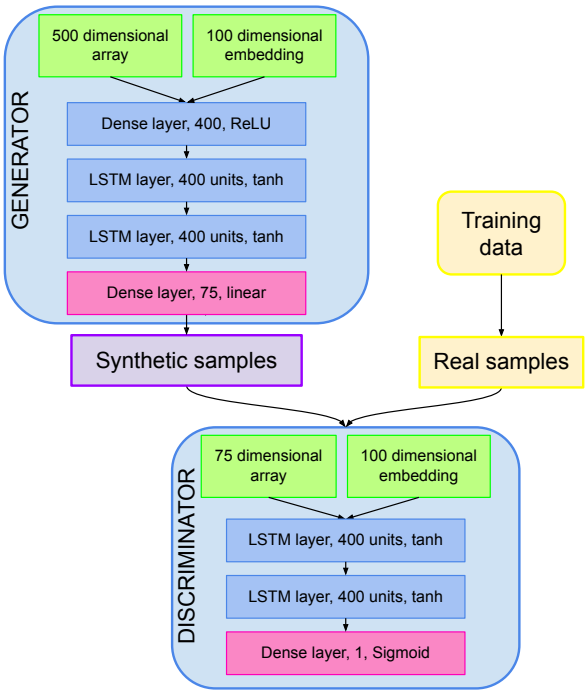

FIG. 16. Architecture of the Conditional LSTM-GAN.

3.3.3. Optimizers. In what training optimizer details concern, both models have been trained with the Adam optimizer and the `binary_crossentropy` loss function. In the case of the Vanilla-cGAN the learning rate of both the generator and the discriminator used is 0.0002 and the default values for the rest of the parameters in the optimizer (`beta_1=0.9` `beta_2=0.999` and `epsilon=1e-07`). In the case of the LSTM-cGAN the learning rate for the discriminator used is 0.0004 and 0.0001 for the generator, `beta_1` has been fixed to 0.5 and the rest of the parameters were used in their default value (`beta_2=0.999` and `epsilon=1e-07`). With respect to the batch size both models have been trained with the batch size fixed to 32.

3.4. Use cases

3.4.1. Use case 1. In this section, the details of the first use case are explained. As previously mentioned, this use case assumes Bob has captured data directly, without noise. That is, Bob knows exactly the order in which Alice has typed all the keystrokes, words and sentences. Therefore, by means of the implemented models, Bob can recreate the same keystroke sequences to impersonate Alice.

Note that both models have been trained by words, therefore, in order to validate the results, the sentences need to be recreated word by word taking into account the order in the original dataset. To reconstruct the sequences and validate them with the external model, TypeNet (see section 2.6.1), sequences of length 15 keystrokes are required. To that end, each word in the sequence has been generated separately and then concatenated in the corresponding order to recreate realistic samples considering the original order of the words.

For a better understanding of the process, let us consider the sentence "*we would have waited up all night*". In this example, it is assumed that the user has typed the sentence without making any mistake (without using the backspace key). To generate a sample that corresponds to the first 15 keystrokes ("*we would have w*"), as both experiments generate samples by words, the sequence needs to be divided into words. Then, by means of the corresponding 100 dimensional character embedding and a 500 dimensional random noise vector for each word, the synthetic samples are generated using the generators (G) in the cGANs. Last, the initial sequence is reconstructed supressing the padding for each generated data for all words. Hence, the times and *KEYCODE* sequences are obtained without padding and can be validated directly by the external model. See Figure 17.

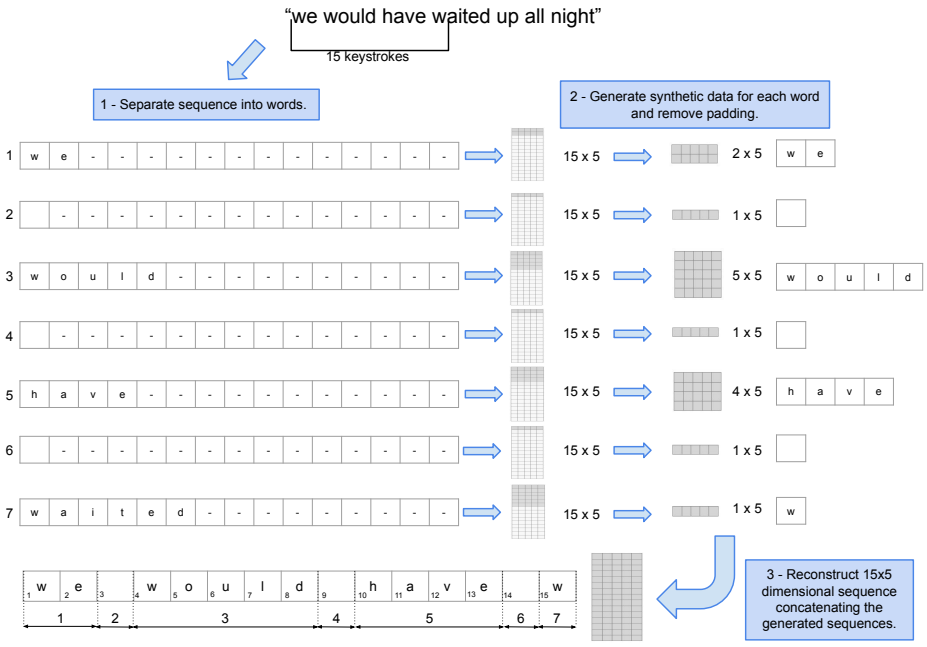

FIG. 17. Example of generation and reconstruction of a sequence in use case 1.

3.4.2. Use case 2. In what the second use case concerns, it is assumed that Bob captures data on Alice's keystroke dynamics affected by some noise that does not

allow Bob to know the order the words that have been typed. Therefore, after generating synthetic samples with the captured words, Bob is not able to recreate exactly the sequences written by Alice.

In this use case, after training the models in both models, in order to validate the results, the captured words are concatenated randomly to create sequences that length 15 keystrokes. Unlike in the use case 1, the order of the words in the sequences is not the order in which Alice has typed them. However, when recreating realistic sequences with 15 keystrokes, after every word a space has been added.

3.5. Metrics

A metric is a measurement number that gives information about the performance of a model. Depending on the output of the model there are different metrics that can be used to measure the performance. In this study, because of the binary output of the GAN model, that is, the output of the discriminator $\hat{p}_i$ (see Section 3.3) that ranges between 0 and 1, some metrics for binary classification have been used. In addition, in what external model of the validation model concerns, as the output ranges between 0 and ∞, a threshold has been fixed (0.5) to decide whether the distance was small to consider that the sequences belong to the same user (0) or big enough to consider that they belonged to two different users (1).

When the output of the model $\hat{p}_i$ which performance is to be measured is binary, a threshold is defined to divide the outputs in two groups: estimated real samples and estimated fake samples. Let $c \in (0, 1)$ be the threshold and n the number of estimated values. Therefore, the criteria to divide the outputs in two groups is the following:

$$\hat{Y}_i = \begin{cases} 0 & \hat{p}_i \geq c \\ 1 & \hat{p}_i < c \end{cases} \forall i = 1, \ldots, n \tag{3}$$

Let us build the confusion matrix of the observed and estimated values of the response variable Y which depend on the fixed threshold.

		Estimated		
		0	**1**	**Total**
Observed	**0**	TN(c)	FP(c)	TN(c)+FP(c)
	1	FN(c)	TP(c)	FN(c)+TP(c)
	Total	TN(c)+FN(c)	FP(c)+TP(c)	n

TABLE 2. Confusion matrix of the estimated and observed values depending on the threshold c.

Once the confusion matrix has been built, some common metrics for binary classification can be computed, such as, accuracy, recall, precision and F1 Score.

- Accuracy: the most common metric. It explains what percentage of the predictions are correctly classified. Accuracy ranges between 0 and 1, the closer to one the better.

$$Accuracy(c) = \frac{TP(c) + TN(c)}{TP(c) + TN(c) + FP(c) + FN(c)}$$

- Recall or True Positive Rate: It explains how accurately were the ones classified. It ranges between 0 and 1 and the closer to one the better.

$$Recall(c) = \frac{TP(c)}{FN(c) + TP(c)}$$

- Precision or Positive Predictive Value: It explains the how many of the predicted positives were in fact true positives. This also ranges between 0 and 1 and the closer to 1, the better.

$$Precision(c) = \frac{TP(c)}{TP(c) + FP(c)}$$

- F1 Score: It corresponds to the harmonic mean between the precision and recall. This metric is recommended when the positives and negatives are unbalanced.

$$F1\ Score(c) = 2\frac{Precision(c) \cdot Recall(c)}{Precision(c) + Recall(c)} = \frac{2TP(c)}{2TP(c) + FN(c) + FP(c)}$$

3.6. Validation

As previously mentioned, unlike in discriminative modelling, a drawback of generative modelling is that it can be difficult to validate the synthetic results generated by those models. In this work, two types of validation approaches have been applied to measure the goodness-of-fit of the results.

3.6.1. First validation. The first validation approach uses the discriminator in the cGANs. As this subnetwork decides whether a given sample is a real sample or has been generated by the generator, during the training procedure of the cGAN, this model can be used to evaluate if the generated samples are realistic enough. To that end, every 50 epochs, training has been stopped and 5 subsets of 32 real and 32 synthetic samples each were given to the discriminator as input. Hence, the accuracy of the discriminator was measured. If in mean (with the 5 subsets), more than 85% of the real samples were classified as real and more than 85% of the generated samples were classified as real, training was stopped. Note that this condition can be satisfied even when the generator is generating garbage samples and the discriminator has not been trained enough.

Next, the steps taken to generate the samples for each of the 5 subsets are explained in detail.

(1) **Generate** as many **random noise** vectors as the batch size (in our case 32).

(2) **Obtain a subset** of the training dataframe with as many rows as the batch size (in our case 32).
(3) With the random noise vectors and the character embeddings of the words in the subset **generate fake samples** (*HL*, *PL*, *RL*, *IL* and *KEYCODE*) using the trained generator.
(4) **Evaluate the generated samples** with the trained discriminator and save predictions.
(5) **Evaluate the real samples** in the subset (obtained in step 2) with the trained discriminator and save predictions.
(6) Count the number of times the prediction of the discriminator has been above 0.5 for both generated and real samples and **compute the accuracy of the discriminator to classify the samples as real**. Save accuracies.

3.6.2. Second validation. The second validation approach has been done by means of the model presented at section 2.6.1, TypeNet. As mentioned in such section, this model receives two keystroke sequences of length 15 with the corresponding times (*HL*, *PL*, *RL* and *IL*) and *KEYCODE* values. Hence, it recevies two 15×5 sized inputs. This model returns an euclidean distance of two embeddings (one for each sequence) that ranges between 0 and ∞. If the distance is higher than 0.5, the threshold, then it will be considered that the two input sequences correspond to different users. In contrast, if the output distance of the input sequences is lower or equal to 0.5, then, it will be considered that both sequences correspond to the same user.

TypeNet has been used to validate the generated results with three different tests, see Figure 18.

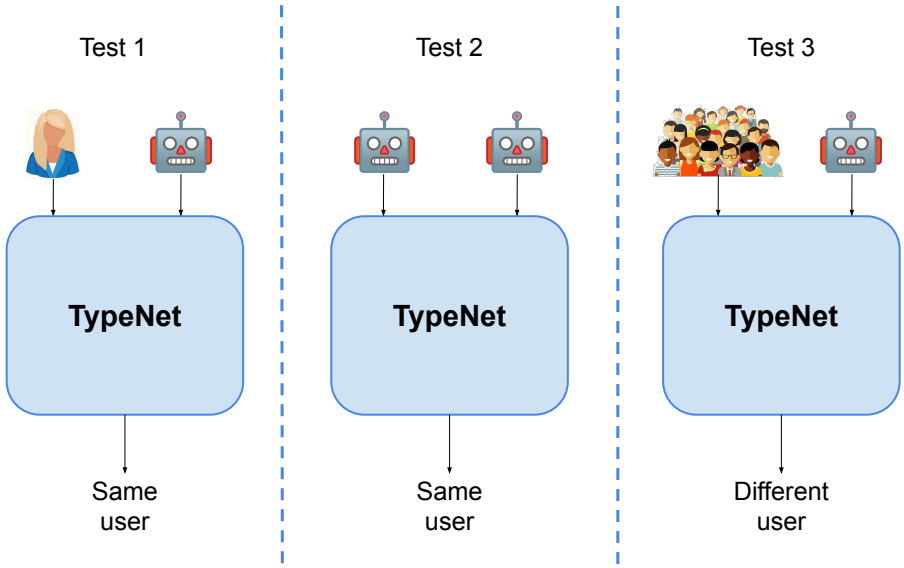

FIG. 18. Ilustration for the three tests in the second validation approach.

(1) **Test 1: Real vs. Fake** (*Acc_rf*). 20 real sequences and 20 generated sequences from the same user are validated to try to fool the external model. The correct classification for computing accuracy in this test are considered to be the predictions that are below 0.5, that is, they belong to the same user. This test is repeated with 10 different subsets, that is, with 200 different real vs. generated sequence pairs. The aim of this test is to measure how similar are the real sequences and the synthetic sequences.
(2) **Test 2: Fake vs. Fake** (*Acc_ff*). 20 generated sequences used in test 1 and another 20 generated sequences from the same user are validated with the external model, TypeNet. The correct classification for computing the accuracy in this second test are considered to be the predictions that are below 0.5, that is, they belong to the same user. This test is repeated with 10 different subsets, that is, with 200 different generated vs. generated sequence pairs. The aim of this test is to measure how similar are the synthetic sequences between them.
(3) **Test 3: Real other vs. Fake** (*Acc_rof*). 20 generated sequences used in test 1 and 20 real sequences that belong to other users (the other 24 users for which TypeNet has been trained) are validated with the external model. Hence, in this case, the sequence pairs that are validated belong to different users. In this test, the correct classification for computing accuracy are considered to be the predictions that are above 0.5, that is, they belong to different users. This test is repeated with 10 different subsets, that is, with 200 different real vs. generated sequence pairs. The aim of this test is to measure how different are the real sequences from other users to the synthetic sequences from our user.

3.7. System architecture

To develop this project `Python 3.7.6` has been used as programming language. `Tensorflow 2.6` is used for implementing the Machine Learning algorithms, together with `Keras 2.6` and `Skikit-learn 0.22.1`. All the experiments - data preprocessing, training, validations and testing of the models - were executed in a server (Virtual Machine) with 32GB RAM memory, Intel Xeon Silver 4114 CPU 2.20GHz, and 428GB of disk size.

Chapter 4
Analysis

In this chapter, first, the data preprocessing is explained. Then, the obtained results for each use case and model are presented. Hence, for each use case, both models that have been mentioned in section 3.3 have been used and validated.

4.1. Data preprocessing

In order to prepare the data to train the both models, the original dataset presented in section 3.2 has been preprocessed. To that end, data has been filtered, divided into words and normalized.

First, data has been filtered by user and the user with the $PARTICIPANT\_ID = 100182$ has been chosen to perform the analysis as if it was Alice. Once the sentences that Alice typed were selected, they have been divided into words. For that purpose, each time Alice typed the space (32 according to the ASCII code), a new word was recorded. It is important to outline that the space itself has been considered a word because the typing times of this keystroke also contribute to the typing pattern of Alice.

The data samples that the generative models will try to imitate, contain information about four typing times (HL, PL, RL and IL) and the corresponding $KEYCODE$ for those times. In order to prepare the values to train the cGANs, training data has been normalized. To normalize it, on the one hand, the four typing times have been converted from milliseconds to seconds (they have been multiplied by 0.001). On the other hand, as the variable $KEYCODE$ ranges between 0 and 255 according to the ASCII code, it has been divided by 255 to obtain values that range between 0 and 1. Last, because of the fact that in ANNs, the input dimension has to be fixed, a padding with value -1 for all variables has been added to the words at the end. Hence, all the word sequences have a maximum length of 15 keystrokes. Those words that length more than 15 keystrokes have been supressed from the training data. Therefore, for each word, the models receive 5 normalized and padded sequences of length 15.

In order to validate the synthetic data generated by the cGANs, first, it is needed to denormalize. That is, the typing times (*HL*, *PL*, *RL* and *IL*) need to be converted back into milliseconds multiplying them by 1000 (as in the original dataframe) and the *KEYCODE* values need to be multiplied by 255 (the maximum value for a *KEYCODE* value according to the ASCII code). However, as TypeNet receives typing times in milliseconds and *KEYCODE* values normalized (divided by 255), in order to use the generated samples from the cGANs, in the end, only the typing times need to be converted to milliseconds and *KEYCODE* values stay the same. Last, it is needed to reconstruct the 15×5 sequences concatenating words and spaces without any padding as the input of TypeNet requires. This last process is different in each use case, because unlike in use case 1, in use case 2, the order of the typed words is not known (see Section 3.4).

4.2. Results

The aim of this project was to access a protected service by impersonating a registered user. To that aim, two models have been designed and implemented. In order to validate the models, two validation approaches have been used, one during the training process and the other after the training process was stopped. However, it is important to outline that during training the trained models have only been saved if the first validation condition (the accuracy of the discriminator classifying for both real and synthetic samples as real samples was higher than 0.85) was satisfied.

The Vanilla-cGAN, has satisfied the first validation condition after being trained for $163,000$ epochs. In contrast, in what the second model concerns, the LSTM-cGAN, it has satisfied the first validation condition after being trained for 3600 epochs. As the difference in epoch number might have had an impact in the results of the second validation procedure, we have continued training the LSTM-cGAN model until the first validation condition was satisfied after more than $10,000$ epochs. In this case, the first validation condition was satisfied at epoch $10,150$. Subsequently, this model will be referenced as LSTM-cGAN 2.

As mentioned previously (see section 3.6), in order to measure the performance of each model three tests have been made with 10 different repetitions each. Hence, in Figure 19, the results obtained for each repetition with the three models (Vanilla-cGAN, LSTM-cGAN and LSTM-cGAN 2) are shown.

In Figure 19, it can be observed that the Vanilla-cGAN and the LSTM-cGAN 2 are the models with the highest accuracy in which all the accuracy values for all repetitions in both cases are above 0.8. Nevertheless, it can be observed that the LSTM-cGAN does not behave that well because the accuracy values for the 10 repetitions range between 0.3 and 0.55. In addition, the values of the LSTM-cGAN present more variability, that is, bigger differences between repetitions than in the case of the other two models.

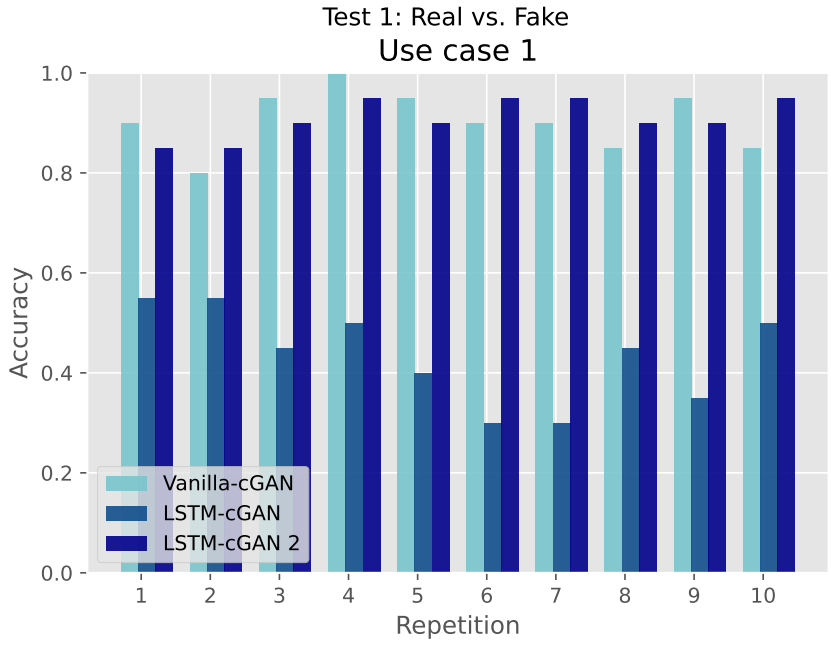

FIG. 19. Side-by-side barplot with accuracies of the Vanilla-GAN, the LSTM-cGAN and the LSTM-cGAN 2 for each repetition in use case 1.

In order to compare the results from use case 1, with the results in use case 2, the side-by-side barplot in Figure 20 shows the accuracy of the predictions of each of the 10 repetitions in use case 2.

By looking at Figure 20, we can observe a similar behaviour to the results in use case 1, but in general, with slightly lower values. It can be observed, that once again, the Vanilla-cGAN and the LSTM-cGAN 2 have a high accuracy, they range between 0.8 and 1. Note that in terms of accuracy, despite the case of the repetitions 1, 4 and 7, the LSTM-cGAN 2 has got a better performance than the Vanilla-cGAN. By looking at the results of the LSTM-cGAN, it can be observed that the accuracy for the repetitions also ranges between 0.3 and 0.55 but the results are lower.

Table 3 and Table 4, show the mean accuracy values for each test.

Looking at Table 3, the one that shows the results from use case 1, it can be observed that comparing both models that satisfied the first validation condition for the first time, the Vanilla-cGAN has got a much higher accuracy comparing to the LSTM-cGAN. In the case comparison between the Vanilla-cGAN and the LSTM-cGAN 2, it can be observed that the TypeNet on a very high percentage ($90\% - 91\%$ for the Vanilla-cGAN and $91\% - 93\%$ for the LSTM-cGAN 2) considers the samples as if they belonged to the same user. However, by looking at the results

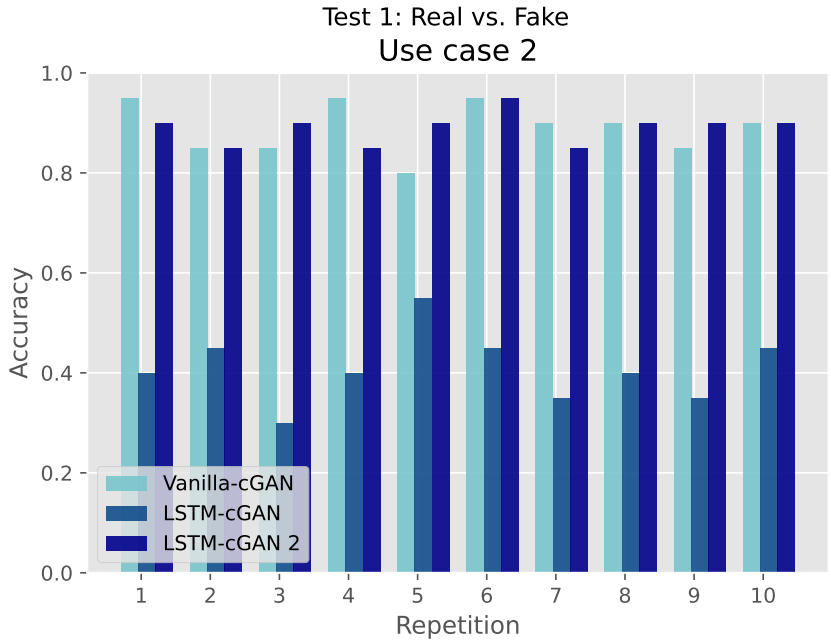

FIG. 20. Side-by-side barplot with accuracies of the Vanilla-GAN, the LSTM-cGAN and the LSTM-cGAN 2 for each repetition in use case 2.

	Mean accuracy		
Use case 1	**Test 1: Real vs. Fake**	**Test 2: Fake vs. Fake**	**Test 3: Real other vs. Fake**
Vanilla-cGAN	0.90	0.91	0.94
LSTM-cGAN	0.44	0.45	0.94
LSTM-cGAN 2	0.91	0.93	0.96

TABLE 3. Accuracy of the classification of TypeNet for the generated data in use case 1.

	Mean accuracy		
Use case 2	**Test 1: Real vs. Fake**	**Test 2: Fake vs. Fake**	**Test 3: Real other vs. Fake**
Vanilla-cGAN	0.89	0.87	0.95
LSTM-cGAN	0.41	0.5	0.97
LSTM-cGAN 2	0.89	0.88	0.97

TABLE 4. Accuracy of the classification of TypeNet for the generated data in use case 2.

of the LSTM-cGAN, it is evident that they are worse than the results of the other two models. Probably, this is because although according to the discriminator

in the LSTM-cGAN the generator has learned the typing behaviour of our user, the generator has not learned to generate samples that resemble real data. When looking at the results of test 3, all three models have got very good results in terms of accuracy. Thus, all three models generate synthetic data that do not resemble any other user in the training data used to train TypeNet (any of the rest 24 users). Note that in the case of the LSTM-cGAN, even if not having a high accuracy tests 1 and 2, according to the results in test 3, the generated samples are not similar to other users real samples.

If we compare the results obtained for the Vanilla-cGAN and the LSTM-cGAN 2, we can observe that in spite of being both models quite good at impersonating Alice, the model LSTM-cGAN 2 behaves better than the Vanilla-cGAN because in all three tests accuracy is higher.

In what the results in use case 2 concern, Table 4 shows the results of all three tests from the second validation approach. Alike in use case 1, in the three models, the accuracy values are similar in test 1 and test 2. Once again, it is evident that the LSTM-cGAN is the worst model because it has obtained an accuracy of 0.41 at test 1 and 0.5 at test 2. In contrast, the LSTM-cGAN 2 has got the best results in both tests with accuracies 0.89 and 0.88 at test 1 and test 2 respectively. Alike in use case 1, in use case 2, test 3 has got very high accuracy values for all three models. Once again, it can be observed that the generated samples, in spite of not having similar results in test 1 and test 2, all three models generate synthetic data that does not resemble to the rest 24 users registered in TypeNet. Therefore, there are no significant differences between models when comparing the synthetic samples with real samples from other users.

Next, the results for other metrics (see section 3.5) for all three tests together are presented for both use cases.

Use case 1	**Recall**	**Precision**	**F1 Score**
Vanilla-cGAN	0.94	0.83	0.88
LSTM-cGAN	0.94	0.46	0.61
LSTM-cGAN 2	0.97	0.86	0.91

TABLE 5. Recall, precision and F1 score values for the three models in use case 1.

Use case 2	**Recall**	**Precision**	**F1 Score**
Vanilla-cGAN	0.95	0.83	0.87
LSTM-cGAN	0.97	0.47	0.63
LSTM-cGAN 2	0.97	0.80	0.88

TABLE 6. Recall, precision and F1 score values for the three models in use case 2.

By looking at Table 5 and Table 6, it can be observed that the values for recall are high in both use cases, this means that the positives, in our case, the different user classification cases, have been sucessfully classified. Thus, once again, it is confirmed that all three models generate synthetic data that does not resemble to any other known user of TypeNet. In what precision concerns, the values are lower in the case of the LSTM-cGAN. This might be due to the abundance of false positives, that is, two sequences that belonged to the same user, have been classified as different, and consequently, lower values of precision have been obtained in these cases (use case 1 and use case 2). In the case of the Vanilla-cGAN and the LSTM-cGAN 2, the values for precision have been higher on account of a lower amount of false positives and high accuracy values. In what the F1 score concerns, as it is a harmonic mean between the precision recall, if the value is low, then the classifier has been optimized to increase one and discredit the other. In our case, the F1 score is high for the Vanilla-cGAN and the LSTM-cGAN 2. However, as the precision of the LSTM-cGAN is low in both use cases, then the F1 score decreases in these cases.

With regards to the difference between use cases, according to the results of tests 1 and 2, slightly better results have been obtained when more prior information was known about the order of the typed words. However, there are no significant differences between use cases when the sequences that are being compared belong to two different users.

According to the tables and barplots above, it can be concluded that the Vanilla-cGAN performs better than the LSTM-cGAN. However, by looking at the results of the LSTM-cGAN 2, it has been observed that this difference can be reduced by training the model for more epochs ignoring the first validation condition. It is important to outline that the model LSTM-cGAN 2 has been trained for more epochs ($10,150$ epochs) in spite of having satisfied the first validation condition at epoch 3600. What is more, when training the LSTM-cGAN for more epochs, (LSTM-cGAN 2), better results have been obtained comparing to the results of the Vanilla-cGAN trained for $163,000$ epochs.

4.2.1. Summary. After analyzing the performance of the synthetic data obtained from the implemented models in both real use cases, it can be concluded that:

- In what the effect of the prior information on the order of the words that have been typed concerns, that is, in what the difference between the two analyzed use cases concerns, in spite of having better results when the order of the typed words was known (more prior information was known), there are no significant differences. Hence, it can be concluded that the relevance of the order of the typed words does not affect the typing behaviour of the user.
- Both architectures Vanilla-cGAN and LSTM-cGAN are valid architectures to generate realistic synthetic data.
- Treating data on keystroke dynamics as sequential data, that is, using LSTM units and RNNs is more efficient because with less training than with a "traditional" ANN, better results can be obtained.

- The internal discriminator's accuracy from the cGANs cannot always be reliable in order to decide when to stop training and external models are of great advantage when measuring how realistic are the generated samples.

4.3. Discussion and future work

The key findings in this study on synthetic data generation on keystroke dynamics present promising results with high performance at impersonating a user. The two designed Conditional Generative Adversarial Networks have shown that these architectures are able to learn the typing behaviour of a determined user and impersonate it. The Vanilla-cGAN performed better than the LSTM-cGAN in spite of being trained for many more epochs to fulfill the same stopping criteria. However, when this stopping criteria was removed for the LSTM-cGAN and training continued longer, better results were obtained with still less training epochs than the Vanilla-cGAN. Therefore, it can be concluded that the RNN subnetworks with LSTM units in the architecture of the cGANs have played an important role when generating synthetic data on keystroke dynamics. Consequently, it has been proved that treating keystroke dynamics as sequential data can lead to better results.

In line with the hypothesis of the importance of biometrics in digital security user authentication systems, this experiment provides a new insight into the data generation on keystroke dynamics in an adversarial way. Not only it has been demonstrated that it is possible to generate this type of data in an adversarial way, but also, it is valid data to access some secure authentication system by impersonating a user with the generated data.

The methodological choices were constrained by the amount of data on the keystroke dynamics of each of the users used in this study. Due to the lack of data on the number of words typed by each user, the results cannot confirm generalizability of the typing behaviour of the user in its integrity when it comes to typing other words that are not in the training data. In addition, the reliability of this results is impacted by the goodness-of-fit of the external model TypeNet used to validate the generated samples. Thus, future studies should take into account that more data on each user can make a big difference in data generation.

Further research is needed to determine a proved condition/recommendation in which the training of the Generative Adversarial Networks should stop because either the whole adversarial game will not converge or the generator has already learned the distribution of the given training data.

Future research should consider the potential of keystroke dynamics data generation in an adversarial way. This study is the key component in future attempts to improve adversarial data generation on the area. Not only for the state of the art, but also for the Protection&Trust line at Vicomtech, it would be interesting to examine a cGAN composed of two Convolutional Neural Networks, and work is already being done in this direction with a future scientific article coming soon.

Chapter 5
Conclusion

It is widely known that user authentication systems are coming to power on this era of technology growth, and to that aim, biometrics are to date, the most reliable user authentication system. However, presentation attacks have become of great consequence of this development. In line with this hypothesis, in this work a keystroke presentation attack has been implemented by means of Conditional Generative Adversarial Networks.

The main goal of this work has been to generate synthetic data on keystroke dynamics in an adversarial way. Throughout this work, two different architectures of cGANs have been designed, implemented, validated and improved. In order to validate the results, two validation approaches have been used. On the one hand, a stopping criteria has been fixed using the internal discriminator of the designed architectures to decide throughout training when the models were trained enough. On the other hand, an external model, TypeNet, has been trained to detect whether the synthetic samples could be distinguished from real samples or not. This analysis has been done in two different use cases considered to implement and validate the models in two real scenarios that differed in the captured information about the order of the typed words.

Results indicate that keystroke dynamics patterns can be adversarially generated and promising results can be obtained in the two different real scenarios. Evidence suggests that using LSTM units, and thereupon, RNNs can lead to better results with less training than "traditional" ANNs. In contrast, on account of the accuracy of the internal discriminator of the models, findings show that at the beginning of the training, the generator in the LSTM-cGAN can generate samples that successfully fool the discriminator in spite of not being realistic samples. Consequently, it can be concluded that the discriminator's accuracy throughout training cannot be always reliable in terms of realistic data generation.

This project shows the relevance of typing behaviour data generation using adversarial networks with several approaches. It has been proved that valid data to impersonate users can be generated using generative modelling from ML. A future

analysis using more GAN extensions would be of great continuity, such as a cGAN using Convolutional Neural Networks.

Indeed, Bob can now benefit from Alice's service accessing with her credentials as if he was Alice without the system recognizing him.

References

[1] Recognizing faces [and discussion]. 302(1110):423–436, 1983.

[2] Ya-Ping Huang, Si-Wei Luo, and En-Yi Chen. An efficient iris recognition system. In *Proceedings. International Conference on Machine Learning and Cybernetics*, volume 1, pages 450–454 vol.1, 2002.

[3] Le Hoang Thai and Ha Nhat Tam. Fingerprint recognition using standardized fingerprint model, 2011.

[4] G.K. Venayagamoorthy, V. Moonasar, and K. Sandrasegaran. Voice recognition using neural networks. In *Proceedings of the 1998 South African Symposium on Communications and Signal Processing-COMSIG '98 (Cat. No. 98EX214)*, pages 29–32, 1998.

[5] Eladio Alvarez Valle and Oleg Starostenko. Recognition of human walking/running actions based on neural network. In *2013 10th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE)*, pages 239–244, 2013.

[6] Alejandro Acien, Aythami Morales, Ruben Vera-Rodriguez, Julian Fierrez, and John V Monaco. Typenet: Scaling up keystroke biometrics. In *2020 IEEE International Joint Conference on Biometrics (IJCB)*, pages 1–7. IEEE, 2020.

[7] Faseela Abdullakutty, Eyad Elyan, and Pamela Johnston. A review of state-of-the-art in face presentation attack detection: From early development to advanced deep learning and multi-modal fusion methods. *Information fusion*, 2021.

[8] Abhinav Muley and Vivek Kute. Prospective solution to bank card system using fingerprint. In *2018 2nd International Conference on Inventive Systems and Control (ICISC)*, pages 898–902, 2018.

[9] Andrew Bud. Facing the future: The impact of apple faceid. *Biometric technology today*, 2018(1):5–7, 2018.

[10] Ioannis Stylios, Spyros Kokolakis, Olga Thanou, and Sotirios Chatzis. Behavioral biometrics & continuous user authentication on mobile devices: A survey. *Information Fusion*, 66:76–99, 2021.

[11] Jonathan Ness. Presentation attack and detection in keystroke dynamics.

[12] Christian Rathgeb, Pawel Drozdowski, and Christoph Busch. Makeup presentation attacks: Review and detection performance benchmark. *IEEE Access*, PP:1–1, 12 2020.

[13] Athos Antonelli, Raffaele Cappelli, Dario Maio, and Davide Maltoni. Fake finger detection by skin distortion analysis. *Information Forensics and Security, IEEE Transactions on*, 1:360 – 373, 10 2006.

[14] Deian Stefan, Xiaokui Shu, and Danfeng Daphne Yao. Robustness of keystroke-dynamics based biometrics against synthetic forgeries. *computers & security*, 31(1):109–121, 2012.

[15] Itay Hazan, Oded Margalit, and Lior Rokach. Securing keystroke dynamics from replay attacks. *Applied Soft Computing*, 85:105798, 2019.

[16] Enzhe Yu and Sungzoon Cho. Keystroke dynamics identity verification—its problems and practical solutions. *Computers & Security*, 23(5):428–440, 2004.

[17] Sungzoon Cho, Chigeun Han, Dae Hee Han, and Hyung-Il Kim. Web-based keystroke dynamics identity verification using neural network. *Journal of Organizational Computing and Electronic Commerce*, 10(4):295–307, 2000.

[18] Sonali B Maind, Priyanka Wankar, et al. Research paper on basic of artificial neural network. *International Journal on Recent and Innovation Trends in Computing and Communication*, 2(1):96–100, 2014.

[19] Claude Lemaréchal. Cauchy and the gradient method, 2012.

[20] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

[21] Tijmen Tieleman, Geoffrey Hinton, et al. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[22] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[23] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural networks*, 3(5):551–560, 1990.

[24] N Ganesan, K Venkatesh, MA Rama, and A Malathi Palani. Application of neural networks in diagnosing cancer disease using demographic data. *International Journal of Computer Applications*, 1(26):76–85, 2010.

[25] Robin Nix and Jian Zhang. Classification of android apps and malware using deep neural networks. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 1871–1878, 2017.

[26] Lucie P Aarts and Peter Van Der Veer. Neural network method for solving partial differential equations. *Neural Processing Letters*, 14(3):261–271, 2001.

[27] David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016.

[28] Murat Cihan Sorkun, Christophe Paoli, and Özlem Durmaz Incel. Time series forecasting on solar irradiation using deep learning. In *2017 10th International Conference on Electrical and Electronics Engineering (ELECO)*, pages 151–155, 2017.

[29] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. Ieee, 2013.

[30] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[31] Xuan-Hien Le, Hung Viet Ho, Giha Lee, and Sungho Jung. Application of long short-term memory (lstm) neural network for flood forecasting. *Water*, 11(7):1387, 2019.

[32] Daniel Soutner and Luděk Müller. Application of lstm neural networks in language modelling. In *International Conference on Text, Speech and Dialogue*, pages 105–112. Springer, 2013.

[33] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[34] William Fedus, Mihaela Rosca, Balaji Lakshminarayanan, Andrew M Dai, Shakir Mohamed, and Ian Goodfellow. Many paths to equilibrium: Gans do not need to decrease a divergence at every step. *arXiv preprint arXiv:1710.08446*, 2017.

[35] Farzan Farnia and Asuman Ozdaglar. Do GANs always have Nash equilibria? In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 3029–3039. PMLR, 13–18 Jul 2020.

[36] Divya Saxena and Jiannong Cao. Generative adversarial networks (gans): Challenges, solutions, and future directions. *ACM Comput. Surv.*, 54(3), May 2021.

[37] Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks.

[38] Soumith Chintala Alec Radford, Luke Metz. Unsupervised representation learning with deep convolutional generative adversarial networks.

[39] Francesco Zola, Jan Lukas Bruse, Xabier Etxeberria Barrio, Mikel Galar, and Raul Orduna Urrutia. Generative adversarial networks for bitcoin data augmentation. In *2020 2nd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*, pages 136–143. IEEE, 2020.

[40] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[41] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.

[42] Simon Canales Yi Yu, Abhishek Srivastava. Conditional lstm-gan for melody generation from lyrics.

[43] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *CoRR*, abs/1710.10196, 2017.

[44] Cristóbal Esteban, Stephanie L. Hyland, and Gunnar Rätsch. Real-valued (medical) time series generation with recurrent conditional gans, 2017.

[45] Ofir Press, Amir Bar, Ben Bogin, Jonathan Berant, and Lior Wolf. Language generation with recurrent generative adversarial networks without pre-training. *CoRR*, abs/1706.01399, 2017.

[46] Olof Mogren. C-RNN-GAN: continuous recurrent neural networks with adversarial training. *CoRR*, abs/1611.09904, 2016.

[47] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[48] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[49] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[50] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[51] Kris Cao and Marek Rei. A joint model for word embedding and word morphology. *arXiv preprint arXiv:1606.02601*, 2016.

[52] André Pimenta, Davide Carneiro, José Neves, and Paulo Novais. Analysis of mental fatigue and mood states in workplaces. In Paulo Novais, David Camacho, Cesar Analide, Amal El Fallah Seghrouchni, and Costin Badica, editors, *Intelligent Distributed Computing IX*, pages 111–119, Cham, 2016. Springer International Publishing.

[53] Fabian Monrose and Aviel D Rubin. Keystroke dynamics as a biometric for authentication. *Future Generation computer systems*, 16(4):351–359, 2000.

[54] Basma Hassan, Khaled Fouad, and Mahmoud Hassan. *Keystroke Dynamics Authentication in Cloud Computing*, pages 923–945. 01 2019.

[55] Vivek Dhakal, Anna Maria Feit, Per Ola Kristensson, and Antti Oulasvirta. *Observations on Typing from 136 Million Keystrokes*, page 1–12. Association for Computing Machinery, New York, NY, USA, 2018.

Statement of independent work

"I certify that to the best of my knowledge, this project is my own work and all support has been acknowledged."