MÁSTER EN INGENIERÍA EN BIG DATA, TECNOLOGÍA Y ANALÍTICA AVANZADA

ANÁLISIS DE GENERADORES DE CONTRASEÑAS BASADOS EN DEEP LEARNING PARA APLICACIONES CRIPTOGRÁFICAS

Autor:

Alejandro Rodríguez García

Director:

Jaime Boal Martín-Larrauri

Declaro, bajo mi responsabilidad, que el proyecto presentado con el título

"ANÁLISIS DE GENERADORES DE CONTRASEÑAS BASADOS EN DEEP LEARNING PARA APLICACIONES CRIPTOGRÁFICAS"

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2022/23 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.

Fdo.: Alejandro Rodríguez García Fecha: 13 / 07 / 2023

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Firmado digitalmente por
BOAL MARTIN LARRAURI
JAIME - 05304600H
Fecha: 2023.07.13
19:56:02 +02'00'

Fdo.: Dr. Jaime Boal Martín-Larrauri Fecha: 13 / 07 / 2023

Análisis de Generadores de Contraseñas Basados en Deep Learning para Aplicaciones Criptográficas

Autor: Alejandro Rodríguez García. Director: Dr. Jaime Boal Martín-Larrauri

Resumen **- El uso de modelos de deep learning ha estado revolucionando el estado del arte de la adivinación de contraseñas en los últimos años. En este proyecto se analizan los modelos más recientes de generación de contraseñas, especialmente aquellos que usan redes generativas antagónicas (GAN). Se han estudiado los aspectos teóricos y la implementación de los modelos poniendo el foco en *GNPassGAN*, la alternativa que se ha encontrado que presenta mejores resultados. De los resultados obtenidos se ha concluido que *GNPassGAN* se puede considerar como la mejor solución de todos los estudiados alcanzando con una tasa de adivinación de contraseñas del 7,57% en el conjunto de contraseñas *RockYou* y del 23% en el conjunto de *Pwned*. Finalmente, se han propuesto nuevas líneas de trabajo para mejorar la seguridad de los sistemas que hacen uso de contraseñas, protegiéndolos ante el uso ofensivo de los modelos de adivinación de contraseñas.**

Palabras clave **– PassGAN, GAN, contraseñas, *deep learning*, criptografía**

I. Introducción

La investigación en métodos de autenticación y predicción de contraseñas está en constante desarrollo. Estos métodos no solo son relevantes para los delincuentes cibernéticos y las personas que buscan protegerse, sino que también tienen diversas aplicaciones como las de los cuerpos de seguridad oficiales que requieren acceder legalmente a archivos o defenderse de los criminales. Por lo tanto, es vital estudiar y desarrollar estas herramientas de manera abierta y transparente.

En este trabajo se evalúan modelos de redes generativas antagónicas (GAN) del estado del arte de las técnicas de generación de contraseñas, siendo las técnicas de adivinación que presentan mejores resultados en la actualidad. El objetivo es contribuir con un análisis exhaustivo del uso de modelos basados en *PassGAN* [1], que fue el primer modelo de generación de contraseñas implementado con una GAN. La aplicación directa de estos modelos es en el ámbito de la criptografía ofensiva, donde la capacidad de poder enriquecer las bases de datos de contraseñas permite incrementar la probabilidad de romper los *hash* de contraseñas capturados entre otras muchas aplicaciones.

II. Estado del Arte

La mayoría de los usuarios tienden a establecer contraseñas simples y fáciles de recordar, como se apunta en [2], [3]. Además, normalmente no se da suficiente información a los usuarios sobre la importancia de las contraseñas seguras y se limita a mensajes como "La contraseña debe contener minúsculas, mayúsculas y números". En [4], [5]–[8], se realizan estudios a gran escala sobre la creación de contraseñas concluyendo que la mayoría de los usuarios repiten contraseñas, usan contraseñas correlacionadas entre sí y que pueden recordar fácilmente vinculándolas a objetos o acciones cotidianos. Además, muestran que no se cumplen las suposiciones de las publicaciones NIST SP 800-63 [9] donde se definen métodos de estimación de la entropía de las contraseñas generadas por el usuario en función de si se usan palabras conocidas, la longitud, el tipo de caracteres usados, etc. Posteriormente la estimación de la calidad de las contraseñas basado en la entropía se eliminó en la última versión del documento, la NIST SP 800-63-3 [10]. En su documento asociado NIST SP 800-63B [11] se expone una serie de requisitos que deben tener cada uno de los métodos de autentificación, como la longitud mínima de 8 caracteres, el uso de una *blacklist*[1] o que las contraseñas deben almacenarse como hash, con la técnica de *salted*[2] y nunca truncadas.

A. Adivinación de Contraseñas

Las contraseñas pertenecen según lo definido en NIST SP 800-63 [9] a los factores de autentificación denominados como de conocimiento. Esto significa que son vulnerables a los ataques de adivinación, si se conoce lo que se presupone como "secreto" se pierde la capacidad de autenticación.

Los principales métodos de adivinación de las contraseñas son aquellos que se en conseguir las contraseñas reales de los usuarios almacenadas en los sistemas. Son ataques como el *phishing* [12] y sus múltiples variantes, malware diseñado para recopilar credenciales de un sistema, *SQL Injection* como en una de las filtraciones más famosas, *RockYou* [13], *cold boot attack* [14] donde se deja sin alimentación al dispositivo y se carga sobre este un sistema operativo ligero para leer la memoria temporal, y cualquier otra técnica de ingeniería social que persiga obtener credenciales [15].

Con estas fugas de información se pueden realizar distintos ataques [16], [17] tanto de fuerza bruta (probar iterativamente

[1] Base de datos con el histórico de contraseñas filtradas.

[2] Número de dígitos aleatorios que se le agrega a la contraseña y que el usuario no conocerá para complicar el descifrado.

múltiples combinaciones de números, letras y caracteres especiales con herramientas como herramientas como *Crunch* [18]), ataques basados en diccionarios o *Rainbow Attack* (generarse listas de *hash* de contraseñas filtradas [19], [20]).

Sin embargo, en donde se ha puesto el foco en las últimas décadas es en cómo se puede incrementar el tamaño de los dataset de contraseñas filtradas con contraseñas generadas a partir del conocimiento de como son las contraseñas escogidas por las personas. La primera propuesta fue el uso de *modelos de Márkov* [21], el modelado del lenguaje natural de Shannon [22] y los *Probabilistic Context-Free Grammars* (PCFGs) [23] que aprenden reglas a partir de la probabilidad de consecución de los caracteres. Con la aparición de las nuevas técnicas de aprendizaje automático se han hecho diversas propuestas como el uso de redes *Multi-Layer Perceptron* (MLP) en [24], de *Long Short-Term Memory* (LSTM) en [25]–[27] o con *progressive learning* (PL) [27].

Lo cierto es que dichas aproximaciones no han tenido la repercusión que sí ha tenido *PassGAN* [1]. Su principal aportación es la introducción de las redes antagónicas generativas (GAN) dentro del ecosistema de predictores de contraseñas, consiguiendo según afirman sus autores, una mejorar de la capacidad de predicción de contraseñas de las herramientas de adivinación tradicionales como *HashCat* [28] y *John the Ripper (JTR)* [29] entre un 51% y 73%.

B. Redes Generativas Antagónicas

Las GAN, propuestas por primera vez por *Goodfellow et al.* en [30], definen una arquitectura donde hay un bloque que produce candidatos, el generador G, y otra que los evalúa e intenta diferenciarlos de los de un conjunto de datos real, el discriminante D. Para la definición de la arquitectura y el protocolo de entrenamiento existe una gran variedad de alternativas que dependen de la aplicación. *PassGAN* se diseñó basándose en la arquitectura *Wasserstein*, en concreto en la mejora del modelo propuesta en [31].

La acogida de *PassGAN* ha sido muy amplia en la comunidad científica y fruto ello son los numerosos trabajos que lo analizan, como [32] y [33] donde usan el modelo con distintas bases de datos, o en [17] y [34] que consiguen una mejora de entre un 5% y 10% realizando ajustes en [1].

A parte de los estudios que analizan a [1] se han hecho distintas propuestas:

- [33] *GS-PassGAN* que utiliza la relajación *Gumbel-Softmax* y el *S-PassGA*N que utiliza una representación suave de una contraseña real obtenida por un autoencoder adicional.
- [35] realiza un estudio de las distintas técnicas de aprendizaje profundo aplicables a la predicción de contraseñas y proponen un nuevo modelo, *GNPassGAN*, donde consiguen una mejora del 88,03% en la adivinación de contraseñas y generan un 31,69% menos de duplicados frente a *PassGAN*.
- [36] propone un enfoque no probabilístico de generación que se centra y otro probabilístico que es capaz de generar con mayor frecuencia las contraseñas más comunes. También, analiza otras alternativas como el uso de autoencoders, redes neuronales compuestas por dos elementos, un encoder que comprime la información y un decoder que la descomprime, también para generar contraseñas. El modelo propuesto lo bautizan como *Context Wasserstein Autoencoder* (CWAE) y su arquitectura está basada en *Wasserstein Autoencoder* (WAE) [37].
- [38] se centra en construir un modelo ligero que pueda reducir el tiempo necesario para su entrenamiento pero que siga manteniendo la misma capacidad predictiva de *PassGAN*. Su propuesta es *VAEPass*, un modelo basado en un *Variational Auto-Encoder* (VAE) que consta de un codificador y un decodificador establecidos mediante una *Gated Convolutional Neural Network* (GCNN). Sus resultados son una mejora de entre un 2,7% y 9,3% frente a los resultados de *PassGAN*. Además, siguiendo sus objetivos de implementar un modelo ligero, consiguen que los parámetros de *VAEPass* sean aproximadamente el 32% de los de *PassGAN* y el tiempo de entrenamiento requerido sea aproximadamente el 11% del requerido por *PassGAN*.

III. Análisis Teórico de los Modelos

Para poder evaluar *PassGAN* y los modelos relacionados con él es necesario entrar en el detalle teórico de cómo es cada uno. Todos los modelos analizados nacen a raíz de estudios con un mismo hilo conductor, el conseguir estabilizar el entrenamiento de las GAN. Este es uno de los principales retos de las GAN, especialmente, hablando de la generación de texto debido a que si en el proceso de entrenamiento no se encuentran en igualdad de condiciones al generador y al discriminante siempre ganará el mismo y, por lo tanto, no se retroalimentaran en el aprendizaje.

A. WGAN

Para resolver el problema de la estabilización del entrenamiento se proponen el modelo *WGAN* en [39]. Su principal mejora con respecto al GAN tradicionales es el uso de la distancia de *Wasserstein* para medir la diferencia entre la distribución de probabilidad de las muestras generadas y la distribución real de los datos. La distancia de *Wasserstein*, también conocida como distancia de *Earth Mover* (EMD), proporciona una métrica más significativa y estable para cuantificar la disparidad entre las distribuciones.

La distancia de *Wasserstein* entre dos distribuciones de probabilidad $\boldsymbol{P}$ y $\boldsymbol{Q}$ se define como:

$$\boldsymbol{W(P,Q) = \inf_{\gamma} \int c(x,y)\ d\gamma(x,y)}$$

donde $\boldsymbol{\gamma}$ es una medida de probabilidad en el espacio de muestra $\mathbf{X} \times \mathbf{Y}$, $\boldsymbol{c(x,y)}$ es una función de coste que mide el coste de transportar una unidad de masa desde $\boldsymbol{x}$ a $\boldsymbol{y}$, y el

$\inf_{\gamma}$ significa que se toman todas las medidas de probabilidad γ que tienen P y Q como marginales.

La función de pérdidas se define como:

$$L(D_W, G) = E[D_W(x)] - E\left[D_W\left(G(z)\right)\right]$$

donde $G(z)$ es la muestra generada por el generador a partir de un vector de ruido z, x es una muestra real de los datos, y $\mathbf{E}[.]$ es la esperanza. El objetivo es minimizar esta función de pérdida con respecto a los parámetros del generador G y maximizarla con respecto a los parámetros del discriminador D_W.

B. IWGAN

El modelo *Wasserstein GAN* (WGAN) utiliza la distancia de *Wasserstein* para evitar las limitaciones en el entrenamiento de GANs, pero tiene otros defectos como el "colapso de modos" y la falta de una métrica para detectar la convergencia. Se conoce como "colapso de modos" cuando el generador produce una variedad limitada de salidas, ignorando u omitiendo la diversidad en los datos reales.

En [31] se introduce un nuevo modelo llamado *Improved Wasserstein GAN* (IWGAN). La principal diferencia que distingue a *IWGAN* de *WGAN* es la introducción de un término de regularización en la función de pérdida del generador.

En *IWGAN*, se utiliza una función de pérdida mejorada para el generador que incluye un término adicional de regularización. Este término se basa en la divergencia de Jensen-Shannon (JS) entre la distribución generada y una distribución de referencia, como la distribución de datos reales.

La función de pérdida del generador en *IWGAN* se define como:

$$[L_G = -E\left[\log\left(D(x)\right)\right] + E\left[\log\left(D\left(G(z)\right)\right)\right] + \lambda E[JS(P_r, P_g)]]$$

Donde(E) denota el valor esperado, $(D(x))$es la salida del discriminador para una muestra real (x), $(G(z))$ es la muestra generada por el generador a partir de un vector de ruido (z), (P_r) es la distribución de referencia (distribución de datos reales), (P_g) es la distribución generada por el generador, (λ) es un parámetro de regularización y (JS) representa la divergencia de Jensen-Shannon.

Al introducir este término de regularización basado en la divergencia de JS, *IWGAN* busca mejorar la estabilidad y la calidad de las muestras generadas, promoviendo una mejor correspondencia entre las distribuciones generada y de referencia.

C. WGAN-GP y PassGAN

El siguiente trabajo realizado con el objetivo de estabilizar aún más el entrenamiento fue con *WGAN-GP* [40], que propone una penalización del gradiente conocida como la restricción de *Lipschitz*. La restricción de *Lipschitz* se refiere a limitar la magnitud de las derivadas parciales del discriminador en relación con las entradas.

En términos matemáticos, la restricción de *Lipschitz* se puede expresar de la siguiente manera:

$$[|\nabla \mathcal{D}_W(x)| \leq K]$$

Donde $(|\nabla \mathcal{D}_W(x)|)$ representa la magnitud del gradiente del discriminador en un punto de entrada (x), y (K) es una constante que limita dicha magnitud.

En el entrenamiento se busca ajustar los pesos del discriminador de manera que se cumpla esta restricción de *Lipschitz*. Esto se logra mediante la penalización del gradiente, que se añade a la función de pérdida. La penalización del gradiente ayuda a suavizar las funciones de activación y limitar las derivadas parciales del discriminador, lo cual promueve una mayor estabilidad en el entrenamiento.

Para garantizar que D_W satisfaga la condición de *Lipschitz* con una constante K, se utiliza la penalización de gradiente. La función de pérdida actualizada con la penalización de gradiente se denomina *WGAN-GP*.

La penalización de gradiente se aplica agregando un término adicional a la función de pérdida del *WGAN*. En la formulación original del *WGAN-GP*, este término se calcula mediante una media entre muestras reales y generadas. Se penaliza cuando el gradiente de la función discriminador D_W excede un umbral.

La función de pérdida queda:

$$L_{\text{WGAN-GP}}(D_W, G) = E[D_W(x)] - E\left[D_W\left(G(z)\right)\right] + \lambda E[(|\nabla_{\tilde{x}} D_W(\tilde{x})|_2 - 1)^2]$$

donde x es una muestra real de los datos, z es un vector de ruido de entrada al generador G, $(D_W(x))$ es la salida del discriminador para una muestra real x, $(D_W(G(z)))$ es la salida del discriminador para una muestra generada $(G(z))$, $(\tilde{x})$ es una interpolación entre (x) y $(G(z))$ definida como $(\tilde{x} = \epsilon x + (1 - \epsilon)G(z))$ con (ϵ) muestreado de una distribución uniforme entre 0 y 1, (λ) es un hiperparámetro que controla la importancia relativa de la penalización de gradiente, $(|\cdot|_2)$ es la norma (L_2) y $(\nabla_{\tilde{x}})$ es el gradiente con respecto a $(\tilde{x})$.

PassGAN se basa la arquitectura *WGAN-GP*, utilizando las técnicas de penalización de gradiente para lograr una función continua 1-Lipschitz. No utiliza ninguna implementación novedosa ni del gradiente ni del entrenamiento ni de la arquitectura.

D. GNPassGAN

GNPassGAN es una variante de *PassGAN* que incorpora varias propuestas para mejorar también la estabilidad. Los aspectos innovadores respecto a los anteriores de este modelo son:

1) Normalización de gradientes

En *GNPassGAN* se agrega la normalización de gradientes propuesta en [41] y bautizada como *Adaptive Gradient Normalization* (AGN). Al normalizar los gradientes, se ayuda a mitigar el problema de los gradientes que crecen exponencialmente o se anulan, lo cual dificulta el entrenamiento. Esto se logra mediante el cálculo de un factor

de normalización adaptativo para cada gradiente parcial (derivada parcial de la función objetivo con respecto a un parámetro concreto del modelo). El factor de normalización adaptativo para un gradiente parcial (∇) se define como:

$$\mathcal{N} = \frac{|\nabla|_2}{\max(|\nabla|_2, \epsilon)}$$

Donde ($|\nabla|_2$) es la norma euclídea del gradiente y (ϵ) es una pequeña constante positiva para evitar divisiones por cero. El gradiente normalizado se obtiene multiplicando el gradiente parcial (∇) por el factor de normalización adaptativo:

$$\nabla_{norm} = \mathcal{N} \cdot \nabla$$

El gradiente normalizado (∇_{norm}) se utiliza entonces para actualizar los parámetros del modelo durante el proceso de optimización. La idea detrás de la normalización del gradiente adaptativa es asegurar que los gradientes tengan magnitudes similares y evitar que algunos gradientes dominen sobre otros.

2) Función de activación

En *GNPassGAN*, se modifica la función de activación en la última capa del generador, pasando de una *softmax* a una *tanh*. La función de activación *softmax* se utiliza comúnmente en tareas de clasificación binaria [0,1], sin embargo, *tanh* aplana la salida del generador al rango [-1, 1]. El motivo de por qué se cambia la función de activación no lo explican en el artículo donde se publicó [35], pero se intuye que es por la función de perdida escogida.

3) Función de pérdidas

En lugar de utilizar la pérdida de *Wasserstein* como en *PassGAN*, *GNPassGAN* emplea la pérdida de entropía binaria dentro de una capa sigmoide como función de pérdida. La pérdida de entropía binaria es una función de pérdida comúnmente utilizada en tareas de clasificación binaria. Al utilizar esta función de pérdida, *GNPassGAN* busca optimizar el generador y el discriminador para la clasificación binaria, específicamente para distinguir entre contraseñas reales y contraseñas generadas. La capa sigmoide en la función de pérdida asegura que la salida esté limitada entre 0 y 1, representando la probabilidad de que la contraseña generada sea real.

4) *Análisis del entrenamiento*

En cada iteración se actualiza tanto el generador como el discriminador. En cada iteración, se selecciona un lote de ejemplos de entrenamiento y se realiza una actualización del discriminador, seguida de una actualización del generador. Por lo tanto, el número de veces que se actualizan el discriminador y el generador depende de la cantidad de iteraciones sobre la actualización del generador y del número de veces que se actualiza el discriminante por cada actualización del generador. Estos valores determinan cuántas veces se actualizan los parámetros del discriminador y el generador respectivamente en cada iteración del bucle de entrenamiento.

Otro de los parámetros importantes a la hora de entrenar *GNPassGAN* es el tamaño del lote (*batch size*). Esté es la cantidad de ejemplos de entrenamiento que se procesan en paralelo antes de que se realice una actualización de los parámetros del generador y el discriminador. El uso de un tamaño de lote más grande puede acelerar el proceso de entrenamiento al permitir cálculos en paralelo, pero también puede requerir más memoria. Por otro lado, un tamaño de lote más pequeño puede ser útil si existen limitaciones de memoria o si se desea una actualización más frecuente de los parámetros del modelo. Otro de los aspectos importantes en los que puede afectar el tamaño del lote es en el ruido con el que aprende el modelo. Un tamaño de lote pequeño puede provocar que el modelo aprenda más rápidos porque usa una menor cantidad de datos para esperarse a actualizar los pesos. Sin embargo, esto no significa que aprenda en la dirección correcta, al actualizar sus parámetros basados en un volumen pequeño de datos la probabilidad de equivocarse es mayor. En el caso de escoger un tamaño de lote muy grande pasa lo contrario, basas la actualización de los parámetros del modelo en muchos datos por lo que avanzas en una dirección mucho más correcta. En contraposición, en este caso, el entrenamiento se vuelve muy lento y el coste de converger en una solución óptima es muy elevado

E. GS-PassGAN

Por último, el modelo *GS-PassGAN* propuesto en el artículo [42], se centra en un problema distinto al descrito en los anteriores. Implementa la relajación *Gumbel-Softmax* para abordar el problema de la naturaleza discreta de las contraseñas en el entrenamiento.

En *PassGAN*, las contraseñas reales se representan como secuencias de representaciones *one-hot* de sus caracteres. Esto es un tipo de codificación básica que codifica una contraseña como una matriz binaria en la que cada fila le corresponde un carácter de la contraseña y una cada columna por cada posible carácter. La columna tendrá un valor de 1 si es la correspondiente con la del carácter. Sin embargo, al generar contraseñas falsas (por el generador de la GAN) las salidas son las de una capa *softmax*, no son números enteros, y el discriminante puede aprenderse estas características para distinguir entre las contraseñas reales y falsas.

Para superar esta limitación, *GS-PassGAN* utiliza la relajación *Gumbel-Softmax*. Esta técnica permite en lugar de muestrear directamente las contraseñas falsas utilizando la capa *softmax*, se utiliza una aproximación basada en la distribución Gumbel con la función *softmax*.

En la siguiente ecuación, las salidas del generador se denotan como $\boldsymbol{h_j}$, donde $\boldsymbol{j}$ representa el carácter en la contraseña generada. En lugar de muestrear directamente $\boldsymbol{y_j}$, la representación *one-hot*, se utiliza la siguiente aproximación diferenciable:

$$\boldsymbol{y_j} = \boldsymbol{softmax}\left(\frac{\mathbf{1}}{\boldsymbol{\tau}(\boldsymbol{h_j} + \boldsymbol{g})}\right)$$

En esta ecuación: ***softmax*** es la función *softmax* que calcula las probabilidades de cada carácter, $\boldsymbol{\tau}$ es un parámetro que controla la suavidad de la distribución resultante, y $\boldsymbol{g}$ es un

vector de ruido independientes que sigue una distribución *Gumbel* con media cero y escala unitaria.

IV. Análisis Empírico

A. Evaluación de los Repositorios de Código

En la siguiente tabla se resume todos los repositorios analizados junto con los artículos relacionados con *PassGAN* descritos en el estado del arte que proponen mejoras y arquitecturas relacionadas.

La tabla está ordenada según la columna de "Similitud original", columna que ordena por similitud respecto el artículo original. La similitud se ha establecido, en primer lugar, considerando los artículos asociados a cada repositorio de código: que diferencian a su arquitectura respecto a la original, funciones de perdidas, etc. En segundo lugar, se ha analizado el código prestando detalle si se han usado clases y una estructura similar a la del repositorio original. En tercer lugar, se ha usado como criterio diferenciador la versión de Python y de las librerías.

GitHub Link Resumn	Autor	Python	Pytorch / Tensorflow	Similitud original	Referencia Paper Anexo
PassGAN	brannond	2	TF 1.4.1	1	[1]
PassGAN	d4ichi	3.x	TF 1.13	2	
PassGAN	rarecoil	3.x	TF 1.15	3	
PassGAN	Karlosse	3.x	TF 2	4	
PassGAN	Riathoir	3.7	TF 2.1	5	
PassGAN	r-khanna	3.x	PT 1.4	6	
PassGAN Based Honeywords	.	.	.	7	[43]
PassGAN con PCFG	.	.	.	8	[27]
PLR	pasquini-dario	3.x	TF 1.14	9	[36]
GNPassGAN	fangyiyu	3.x	PT 1.10	10	[35]
Adversarial Password Cracking	.	.	.	11	[33]
GS-PassGAN y S-PassGAN	.	.	.	12	[42]
VAEPass	.	.	.	13	[38]
LSTM con RNN	cupslab	3.x	TF 1.4	14	[26]
Improved WGAN Training	igul222	3.x	TF	15	[31]
WGAN PT	kzkadc	3.x	PT	16	
wgan_pytorch	Alexrgg	3.x	PT	17	.

Figura 1:Estado del Arte de Modelos y Repositorios Relativos a PassGAN

Del análisis del estado del arte de la implementación de los modelos disponibles, se pueden extraer varias conclusiones:

• En la mayoría de los casos no están publicados los códigos en los que se basan los artículos. Y de los que si hay código publicado casi ninguno se realizó por los autores, si no por implementaciones de particulares a posteriori.

• La documentación de los repositorios es escasa. Aunque cuentan con README.md no suelen describir el entorno en el que lo implementaron y las versiones específicas de las librerías usadas mediante un requirements.txt, environment.yml o similares.

• Debido a que las dos librerías de deep learning usadas (*TensorFlow* y *PyTorch*) han sufrido y siguen sufriendo constantes evoluciones de sus versiones es necesario una constante adaptación del código. A pesar de que siempre se puede trabajar con entornos de Python donde con versiones antiguas de las librerías locales, en el caso de *PassGAN* es un problema relevante ya que está basado en las primeras versiones de *Tensorflow* 1. La forma de estructurar tanto la arquitectura como el entrenamiento difiere bastante de como se hace en las últimas versiones y el uso de Python 2 lo complica aún más.

En general se puede afirmar que a pesar de tratarse de un proyecto basado en software la replicabilidad de los experimentos es compleja por los motivos enumerados anteriormente. Para asegurar la estandarización de los experimentos se decide partir del código *GNPassGAN* para evaluar la propuesta de usar GANs para la generación de contraseñas.

B. Metodología de entrenamiento de GNPassGAN

En el proceso de entrenamiento, se ha guardado el modelo en cada diferente nuero de iteraciones de entrenamiento para poder, posteriormente, evaluar cómo evoluciona. El modelo se ha entrenado utilizando el conjunto de contraseñas filtradas llamado *RockYou* [13]. Este conjunto se dividió en un 75% para el conjunto de entrenamiento y un 25% para el conjunto de test. La selección de ejemplos para cada conjunto se ha realizado de forma aleatoria. En total, el conjunto de contraseñas contiene 14.344.391 registros, estructurados con una contraseña por línea y con el carácter final '\n'.

El tamaño del lote (batch size) utilizado en el entrenamiento es de 64, lo que significa que en cada iteración se procesan 64 pasadas sobre el generador. Además, se ha establecido que el valor de critic_iters sea 10, lo que implica que se realizan 10 actualizaciones de los parámetros del discriminador por cada iteración del generador. En consecuencia, en cada iteración se utilizan un total de 64 * 10 = 640 contraseñas para el entrenamiento.

Teniendo en cuenta que el conjunto de contraseñas tiene un total de 14.344.391 registros, y considerando que el conjunto de entrenamiento representa el 75% del total, se utilizan aproximadamente 14.344.391 * 0.75 / (64 * 10) ≈ 17.000 iteraciones para recorrer completamente el dataset durante el entrenamiento. Es importante mencionar que este cálculo depende de la proporción del conjunto de entrenamiento, del tamaño del lote y el número de actualizaciones del discriminador escogidas en este caso en particular.

En cuanto a la longitud de las contraseñas se ha establecido

una longitud máxima de 12 caracteres. Cuando una contraseña excede esta longitud, la contraseña no se usa para el entrenamiento. Una alternativa podría ser seleccionar los primeros caracteres hasta alcanzar la longitud máxima fijada, descartando el resto de la contraseña, pero en este caso estaríamos entrenando el modelo con contraseñas que no son completamente reales.

Una vez entrenado el modelo el objetivo es cargar únicamente el generador y usarlo para crear un nuevo conjunto de contraseñas. Para cada uno de estos modelos guardados, se han generado un total de 10^8 contraseñas en cada número de iteración distinto analizados. Esto significa que se han producido 600 millones de contraseñas (10^8 por cada número de iteración distinto analizados), una cantidad masiva que permite explorar una amplia variedad de opciones y patrones durante el proceso de generación.

C. Análisis del entrenamiento de GNPassGAN

Se ha estudiado la evolución de las pérdidas del entrenamiento. La función de pérdida se usa, en este caso en concreto, para medir la diferencia entre la probabilidad que el discriminador asigna a un conjunto de muestras reales y sintéticas de que sean reales.

Tal y como está definida, cuando la contraseña es falsa la salida del discriminante se compara contra un 0, las pérdidas del discriminante serán de 0.7 si no se equivoca nada (si su salida es un 0 aunque pueda tomar valores negativos) y de 1.3 si se equivoca por completo. Por otro lado, si la salida del discriminante tiene que ser un 1 (se introduce contraseña real), las perdidas serán de 0.3 si acierta y de 1.2 si se equivoca por completo (da como salida un -1). Las pérdidas totales son la suma de ambas pérdidas, dicho valor se define como el coste del discriminante.

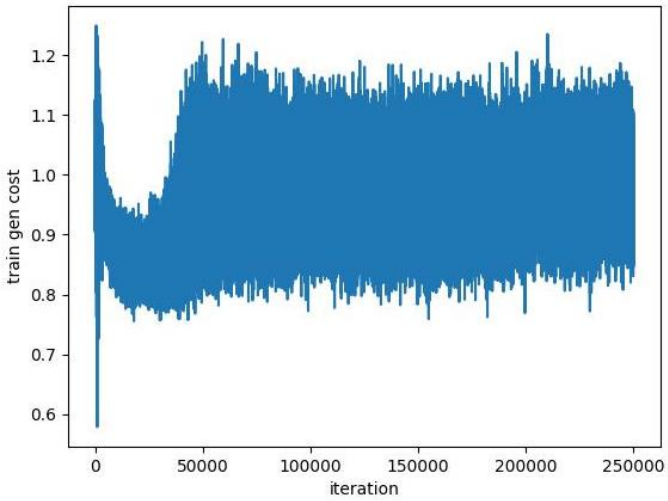

Figura 2:Evolución de las pérdidas del generador para cada iteración

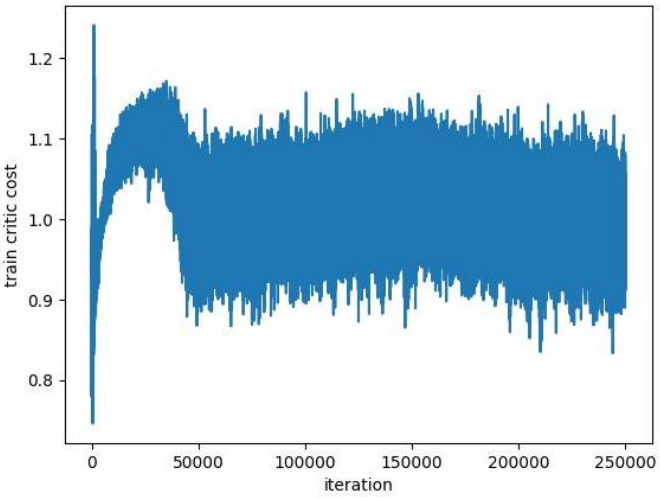

Figura 3:Evolución de las pérdidas del discriminante para cada iteración

Se puede observar cómo en las primeras iteraciones la función de costes tanto para el discriminador como para el generador oscila entre 1.2 y 0.6. A medida que se suceden las iteraciones la función de costes del discriminante tiende a crecer y la del generador a decrecer, es decir, que el discriminador comete menos error. Llega un punto en el que esto se revierte y se estabilizan las dos entorno al 1.

se han analizado la distribución de N-Grams con el cálculo de la divergencia de Jensen-Shannon de las contraseñas generadas como métrica alternativa. La divergencia de Jensen-Shannon (JSD) es una medida de similitud entre dos distribuciones de probabilidad. La JSD se basa en la entropía de la distribución, cuantizando la incertidumbre del conjunto.

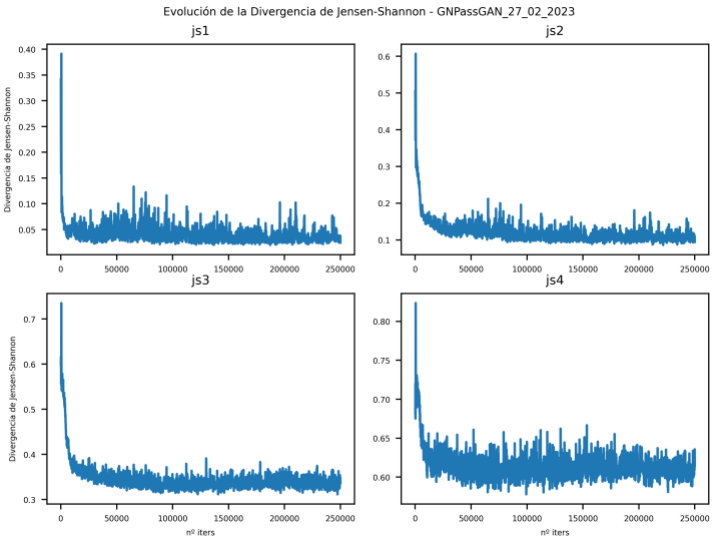

Figura 4: Evolución de la divergencia de Jensen-Shannon para n-grams de hasta 4 caracteres

Lo primero que destaca es que la divergencia decrece significativamente en las primeras iteraciones y una vez se superan las 20.000 iteraciones se estabiliza. También, destaca que para un carácter se suele equivocar o divergir muy poco de la distribución original a lo largo del entrenamiento. Cuando

aumenta el número de caracteres se vuelve más complejo el acierto, por lo que la media del error aumenta y hay iteraciones que obtiene resultados muy buenos y otras muy malos.

D. Análisis de las contraseñas generadas

Para analizar las contraseñas generadas, la primera métrica estudiada ha sido el *accuracy*, calculada como el número de contraseñas generadas que se encuentran en el conjunto *RockYou*.

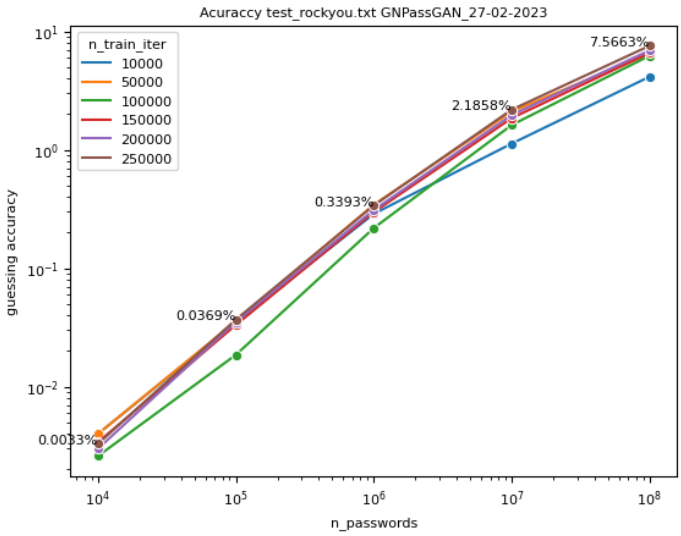

Figura 5: Accuracy del modelo entrenado con RockYou para distinto número de iteraciones y cantidad de contraseñas generadas

Se ha observado que existe una relación prácticamente lineal entre el incremento del *accuracy* y el número de contraseñas generadas. Comparando los resultados con el modelo entrenado por los autores, se ha logrado una mejora de más del 2% respecto al modelo pre-entrenado del repositorio como de los resultados reflejados en el artículo [35]. Además, se ha observado que el punto óptimo en términos de *accuracy* se alcanza después de 250.000 iteraciones de entrenamiento. Esto implica que a medida que el modelo se entrena durante más iteraciones, se logra una mayor precisión en la generación de contraseñas que coinciden con el conjunto de prueba.

Hay que destacar que el modelo de generación de contraseñas supera significativamente la probabilidad de acierto que se obtendría al generar contraseñas de forma aleatoria. La probabilidad de que al menos una de las 10^8 palabras generadas aleatoriamente esté contenida en el archivo de 3.586.097 palabras es extremadamente baja, aproximadamente $7'59 \cdot 10^{-8}$. Por lo tanto, un *accuracy* de tan solo 0'5% ya es una mejora abismal respecto al uso de caracteres aleatorios.

También se ha estudiado el *accuracy* consultando en la API de *Pwned* [44] para evaluar la efectividad del modelo frente a distintitas base de datos de contraseñas filtradas, siendo la base de datos de *Pwned* la mayor de contraseñas filtradas existente.

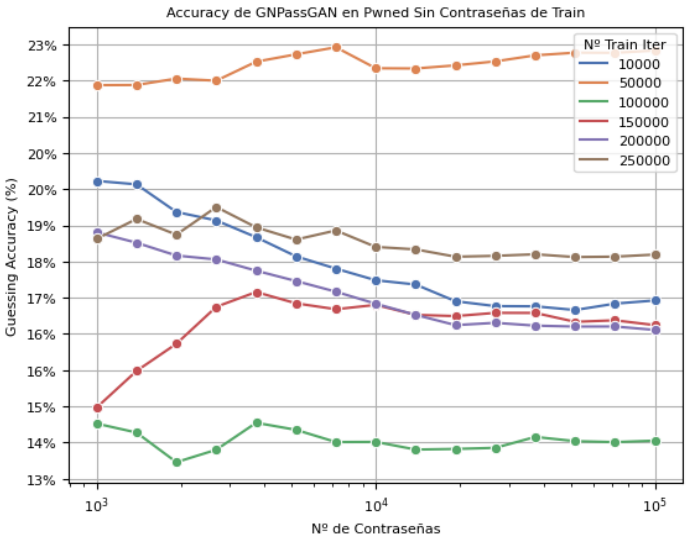

Figura 6: Accuracy del modelo entrenado en la base de datos Pwned

En general, se observa que el *accuracy* no aumenta de manera consistente al incrementar el número de contraseñas evaluadas. Algunos casos muestran un decremento seguido de estabilización, como ocurre en los modelos entrenados con 250,000, 10,000 y 200,000 iteraciones. Por otro lado, también se observa un aumento significativo en el *accuracy* según aumenta el número de contraseñas para el modelo de 150,000 iteraciones. Las variaciones en el *accuracy* para todos los rangos de tamaños analizados no superan el 3%.

Hay que destacar que al considerar 100.000 contraseñas (el dato de *accuracy* más significativo), se obtiene un *accuracy* para el modelo de 50.000 iteraciones que es un 5% más alto en comparación con el siguiente modelo más cercano, el de 250.000 iteraciones. Basándonos en estos resultados se puede afirmar que el modelo esté sobre aprendiendo el estilo de contraseñas de *RockYou* y por eso a medida que aumenta el número de iteraciones el *accuracy* en el conjunto de validación aumente. Pero, al contrastarlo con una base de datos genérica, con miles de millones de contraseñas filtradas esta mejora no aumenta porque el modelo se ha especializado en el estilo de contraseñas de *RockYou* y no generaliza bien.

Es interesante analizar también que el modelo está siendo capaz de generar contraseñas con millones de filtraciones y que sin embargo no estaban en el conjunto de entrenamiento. Por ejemplo, destacan palabras como 'amanda', 'ariana', 'carina' o 'panama' que son nombres españoles comunes.

Otra métrica estudiada es la proporción entre el número total de contraseñas generadas y el número de contraseñas únicas.

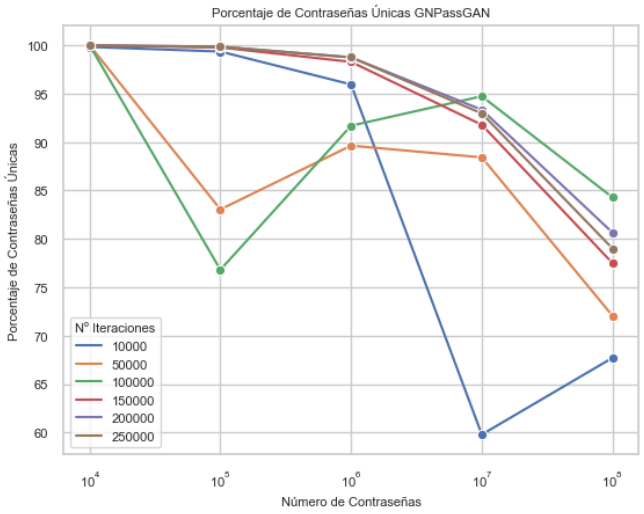

Figura 7: Porcentaje de contraseñas únicas generadas

Se ha observado una tendencia en la disminución de la entropía del modelo a medida que se incrementa el número de contraseñas generadas. Esto implica que, a medida que se generan más contraseñas, se observa una proporción mayor de contraseñas repetidas en comparación con el total.

Estos resultados plantean la pregunta de hasta qué punto se pueden generar contraseñas nuevas en función del tamaño del conjunto de datos con el que se ha entrenado. Existirá un punto en el que el aumento del número de contraseñas generadas ya no conlleva un aumento significativo en el porcentaje de contraseñas únicas. Hay que considerar que el mayor decremento, cuando se pasa de 10^7 a 10^8 es del 15% de contraseñas únicas, se podría argumentar que es un decremento significativo, pero es que has multiplicado por 10 el número de contraseñas generadas. Esto significa que un con 10^8 has generado 80 millones de contraseñas únicas, frente a los 9'5 millones con 10^7 por lo que sigue saliendo muy a cuenta continuar generando contraseñas, aunque la eficiencia disminuya.

En relación con las contraseñas únicas se ha estudiado que porcentaje de contraseñas distintas genera el modelo entrenado para distintas iteraciones. Para ello, se ha calculado usando el porcentaje de contraseñas comunes entre las generadas por cada modelo entrenado con distinto número de iteraciones.

Los resultados obtenidos revelan que, en general, el porcentaje de contraseñas distintas entre archivos de cada iteración es muy alto. Puede indicar un aspecto positivo, que el modelo continúa aprendiendo y no ha alcanzado un punto de saturación en su capacidad de generación de contraseñas. En el lado opuesto puede ser un indicador negativo, el entrenamiento del modelo es muy ruidoso ya que entre iteraciones consecutivas sus contraseñas generadas son muy distintas. En este último punto hay que considerar que el criterio para decir que una contraseña es distinta o no es muy "estricto", ya que contraseñas como "love" y "lover" son muy parecidas, pero a su vez se consideran como contraseñas distintas.

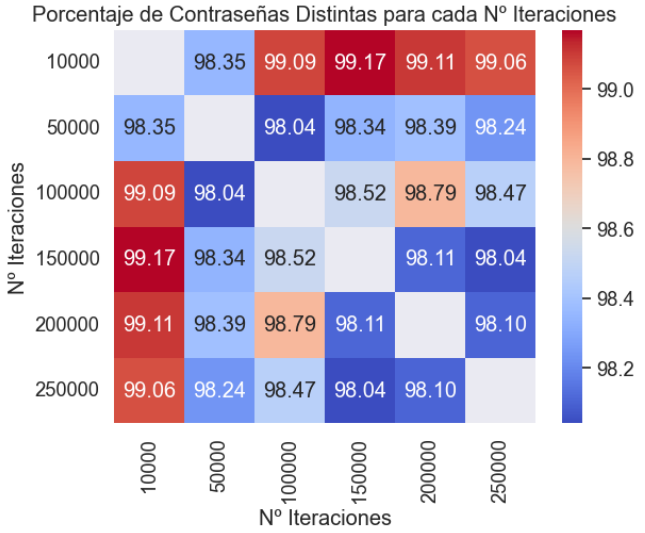

Figura 8: Porcentaje de contraseñas distintas generadas para distinto número de iteraciones en el entrenamiento

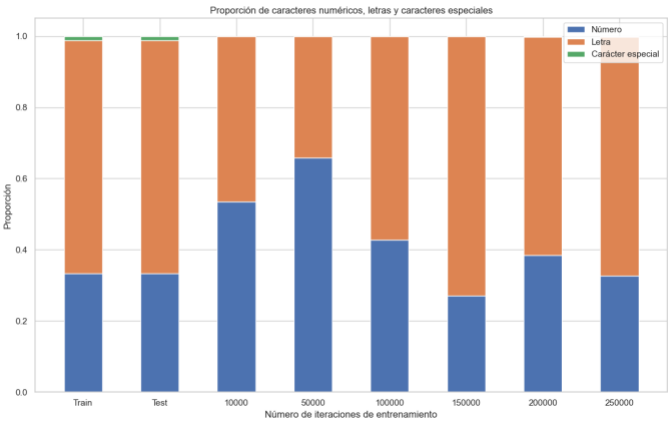

Figura 9: Proporción de caracteres numéricos, letras y caracteres especiales

La proporción de caracteres numéricos y letras se ha considerado como otra métrica importante para determinar si el modelo genera distribuciones similares a las del conjunto de contraseñas original.

En los resultados obtenidos, se puede observar una variación en la proporción de números presentes en las contraseñas generadas por los modelos entrenados con diferentes iteraciones. En particular, destaca que los modelos entrenados con menos iteraciones muestran una proporción mayor de números en las contraseñas generadas. Sin embargo, se observa un cambio significativo en esta tendencia con el modelo entrenado con 50.000 iteraciones, donde se logra alcanzar la mayor proporción de números. Esto sugiere que el discriminador del modelo puede estar distinguiendo de manera efectiva las contraseñas falsas, ya que estas tienden a tener una mayor presencia de números.

Es interesante destacar que, para el modelo entrenado con 250.000 iteraciones, la proporción de números en las contraseñas generadas es similar a la encontrada en el conjunto de entrenamiento y prueba utilizado en el modelo. Esto podría indicar un mayor nivel de convergencia y consistencia entre el

modelo y los datos de entrenamiento.

V. Conclusiones

Se ha podido extraer distintas conclusiones en el análisis de *GNPassGAN* destacando que es uno de los modelos publicados que presentan como una mejora significativa frente a *PassGAN*.

La primera es lo crítico que supone ser capaz de estabilizar el entrenamiento de estos modelos. Esto no es una conclusión novedosa en el sentido de por ser una GAN siempre se presupone la complejidad de entrenarlas para que la lucha entre el generador y el discriminante esté igualada. Por ello es importante que el número de veces que gana el generador y el discriminante esté equilibrado a lo largo del entrenamiento para que vayan aprendiendo los dos y realimentarse. Sin embargo, esto provoca que la función de perdidas oscile significativamente y en consecuencia el aprendizaje del generador. Se ha concluido en las medidas que *GNPassGAN* consigue mejorarlo significativamente con la normalización del gradiente y el cambio de la función de perdidas, consiguiendo unas pérdidas medias de 1.01 para el discriminante y de 0.98 para el generador, muy igualadas.

La segunda conclusión, es que el generador consigue aprender con tan solo 20.000 iteraciones la distribución de los n-grams de las contraseñas de 1 a 4 caracteres (medido con la JSD). Pasadas las 20.000 iteraciones se podría afirmar que se estabiliza dejando de aprender, pero sin embargo se ha observado que el *accuracy* sigue incrementando incluso en la 250.000 iteraciones. Por lo tanto, aunque se tiene que considerar la JSD como una métrica para identificar que aprende no es una medida concluyente. Además, como métrica alternativa se ha identificado que la proporción de números frente a las letras y caracteres especiales no es constante, va evolucionando a lo largo del entrenamiento. En las primeras iteraciones se generan más contraseñas conformadas por más números, evoluciona hacia una menor cantidad de número y termina el entrenamiento con 250.000 iteraciones con una proporción idéntica a las del conjunto de entrenamiento y test. Esto nos refleja que, aunque el JSD permanece constante (con algo de ruido) a partir de las 20.000 iteraciones, necesita hasta 250.000 iteraciones para conseguir la misma distribución que en el conjunto de test de números y letras.

La tercera conclusión, es el ruido con el que va aprendiendo el modelo. Esto no solo se ha podido identificar con el análisis de la evolución función de pérdidas, si no que se ha estudiado cómo de distintas son las contraseñas generadas para distinto número de iteraciones. Se ha identificado que para los modelos guardados cada 50.000 iteraciones entre el 99% y 98% de las contraseñas generadas por cada modelo son distintas.

La cuarta conclusión, es que a medida que se aumenta el número de contraseñas generadas aumenta el número de contraseñas repetidas. Sin embargo, la tendencia decreciente del porcentaje de contraseñas únicas es mucho menor que la del incremento del número de contraseñas sobre las que se evalúa. Es decir, que pasando de 10.000.000 a 100.000.000 contraseñas generadas solo está decreciendo un 8% el porcentaje de contraseñas únicas, por lo que se puede seguir incrementando el número de contraseñas generadas manteniendo la eficiencia de generación.

La quinta conclusión, es que se ha conseguido entrenar el modelo alcanzando un *accuracy* de un 7,57% en el conjunto de contraseñas filtradas *RockYou*. Esté resultado es mejor que lo conseguido por el modelo pre-entrenado subido al repositorio, de 5,49%. El mayor porcentaje de *accuracy* se consigue para el mayor número de iteraciones de entrenamiento, 250.000. Por otro lado, al evaluar el modelo con la base de datos de *Pwned* [60] (la mayor de acceso público en la actualidad) se han conseguido *accuracy* de hasta el 23%. El hecho de que el modelo sea capaz de generar un 23% de contraseñas de dataste completamente distintos una de las principales conclusiones de los resultados, ya que se confirma que el modelo es capaz de generalizar y conseguir aumentar un dataste de contraseñas con una eficacia de un 23%.

La última conclusión y relaciona con la anterior, es que en *Pwned* el mayor *accuracy* se ha dado no para el modelo entrenado con 250.000, si no para 50.000 iteraciones. Tras estudiar distintas alternativas se ha concluido que el modelo no generaliza la generación de contraseñas si se excede el número de iteraciones de entrenamiento óptimas. Por lo tanto, aunque según se aumenta el número de iteraciones se consigue la mayor *accuracy* en el conjunto de validación de RockYou, no es óptimo, es preferible entrenarlo hasta 50.000 iteraciones si se quiere generalizar lo máximo posible.

VI. Contribuciones y Trabajo Futuro

La principal contribución de este trabajo es el análisis de un total de 12 repositorios de código relacionados con modelos basados en *PassGAN*. Esta investigación ha permitido obtener una visión más amplia y completa del estado del arte de adivinadores de contraseñas. Se ha podido comprobar que existe una gran cantidad de artículos que proponen mejoras sobre *PassGAN*, sin embargo, ninguno alcanza el *accuracy* que consigue *GNPassGAN*.

Por otro lado, no solo se ha contribuido con el estudio de los últimos modelos publicados, si no que se ha hecho un análisis en detalle del que se ha considerado como el que preséntame mejores resultados, *GNPassGAN*. Fruto del análisis se presentan los resultados de *accuracy* sobre la base de datos de contraseñas pública más grande. La consulta a *Pwned*, la base de datos, ha permitido obtener resultados más representativos y relevantes en cuanto a la generación y evaluación de contraseñas de los publicados en los estudios hasta la fecha.

En el afán de analizar de manera exhaustiva la capacidad de adivinación de *GNPassGAN*, no solo se ha orientado el proyecto al estudio del *accuracy*, si no que se ha contribuido con el análisis de múltiples aspectos de *GNPassGAN* que no se contemplan en los artículos publicados: la evolución de las perdidas en el entrenamiento, la proporción de contraseñas únicas, el ruido de generación de contraseñas distintas en función del número de iteraciones de entrenamiento o la distribución de las contraseñas generadas en cuanto a caracteres y *n-grams* y longitud.

En cuanto a futuros trabajos, como consecuencia de las conclusión extraída de que *GNPassGAN* es el principal método actual de adivinación de contraseñas, se propone dos nuevas líneas de trabajo:

La primera es el estudio de la capacidad del discrimínate para detectar contraseñas seguras generadas por el modelo. Esta alternativa permitiría garantizar que las contraseñas introducidas por los usuarios cumplan con los estándares de seguridad de tal forma que un atacante que haga uso de un modelo *GNPassGAN* o similar no sea capaz de atacar al sistema con contraseñas generadas falsas. De esta forma se puede evaluar cada vez que un usuario introduzca una nueva contraseña en el sistema como de probable es que un atacante usando *GNPassGAN* la pueda adivinar.

La segunda línea es la implementación de una sistema de *honeyword* que permita detectar si se han producido filtraciones de contraseñas en el sistema. El objetivo sería integrar *GNPassGAN* dentro del sistema de gestión de las contraseñas, añadiendo a la base de datos contraseñas muy similares a las existentes pero que a su vez no hayan sido creadas por ningún usuario. De esta forma se consigue detectar si han intentado con algunas de las contraseñas ficticias si se ha producido un fuga de información.

VII. Referencias

[1] B. Hitaj, P. Gasti, G. Ateniese, and F. Perez-Cruz, "PassGAN: A Deep Learning Approach for Password Guessing." arXiv, 2017. doi: 10.48550/ARXIV.1709.00440.

[2] A. Adams and M. A. Sasse, "Users Are Not the Enemy," *Commun. ACM*, vol. 42, no. 12, pp. 40–46, Dec. 1999, doi: 10.1145/322796.322806.

[3] S. Gaw and E. W. Felten, "Password Management Strategies for Online Accounts," in *Proceedings of the Second Symposium on Usable Privacy and Security*, in SOUPS '06. New York, NY, USA: Association for Computing Machinery, 2006, pp. 44–55. doi: 10.1145/1143120.1143127.

[4] B. Ur *et al.*, "How Does Your Password Measure Up? The Effect of Strength Meters on Password Creation," in *21st USENIX Security Symposium (USENIX Security 12)*, Bellevue, WA: USENIX Association, Aug. 2012, pp. 65–80. [Online]. Available: https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/ur

[5] A. Das, J. Bonneau, M. C. Caesar, N. Borisov, and X. Wang, "The Tangled Web of Password Reuse," in *Network and Distributed System Security Symposium*, 2014.

[6] R. Shay *et al.*, "Encountering Stronger Password Requirements: User Attitudes and Behaviors," in *Proceedings of the Sixth Symposium on Usable Privacy and Security*, in SOUPS '10. New York, NY, USA: Association for Computing Machinery, 2010. doi: 10.1145/1837110.1837113.

[7] R. Wash, E. Rader, R. Berman, and Z. Wellmer, "Understanding Password Choices: How Frequently Entered Passwords Are Re-Used across Websites," in *Proceedings of the Twelfth USENIX Conference on Usable Privacy and Security*, in SOUPS '16. USA: USENIX Association, 2016, pp. 175–188.

[8] E. Stobert and R. Biddle, "The Password Life Cycle: User Behaviour in Managing Passwords," in *10th Symposium On Usable Privacy and Security (SOUPS 2014)*, Menlo Park, CA: USENIX Association, Jul. 2014, pp. 243–255. [Online]. Available: https://www.usenix.org/conference/soups2014/proceedings/presentation/stobert

[9] W. B. Polk Donna Dodson, W., "NIST SP 800-63 Electronic Authentication Guideline," *NationalInstitute of Standards and Technology*, 2004.

[10] P. G. Fenton Michael Garcia James, "NIST SP 800-63-3 Digital Identity Guidelines," *NationalInstitute of Standards and Technology*, 2017.

[11] P. G. Theofanos Elaine Newton ,. James Fenton ,. Ray Perlner ,. Andrew Regenscheid ,. William Burr, Justin Richer ,. Naomi Lefkovitz ,. Jamie Danker ,. Yee-Yin Choong ,. Kristen Greene ,. Mary, "NIST SP 800-63B Digital Identity Guidelines: Authentication and Lifecycle Management," *NationalInstitute of Standards and Technology*, 2020.

[12] S. Gupta, A. Singhal, and A. Kapoor, "A literature survey on social engineering attacks: Phishing attack," in *2016 International Conference on Computing, Communication and Automation (ICCCA)*, 2016, pp. 537–540. doi: 10.1109/CCAA.2016.7813778.

[13] A. L.-F. Han, D. F. Wong, and L. S. Chao, "Password cracking and countermeasures in computer security: A survey," *arXiv preprint arXiv:1411.7803*, 2014.

[14] P. Simmons, "Security through Amnesia: A Software-Based Solution to the Cold Boot Attack on Disk Encryption," in *Proceedings of the 27th Annual Computer Security Applications Conference*, in ACSAC '11. New York, NY, USA: Association for Computing Machinery, 2011, pp. 73–82. doi: 10.1145/2076732.2076743.

[15] T. Kakarla, A. Mairaj, and A. Y. Javaid, "A Real-World Password Cracking Demonstration Using Open Source Tools for Instructional Use," in *2018 IEEE International Conference on Electro/Information Technology (EIT)*, 2018, pp. 0387–0391. doi: 10.1109/EIT.2018.8500257.

[16] P. Tasevski, "Password Attacks and Generation Strategies," 2015, doi: 10.13140/RG.2.1.1247.8807.

[17] S. Nam, S. Jeon, and J. Moon, "A New Password Cracking Model with Generative Adversarial Networks," in *Information Security Applications*, I. You, Ed., Cham: Springer International Publishing, 2020, pp. 247–258.

[18] Sourceforge, "crunch - wordlist generator." Accessed: Dec. 04, 2022. [Online]. Available: https://sourceforge.net/projects/crunch-wordlist/

[19] P. Oechslin, "Making a Faster Cryptanalytic Time-Memory Trade-Off," in *Advances in Cryptology - CRYPTO 2003*, D. Boneh, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 617–630.

[20] N. Mentens, L. Batina, B. Preneel, and I. Verbauwhede, "Time-Memory Trade-Off Attack on FPGA Platforms: UNIX Password Cracking," in *Reconfigurable Computing: Architectures and Applications*, K. Bertels, J. M. P. Cardoso, and S. Vassiliadis, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 323–334.

[21] A. Narayanan and V. Shmatikov, "Fast Dictionary Attacks on Passwords Using Time-Space Tradeoff," in *Proceedings of the 12th ACM Conference on Computer and Communications Security*, in CCS '05. New York, NY, USA: Association for Computing Machinery, 2005, pp. 364–372. doi: 10.1145/1102120.1102168.

[22] C. E. Shannon, "A mathematical theory of communication," *The Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[23] M. Weir, S. Aggarwal, B. de Medeiros, and B. Glodek, "Password Cracking Using Probabilistic Context-Free Grammars," in *2009 30th IEEE Symposium on Security and Privacy*, 2009, pp. 391–405. doi: 10.1109/SP.2009.8.

[24] A. Ciaramella, P. D'Arco, A. De Santis, C. Galdi, and R. Tagliaferri, "Neural Network Techniques for Proactive Password Checking," *IEEE Transactions on Dependable and Secure Computing*, vol. 3, no. 4, pp. 327–339, 2006, doi: 10.1109/TDSC.2006.53.

[25] W. Melicher *et al.*, "Fast, Lean, and Accurate: Modeling Password Guessability Using Neural Networks," in *25th USENIX Security Symposium (USENIX Security 16)*, Austin, TX: USENIX Association, Aug. 2016, pp. 175–191. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/melicher

[26] L. Xu *et al.*, "Password Guessing Based on LSTM Recurrent Neural Networks," in *2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*, 2017, pp. 785–788. doi: 10.1109/CSE-EUC.2017.155.

[27] Y. Liu *et al.*, "GENPass: A General Deep Learning Model for Password Guessing with PCFG Rules and Adversarial Generation," in *2018 IEEE International Conference on Communications (ICC)*, 2018, pp. 1–6. doi: 10.1109/ICC.2018.8422243.

[28] "HashCat.net." Accessed: Dec. 04, 2022. [Online]. Available: https://hashcat.net/hashcat/

[29] "John the Ripper password cracker." Accessed: Dec. 04, 2022. [Online]. Available: https://www.openwall.com/john/

[30] I. Goodfellow *et al.*, "Generative Adversarial Nets," in *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, Eds., Curran

Associates, Inc., 2014. [Online]. Available: https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf

[31] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved Training of Wasserstein GANs." arXiv, 2017. doi: 10.48550/ARXIV.1704.00028.

[32] V. Garg and L. Ahuja, "Password Guessing Using Deep Learning," in *2019 2nd International Conference on Power Energy, Environment and Intelligent Control (PEEIC)*, 2019, pp. 38–40. doi: 10.1109/PEEIC47157.2019.8976614.

[33] S. Nepal, I. Kontomah, I. Oguntola, and D. Wang, "Adversarial Password Cracking," 2019.

[34] A. Chen, "Presenting New Dangers: A Deep Learning Approach to Password Cracking".

[35] F. Yu and M. V. Martin, "GNPassGAN: Improved Generative Adversarial Networks For Trawling Offline Password Guessing," in *2022 IEEE European Symposium on Security and Privacy Workshops (EuroS&amp$\mathsemicolon$PW)*, IEEE, Jun. 2022. doi: 10.1109/eurospw55150.2022.00009.

[36] D. Pasquini, A. Gangwal, G. Ateniese, M. Bernaschi, and M. Conti, "Improving Password Guessing via Representation Learning." arXiv, 2019. doi: 10.48550/ARXIV.1910.04232.

[37] I. Tolstikhin, O. Bousquet, S. Gelly, and B. Schoelkopf, "Wasserstein Auto-Encoders." arXiv, 2017. doi: 10.48550/ARXIV.1711.01558.

[38] K. Yang, X. Hu, Q. Zhang, J. Wei, and W. Liu, "VAEPass: A lightweight passwords guessing model based on variational auto-encoder," *Computers & Security*, vol. 114, p. 102587, 2022, doi: https://doi.org/10.1016/j.cose.2021.102587.

[39] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein GAN." arXiv, Dec. 06, 2017. Accessed: May 10, 2023. [Online]. Available: http://arxiv.org/abs/1701.07875

[40] T. Milne and A. Nachman, "Wasserstein GANs with Gradient Penalty Compute Congested Transport." arXiv, Jun. 30, 2022. Accessed: Jul. 01, 2023. [Online]. Available: http://arxiv.org/abs/2109.00528

[41] Y.-L. Wu, H.-H. Shuai, Z.-R. Tam, and H.-Y. Chiu, "Gradient Normalization for Generative Adversarial Networks." arXiv, Oct. 10, 2021. Accessed: May 30, 2023. [Online]. Available: http://arxiv.org/abs/2109.02235

[42] B. N. Vi, N. Ngoc Tran, and T. G. Vu The, "A GAN-based approach for password guessing," in *2021 RIVF International Conference on Computing and Communication Technologies (RIVF)*, 2021, pp. 1–5. doi: 10.1109/RIVF51545.2021.9642098.

[43] M. A. Fauzi, B. Yang, and E. Martiri, "PassGAN Based Honeywords System for Machine-Generated Passwords Database," in *2020 IEEE 6th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing, (HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS)*, 2020, pp. 214–220. doi: 10.1109/BigDataSecurity-HPSC-IDS49724.2020.00046.

[44] "have i been pwned?" [Online]. Available: https://haveibeenpwned.com/