Uniwersytet Gdański

Wydział Matematyki, Fizyki i Informatyki
Instytut Informatyki UG

Krzysztof Łozowski
Nr albumu: 255157

ZASTOSOWANIE UCZENIA MASZYNOWEGO W CYBERBEZPIECZEŃSTWIE

Machine Learning appliance in cybersecurity

PRACA MAGISTERSKA
na kierunku INFORMATYKA
specjalność: Bezpieczeństwo systemów informatycznych

Praca wykonana pod kierunkiem
dra Andrzeja Borzyszkowskiego

Gdańsk, lipiec 2019

Streszczenie

Celem pracy jest zbadanie możliwości wykorzystania uczenia maszynowego do przewidywania haseł. W tym celu użyte zostały generatywne sieci przeciwstawne (GAN) w modelu PassGAN, a otrzymane wyniki porównane zostały z istniejącymi, standardowymi technikami. Dla podejścia standardowego wykorzystane zostało narzędzie hashcat wraz ze swoim modelem ataku słownikowego.

Taki typ porównania pozwola na wyznaczenie efektywności naszego wygenerowanego modelu.

Abstract

The main purpose of the thesis is to present the possibilities of machine learning usage to predict passwords. In order to achieve that we use generative adversarial networks (GAN), in particural its PassGAN implementation. The obtained results shall be compared to the conventional techniques. To obtain standard result we use the hashcat tool, in particular the dictionary attack module.

This approach allows us to determine the effectiveness of our generated model.

Spis treści

I

Podstawowe pojęcia

1 Hasła

Definicja 1.1 (Hasło)
Hasłem nazywamy ciąg znaków używany w celu autoryzacji użytkownika systemu komputerowego. [1]

W celu uzyskania dostępu do konta użytkownik musi podać prawidłową nazwę oraz hasło. Nazwa użytkownika jest publiczna, natomiast informacje o haśle powinny być tajne. Oznacza to, że użytkownik nie powinien podawać informacji o haśle innym osobom, a administratorzy systemu powinni przechowywać te informacje w sposób bezpieczny - zaszyfrowany.

1.1 Właściwości haseł

Spoglądając na strukturę hasła można wyszczególnić następujące ich własności. Nie są to oczywiście wszystkie właściwości, lecz podana lista jest w pełni wystarczająca do stworzenia podstawowego opisu hasła.

- długość - liczba znaków, z której składa się hasło;
- wykorzystane zbiory znaków - podstawowy podział wygląda następująco [2]
  - małe litery - [a-z]
  - wielkie litery - [A-Z]
  - cyfry - [0-9]
  - znaki specjalne - pozostałe znaki, jakie mogą być wykorzystane przy tworzeniu hasła, zasady te mogą różnić się w zależności od miejsca, w którym podawane jest hasło.
- kolejność - struktura hasła, w której wyszczególniony jest typ pierwszego zbioru znaków w haśle

Przykład 1.2
Podane poniżej hasła posiadają następujące właściwości

- haslo - długość: 5, zbiór znaków: loweralpha, kolejność: allstring
- dtu123 - długość: 6, zbiór znaków: loweralphanum, kolejność: stringdigit
- 321HasLo - długość: 6, zbiór znaków: mixedalphanum, kolejność: digitstring

W ogólności, hasło uważane jest za bardziej bezpieczne, jeśli jest odpowiedniej długości, składa się z mieszanki każdego ze zbioru znaków i nie zawiera popularnie używanych słów. [2]

1.2 Hash

Mówiąc o tym terminie, mamy zazwyczaj na myśli jednostronną funkcję kryptograficzną.

Definicja 1.3 (Kryptograficzna funkcja hashująca)
Dla danego wejścia o dowolnej długości, kryptograficzna funkcja hashująca kondensuje je do wyniku o stałej długości. [3]

Dobrze rozpowszechnionymi przykładami takich funkcji są algorytmy: MD5, SHA1, SHA256, SHA3. Niektóre z nich (MD5, SHA1) nie oferują już dostatecznego poziomu zabezpieczenia i nie powinno się ich już używać.

Przykład 1.4
Wynikiem działania algorytmu hashującego MD5, dla zapytania *password*, będzie zawsze $5f4dcc3b5aa765d61d8327deb882cf99$.

Z punktu widzenia bezpieczeństwa, funkcje hashujące posiadają kilka fascynujących właściwości.

- Z definicji funkcja hashująca jest jednostronna. Oznacza to, że nie ma łatwego sposobu na wyznaczenie oryginalnego wejścia, gdy do dyspozycji mamy jedynie wynik działania takiej funkcji. Jako, że wynik działania funkcji jest zawsze tej samej długości, argument na wejściu może być zarówno prostym tekstem jak i plikiem binarnym o dowolnie dużej wielkości.
- Inną z własności tych funkcji jest efekt lawiny (ang. avalanche effect). Efekt ten oznacza, że każdy wynik działania funkcji, nawet dla najmniejszej zmiany wejścia, znacząco się od siebie różni. [4]

Przykład 1.5 (Avalanche effect)
Wynikami działania algorytmu hashującego MD5 dla poniższych zapytań są następujące:

- The quick brown fox - A2004F37730B9445670A738FA0FC9EE5
- The quick br0wn fox - 617E407AE569EAA40642A75BCF152617
- The quick brewn fox - E3FFF6D33AD19A13E47A3D94ADEAEDC2

Powyższe własności są bardzo użyteczne w kontekście procedur bezpiecznego przechowywania haseł.

1.2.1 Rainbow Tables

Definicja 1.6 (Rainbow Table)
Tablica tęczowa (ang. Ranbow Table) zawiera wyniki działania funkcji hashujących. Przechowywanie wyników ma na celu zanegowanie własności jednostronności wspomnianych funkcji. [5]

Czas wygenerowania takich tablic zależy od algorytmu, dla którego chcemy wygenerować takowe tablice, jak również od długości hasła.

	Speed	Length 4	Length 5	Length 6	Length 7	Length 8	Length 9	Length 10	Length 11	Length 12	Length 13
NTLM	31.82 TH/s	Instant	Instant	Instant	Instant	3 mins 29 secs	5 hrs 30 mins	3 wks 0 day	5 yrs 7 mos	538 yrs 1 mo	51.2 mil
MD5	17.77 TH/s	Instant	Instant	Instant	Instant	6 mins 14 secs	9 hrs 50 mins	1 mo 1 wk	10 yrs 1 mo	963 yrs 4 mos	91.6 mil
NetNTLMv1 / NetNTLMv1+ESS	16.82 TH/s	Instant	Instant	Instant	Instant	6 mins 35 secs	10 hrs 24 mins	1 mo 1 wk	10 yrs 8 mos	1 mil	96.8 mil
LM	15.81 TH/s	Instant	Instant	Instant	Instant						
SHA1	5.89 TH/s	Instant	Instant	Instant	Instant	18 mins 47 secs	1 day 5 hrs	3 mos 3 wks	30 yrs 7 mos	2.9 mil	276.3 mil
SHA2-256	2.42 TH/s	Instant	Instant	Instant	Instant	45 mins 39 secs	3 days 0 hr	9 mos 1 wk	74 yrs 4 mos	7.1 mil	671.9 mil
NetNTLMv2	1.22 TH/s	Instant	Instant	Instant	Instant	1 hr 30 mins	5 days 23 hrs	1 yr 6 mos	147 yrs 10 mos	14.1 mil	1335.5 mil
SHA2-512	801.9 GH/s	Instant	Instant	Instant	2 hrs 17 mins	1 wk 2 days	2 yrs 4 mos	224 yrs 9 mos	21.4 mil	2029.7 mil	
descrypt, DES (Unix), Traditional DES	647.59 GH/s	Instant	Instant	Instant	1 min 48 secs	2 hrs 50 mins	1 wk 4 days	2 yrs 11 mos	278 yrs 3 mos	26.5 mil	2513.3 mil
Kerberos 5, etype 23, TGS-REP	206.97 GH/s	Instant	Instant	Instant	5 mins 38 secs	8 hrs 54 mins	1 mo 0 wk	9 yrs 2 mos	870 yrs 10 mos	82.8 mil	7864 mil
Kerberos 5, etype 23, AS-REQ Pre-Auth	206.78 GH/s	Instant	Instant	Instant	5 mins 38 secs	8 hrs 54 mins	1 mo 0 wk	9 yrs 2 mos	871 yrs 8 mos	82.9 mil	7871.2 mil
md5crypt, MD5 (Unix), Cisco-IOS 1 (MD5)	7.61 GH/s	Instant	Instant	1 min 37 secs	2 hrs 33 mins	1 wk 3 days	2 yrs 7 mos	249 yrs 5 mos	23.7 mil	2252.6 mil	213995.1 mil
LastPass + LastPass sniffed	1.78 GH/s	Instant	Instant	6 mins 52 secs	10 hrs 52 mins	1 mo 1 wk	11 yrs 2 mos	1.1 mil	101.1 mil	9600.8 mil	912079.6 mil
macOS v10.8+ (PBKDF2-SHA512)	335.09 MH/s	Instant	Instant	36 mins 34 secs	2 days 9 hrs	7 mos 2 wks	59 yrs 7 mos	5.7 mil	538.2 mil	51127.7 mil	4857134 mil
WPA-EAPOL-PBKDF2	277.23 MH/s					9 mos 0 wk	72 yrs 0 mo	6.8 mil	650.5 mil	61799.3 mil	5870931.8 mil
TrueCrypt RIPEMD160 + XTS 512 bit	211.78 MH/s	Instant	Instant	57 mins 52 secs	3 days 19 hrs	11 mos 3 wks	94 yrs 3 mos	9 mil	851.6 mil	80839.5 mil	7685455.6 mil
7-Zip	181.51 MH/s	Instant	Instant	1 hr 7 mins	4 days 10 hrs	1 yr 1 mo	110 yrs 0 mo	10.5 mil	993.6 mil	94389.2 mil	8966975.1 mil
sha512crypt 6, SHA512 (Unix)	119.46 MH/s	Instant	1 min 5 secs	1 hr 42 mins	6 days 18 hrs	1 yr 9 mos	167 yrs 2 mos	15.9 mil	1509.7 mil	143419.6 mil	13624861.4 mil
DPAPI masterkey file v1	47.23 MH/s	Instant	2 mins 44 secs	4 hrs 19 mins	2 wks 3 days	4 yrs 5 mos	422 yrs 10 mos	40.2 mil	3818.1 mil	362723.1 mil	34458896.1 mil
RAR5	28.15 MH/s	Instant	4 mins 35 secs	7 hrs 15 mins	4 wks 0 day	7 yrs 5 mos	709 yrs 7 mos	67.4 mil	6407.6 mil	608720.6 mil	57828453.9 mil
DPAPI masterkey file v2	27.82 MH/s	Instant	4 mins 39 secs	7 hrs 20 mins	4 wks 1 day	7 yrs 6 mos	717 yrs 10 mos	68.2 mil	6482.1 mil	615797.6 mil	58500769.5 mil
RAR3-hp	20.84 MH/s	Instant	6 mins 12 secs	9 hrs 47 mins	1 mo 1 wk	10 yrs 1 mo	958 yrs 2 mos	91.1 mil	8652.3 mil	821972.3 mil	78087367.8 mil
KeePass 1 (AES/Twofish) and KeePass 2 (AES)	17.8 MH/s	Instant	7 mins 15 secs	11 hrs 28 mins	1 mo 2 wks	11 yrs 9 mos	1.1 mil	106.7 mil	10131.9 mil	962529.5 mil	91440305.8 mil
bcrypt $2*$, Blowfish (Unix)	11.37 MH/s	Instant	11 mins 21 secs	17 hrs 57 mins	2 mos 1 wk	18 yrs 5 mos	1.8 mil	167 mil	15860.3 mil	1506727.9 mil	143139150.9 mil
Bitcoin/Litecoin wallet.dat	3.55 MH/s	Instant	36 mins 18 secs	2 days 9 hrs	7 mos 2 wks	59 yrs 1 mo	5.6 mil	534.1 mil	50743.7 mil	4820655.6 mil	457962282.7 mil

Rysunek 1: Czasy potrzebne do wygenerowania tablic tęczowych przy użyciu Terahash Brutalis (448x Nvidia RTX 2080)

1.3 Salt

Celem soli jest ochrona systemowej bazy haseł przed atakami, które działanie opierają na wykorzystaniu tablic tęczowych. [6] Osiągane jest to poprzez dodawanie danych losowych do hasła podczas obliczania funkcji skrótu.

```
var salt = "ThisIsARandomString";
var pass = "MyPassword";
var hash = sha3(salt + password);
```

Listing 1: Przykładowy pseudokod implementujący solenie

Sól nie jest wartością ukrywaną, jednak każde wywołanie algorytmu hashującego powinno zawierać unikalną wartość soli.

Przykład 1.7
Przypisując do każdego hasła unikalną 32-znakową sól:

- uzyskujemy zasadniczo 40-znakowe hasło;
- eliminujemy również sytuacje, w których hasła powtarzają się - każdy hash będzie unikalny;
- eliminujemy możliwość użycia tablic tęczowych - ich wygenerowanie staje się czasowo nieopłacalne.

1.4 Pepper

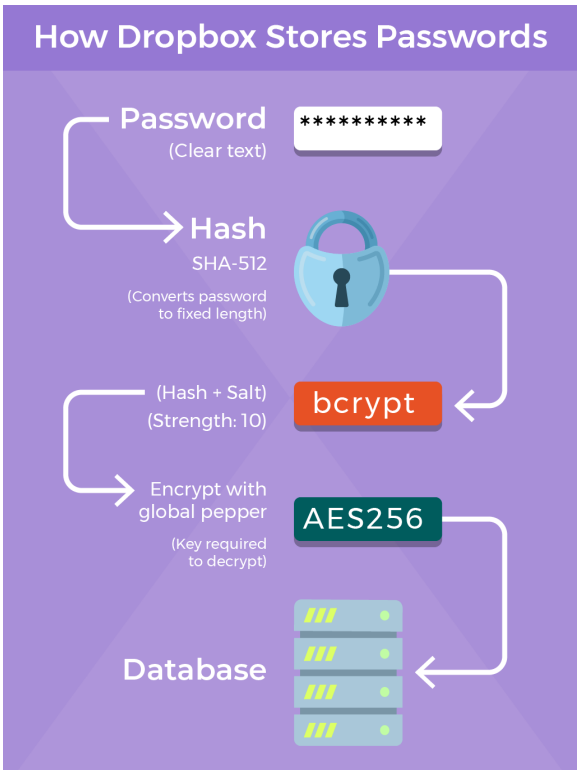
Co do zasady działania, pieprz nie różni się zbytnio od soli. Jednak wartość ta powinna pozostać ukryta (nie przechowywana w bazie razem z hashami). Dzięki takiemu podejściu złamanie hashy haseł przy ewentualnym wycieku bazy danych staje się zadaniem niewykonalnym (przy założeniu, że nie wyciekła również wartość pieprzu).

Zarówno sól, jak i pieprz mogą być użyte w procesie szyfrowania hasła. Przykład bezpiecznego podejścia zaprezentowany zostanie w kolejnym podrozdziale.

1.5 Bezpieczne przechowywanie haseł

W większości przypadków użycie funkcji hashującej (wraz z solą i/lub pieprzem) uchodzi za wystarczające w kontekście bezpiecznego przechowywania haseł. Jednakże, gdy hasła te bronią dostępu do danych wrażliwych użytkownika (dostęp do banku itp.), warto jest zaaplikować kilka dodatkowych warstw zabezpieczających.

Przykładem takiego podejścia jest firma Dropbox [7]. Analiza podejścia tej firmy jest następująca:



Rysunek 2: Schemat szyfrowania haseł

1. Główny mechanizm opiera się na algorytmie hashującym bcrypt. Użycie tego algorytmu wymusza również użycie soli.

2. Przed przekazaniem wejścia do algorytmu bcrypt, na hasła dodatkowo działa algorytm SHA512. Podejście to ma na celu rozwiązanie poniższych problemów

   (a) Niektóre z implementacji algorytmu bcrypt akceptują bardzo długie wejścia. Wynikiem działania algorytmu SHA512 ma zawsze długość 512 bitów.

   (b) Użycie algorytmu bcrypt bezpośrednio na haśle mogłoby mieć wpływ na jego długość (co osłabiałoby jego entropię).

3. Hash otrzymany w wyniku działania algorytmu bcrypt można już uważać za bezpieczny. Rezultat działania ataku typu brute-force, z perspektywy atakującego, nie przyniósłby satyfakcjonujących rezultatów.

4. Dodatkowo zastosowana została kolejna warstwa zabezpieczająca - użycie pieprzy na hashu wynikowym z bcrypt. Krok ten wykonywany jest przed zapisem wyników w bazie danych. Dane stają się bezwartościowe, jeśli atakujący nie posiada sekretnej wartości pieprzu (która przechowywana jest w miejscu bezpiecznym, poza bazą danych).

Uwaga
W czasie implementacji powyższych reguł przez Dropbox, algorytmy bcrypt oraz scrypt uważane były za najbezpieczniejsze.
W latach 2013 - 2015 odbył się jednak konkurs na ich następcę [8].
Zwycięzcą okazał się być algorytm Argon2 [9], który powinien być stosowany w trakcie implementacji nowoczesnych aplikacji.

2 Metody odgadywania haseł

Zgodnie z tym, co zostało pokazane w poprzednim rozdziale, dobrze zabezpieczona baza dadnych powinna być w miarę odporna na różne wektory ataków, które zostaną opisane poniżej. Jednak w sytuacji potencjalnego wycieku słabo zabezpieczonej bazy danych metody te dają dość dobre rezultaty.

2.1 Atak słownikowy

Atak słownikowy jest jedną z najpopularniejszych, jednocześnie cechującą się wysoką skutecznością, metod łamania haseł. Atak polega na wykorzystaniu listy wcześniej zdefiniowanych haseł wraz z wykorzystaniem informacji o częstotliwości ich występowania w przeszłości [10]. Najczęściej występujące hasła umieszczane są bliżej początku listy zwiększając prawdopodobieństwo złamania, w szczególności słabego, hasła. Najpopularniejszy zbiór tak uporządkowanych haseł nosi nazwę rockyou [11].

2.2 Atak słownikowy wraz ze zbiorem zasad

Swoistym ulepszeniem ataku słownikowego jest wzbogacenie go o dodatkowy zbiór zasad. Dla każdego hasła z zadanego zbioru generowane są dodatkowe "mutacje", czyli hasła bardzo zbliżone w swej składni do oryginału [10]. Popularnymi zasadami są:

- leet speak - żargon ten charakteryzuje częsta transliteracja słów z wykorzystaniem cyfr i innych znaków ASCII spoza alfabetu łacińskiego [12] (np. 1337 zamiast leet wymawiane jak el eat, to znaczy elite; pierwotnie miało formę 31337 – czyli po prostu elit), fonetyczny lub zbliżony zapis/odczyt liter i angielskich słów (np. „b” czytane jak nazwa litery zamiast czasownika be, „r” zamiast are), opuszczanie niewymawianych liter, umyślne błędy typograficzne (np. pr0n zamiast porn), nadużywanie „z” w miejsce „s” jako końcówki liczby mnogiej itp.;
- mixed case - mieszane stosowanie wielkich i małych liter w słowie jak również sporadyczne zastępowanie niektórych liter cyframi, których zapis przypomina zapis litery wyjściowej.

2.3 Brute Force

Atak polegający na sprawdzeniu każdej możliwej kombinacji słów, jakie mogą zostać utworzone przy wykorzystaniu każdego dostępnego znaku [13]. Jest to metoda wysoce nieefektywna czasowo, albowiem dla przykładu

Przykład 2.1 (Liczba haseł o długości 1-8)
Zakłóżmy, że mamy do dyspozycji następujące zbiory liter i cyfr (bez użycia znaków spacjalnych): [a-z], [A-Z], [0-9]. Liczba możliwych do wygenerowania haseł o długości z przedziału od 1 do 8 wynosi

$$72^1 + 72^2 + 72^3 + 72^4 + 72^5 + 72^6 + 72^7 + 72^8 \approx 7.234e^{14}$$

Liczba ta znacząco przewyższa te, które pojawiają się przy okazji wykorzystania innych metod. Atak typu bruteforce przestaje być efektywny, gdy długość hasła jest większa niż 8.

Przykład 2.2
Załóżmy, że maszyna oblicza funkcje skrótu haseł z prędkością $47GHz$. Zachodzą następujące czasy

1. sprawdzenie wszystkich haseł o długości 1-8 przy warunkach jak w poprzednim przykładzie zajęłoby ok. 4h 26min;
$$\frac{72^8}{47 \cdot 10^9} \approx 15366 \Rightarrow \frac{15366}{3600} \approx 4.26$$
2. sprawdzenie wszystkich haseł o długości 1-9 przy warunkach jak w poprzednim przykładzie zajęłoby ok. 12.8 dnia,
$$\frac{72^9}{47 \cdot 10^9} \approx 1106383 \Rightarrow \frac{1106383}{3600 \cdot 24} \approx 12.8$$

Czas ten, w większości przypadków, uważany jest za zbyt duży.

3 Uczenie maszynowe

Wraz ze stale zwiększającą się ilością danych elektronicznych, wzrasta również zapotrzebowanie na metody automatycznego ich analizowania. Celem uczenia maszynowego (ang. machine learning) jest stworzenie do takich rozwiązań, które potrafiłyby (bez ingerencji człowieka) wykrywać wzorce w danych, a następnie wykorzystać je w celu przewidywania nieznanych jeszcze rezultatów bądź zachowań.

Uczenie maszynowe jest formą sztucznej inteligencji, która pozwala na odnajdywanie wzorców w danych (nazywane również uczeniem się). Podejście to różni się zatem od klasycznego, polegającego na programowaniu wszystkiego wprost.

3.1 Podejście probabilistyczne

Zagadnienia uczenia maszynowego powiązane są z pojęciami wykorzystywanymi w statystyce. Zachodzą jednak subtelne różnice w używanej terminologii. Dziedziczone jest jednak podejście, iż najlepszym sposobem, aby maszyny były w stanie uczyć się z dostarczonych im danych, jest użycie narzędzi, które oferuje nam teoria prawdopodobieństwa. [14]

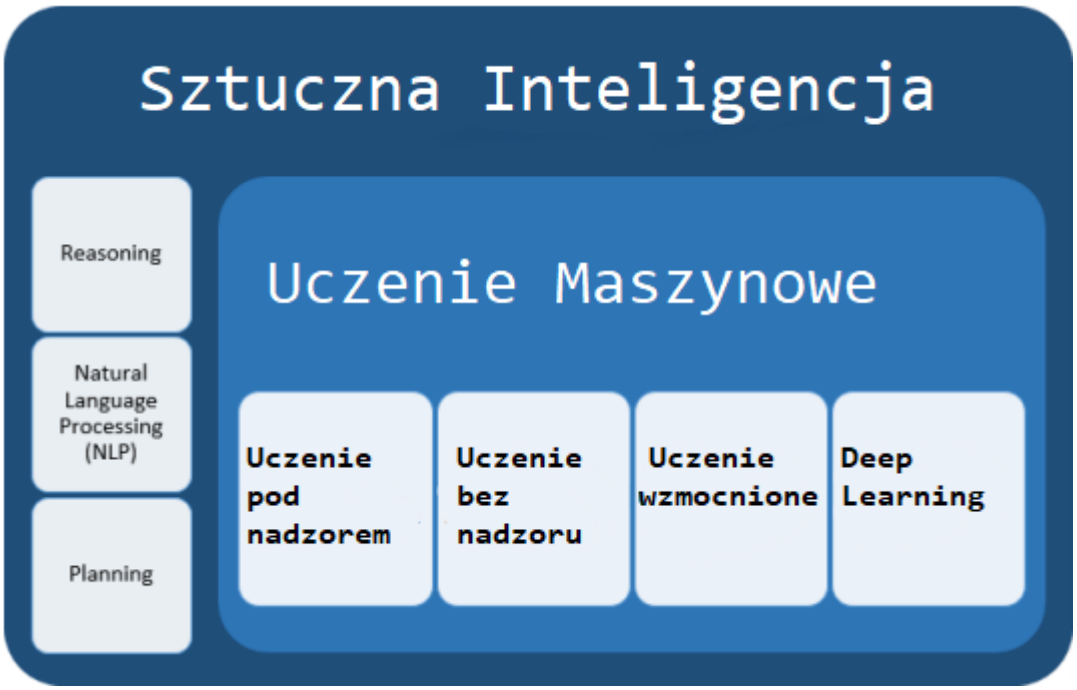
Teoria ta może być użyta tam, gdzie występuje niepewność. W uczeniu maszynowym niepewność pojawia się pod wieloma postaciami, np.

- jaki jest najlepszy wybór, który można podjąć mając do dyspozycji tylko skrawki danych;
- jaki model da najlepsze wyniki dla naszych danych;
- i wiele innych.

Większość ze spopularyzowanych algorytmów uczenia maszynowego stanowi implementację metod znanych ze statystyki [14]. Stanowią one więc przykład wykorzystanie modeli teoretycznych w praktyce.

3.2 Typy uczenia maszynowego

Uczenie maszynowe dzielone jest zazwyczaj na następujące kategorie:



Rysunek 3: Typy uczenia maszynowego [14]

Przed opisem typów uczenia maszynowego, warto również zaznajomić się z innymi konceptami z dziedziny sztucznej iteligencji [14]

1. Reasoning - narzędzie wspomagające wypełnianie luk w danych (korzystając z odnalezionych już wzorców).

2. NLP - Przetwarzanie Języka Naturalnego (ang. Natural Language Processing) - wytrenowanie komputera w taki sposób, aby żozumiałón ludzki język, zrówno w mowie, jak i piśmie.

3. Planning - umiejętność automatycznego i niezależnego podejmowania akcji w celu osiągnięcia założonego celu.

3.2.1 Uczenie pod nadzorem

Zwyczajowo proces uczenia pod nadzorem polega na odnalezieniu wzorców w danych przy założeniu, że na wejściu dany jest już zbiór możliwych klasyfikacji. Wynikiem działania algorytmu jest więc przyporządkowanie danych do jednego z danych zbiorów [14].

Klasyfikacja Celem klasyfikacji jest zmapowanie danych ze zbioru wejściowego X na zbiór rozwiązań Y (zbiór wyjściowy). Wynikiem tej operacji jest zbiór par typu wejście-wyjście $D = \{(x_i, y_i)\}_{i=1}^{N}$, gdzie D nazywamy zbiorem treningowym, a N oznacza liczbę danych treningowych [15].

W najprostszym podejściu, każda dana treningowa x_i jest D-wymiarowym wektorem, w którym liczby reprezentują pewne cechy, np. wysokość i wagę człowieka. Dane te nazywamy atrybutami. W ujęciu ogólnym x_i może być obiektem o wiele bardziej złożonym.

Podobnie, w teorii postać rozwiązania może być czymkolwiek. Większość metod zakłada jednak, że y_i jest zmienną ukategoryzowaną, tj. należącą do pewnego skończonego zbioru ($y_i \in \{1, \ldots, C\}$, np. płeć).
Jeśli $C = 2$, to klasyfikację nazywamy binarną.
Jeśli $C > 2$, to klasyfikację nazywamy multiklasową. Jeśli klasy nie wykluczają się wzajemnie (np. osoba może być zarówno wysoka jak i silna), to klasyfikację taką możemy utożsamiać z wieloma połączonymi klasyfikacjami binarnymi.

Przybliżenie funkcji Załóżmy, że rozwiązaniem naszego problemu jest $y = f(x)$, gdzie f jest pewną nieznaną funkcją. Celem jest znalezienie estymatora funkcji f, tj. $\hat{y} = \hat{f}(x)$. Im lepszy estymator, tym lepiej możemy przewidywać na temat danych, o których nie posiadamy jeszcze żadnej wiedzy [15].

Regresja Regresję można utożsamiać z klasyfikacją z tą tylko różnicą, że jest ona ciągła. Najpopularniejszym typem regresji jest regresja liniowa (wykres jest funkcją liniową) [14].

3.2.2 Uczenie bez nadzoru

Podejście do uczenia bez nadzoru zakłada, iż dane są tylko informacje o zbiorze wyjściowym. Nie mamy natomiast żadnych informacji o tym, jak wygląda zbiór wejściowy. Celem jest zatem odnalezienie "interesujących właściwości"danych. Nie posiadamy również żadnej wiedzy o tym, jak powinien wyglądać pożądany zbiór wyjściowy dla dowolnego zbioru wejściowego.

Algorytmy uczenia bez nadzoru segregują dane na grupy (klastry). Brak nadzoru bierze się właśnie z powodu tego, że nie jest znany początkowy kontekst. Procedura uczenia bez nadzoru może zostać wykorzystana jako pierwszy krok przy wykorzystaniu uczenia pod nadzorem [14].

3.2.3 Uczenie wzmocnione

Uczenie wzmocnione (ang. Reinforcement learning) jest modelem uczenia behawioralnego. Algorytm otrzymuje pewne informacje o danych, aby stworzyć możliwość otrzymania jak najlepszego wyniku. Różnica pomiędzy innymi typami polega na tym, że nie ma tutaj żadnego zbioru danych przykładowych. Procedura trenowania opiera się na metodzie prób i błędów [14]. Gdy otrzymana sekwencja okazuje się być prawidłowa, nazywamy ją "wzmocnioną", albowiem stanowi one najlepsze dopasowanie do rozwiązania problemu.

4 Deep Learning

Algorytmy należące do tej klasy problemów klasyfikacyjnych wykorzystują sięci neuronowe w celu poznania zbioru danych. Procedura odbywa się w sposób iteracyjny [14]. Sieci neuronowe zaprojektowane są w ten sposób, aby w jak najlepszym stopniu naśladować pracę luzkiego umysłu. Standardowa sieć neuronowa składa się z wielu prostych, połączonych ze sobą procesorów, nazywanych neuronami. Każdy z nich produkuje ciąg aktywacji o wartościach rzeczywistych [16].

4.1 Komponenty sieci neuronowej

4.1.1 Neuron

Neuron j otrzumujący wejście $p_j(t)$ TBD

5 Teoria Gier

Obszar zainteresowań teorii gier obejmuje problemy związane z decyzjami w układach, w których występuje dwóch lub więcej graczy. Każdy z uczestników takiej gry określa swoje preferencje, które określają sposób jego działania. Celem gry dla każdego z graczy jest zmaksymalizowanie osiąganego wyniku niezależnie od decyzji przeciwników. Innymi słowy, jest to teoria podejmowania optymalnych decyzji w warunkach, gdy nie wszystkie dane o uczestnikach są znane [17].

5.1 Czym jest gra

Gra składa się z następujących elementów [17]

1. Zbioru graczy
2. Zbioru reguł gry
3. Zbioru możliwych strategii dla każdego gracza
4. Zbioru możliwych wyników
5. Funkcji wypłaty oznaczającej wynik osiągany przez każdego z graczy w zależności od pojętej dezycji.

5.2 Gra o sumie zerowej

Jest to przypadek gry, w której zysk jednego gracza oznacza stratę drugiego. Suma wypłat wszystkich uczestników gry wynosi więc zero [17].

5.3 Algorytm min-max

Algorytm ten oznacza strategię, która polega na minimalizowaniu maksymalnych możliwych strat, często kosztem potencjalnego zysku. Algorytm służy do wybrania optymalnego ruchu w konkretnym momencie symulując możliwe decyzje przeciwników. Analiza ta może obejmować więcej niż jeden ruch do przodu. Aby osiągnąć optymalne wyniki przy użyciu algorytmu ważne jest, aby używana funkcja oceny dobrze opisywała faktyczny stan gry (optymalna strategia przeważnie nie jest znana) [17].

II

Model teoretyczny

6 GAN

Działanie generatywnych sieci kontradyktoryjnych (ang. Generative Adversarial Networks, GAN) można utożsamiać z grą o sumie zerowej. Mając do dyspozycji dwie sieci neuronowe Struktura GAN (ang. Generative Adversarial Networks; pol. Generatywne Sieci Kontradyktoryjne) służy do estymacji modeli generatywnych poprzez zastosowanie procesu kontradyktoryjnego, w którym jednocześnie trenowane są dwa modele:

- generator G, służący do wychwycenia rozkładu danych, oraz
- dyskryminiator D, którego zadaniem jest oszacowanie prawdopodobieństwa, iż otrzymana próbka pochodzi ze zbioru danych treningowych, a nie została wygenerowana przez G.

Trenowanie generatora G ma na celu zmaksymalizowanie pradopodobieństwa na to, że dyskryminator D popełni błąd. Model ten utożsamiać można z algorytmem pochodzącym z teorii gier o nazwie minimax (w wersji dla dwóch graczy)[17][18].

Przykład 6.1 (Zastosowanie GAN)
Wyobraźmy sobie sytuację, w której fałszerz próbuje zapłacić w sklepie podrobionymi pieniędzmi. Na początku nie ma on jednak żadnego doświadczenia, więc pierwsze sfabrykowane banknoty będą istotnie rozróżnialne od oryginałów, zatem kasjer nie będzie miał jakichkolwiek problemów z ich odróżnieniem.
Z czasem jednak fałszerz nabiera wprawy i zaczyna wytwarzać coraz to lepsze wersje fałszywych pieniędzy, przez co

- sprzedawcy zdarza się popełniać błędy w ocenie, które banknoty są prawdziwe, a po pewnym czasie
- sprzedawca nie jest w stanie odróżnić podróbki od oryginału.

W modelu GAN fałszerz pełni rolę generatora, który nieustannie próbuje ulepszyć wytwarzane przez siebie fałszywe banknoty. Rolą sprzedawcy jest natomiast wydanie prawidłowego osądu na temat tego, które z otrzymanych przez niego banknotów faktycznie są prawdziwe. Pełni on zatem rolę dyskryminatora.

Strukturę GAN najłatwiej zaimplementować przy założeniu, że oba znajdujące się w nim modele są sieciami neuronowymi. Aby poznać rozkład dystrybucji generatora p_g dla danych wejściowych x, definiujemy najpierw zmienną szumu $p_z(z)$, a następnie tworzymy mapowanie przestrzeni danych z oznaczeniem $G(z, \theta_g)$, gdzie G jest funkcją różniczkowalną, która jest reprezentowana przez sieć neuronową z paramterami θ_g.
Definiujemy również drugą sieć neuronową $D(x, \theta_d)$, która jako wynik zwraca pojedynczy skalar. $D(x)$ reprezentuje prawdopodobieństwo tego, iż x pochodzi z danych wejściowych zamiast z p_g.
Sieć D trenowana jest w celu maksymalizacji prawdopodobieństwa przypisania prawidłowej etykiety zarówno dla danych pochodzących ze zbioru treningowego,

jak i dla tych wytworzonych przez G. Równolegle trenujemy również sieć G w celu zminimalizowania $\log(1 - D(G(z)))$. Innymi słowy:

Definicja 6.2 (GAN [18])
Równolegle trenowane sieci neuronowe G oraz D postaci opisanej powyżej są uczestnikami dwuosobowej gry minimax z funkcją wartości $V(G, D)$

$$\min_G \max_D V(G, D) = \mathbb{E}_{x \sim p_{data}} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))]$$

W praktyce, zamiast trenować G w celu minimalizowania $(\log 1 - D(G(z)))$ trenujemy je w celu maksymalizowania $D(G(z))$. Wynikiem działania drugiej funkcji jest ten sam punkt stały dla G i D, lecz dostarcza ona o wiele bardziej dokładny gradient na początkowych etapach działania G (czyli na etapie, gdy G jest ńajsłabsza"). W konsekwencji nie dochodzi do sytuacji, w której D odrzuciłoby prawdopodobne przykłady wygenerowane przez G z tego tylko powodu, iż różniłyby się one mocno od danych treningowych.

7 PassGAN

Model PassGAN model opiera się o wykorzystanie IWGAN [19][20] (będący lekką modyfikacją modelu GAN) w celu wygenerowania haseł, które w jak największym stopniu przypominałyby hasła używane przez człowieka.

Idea polega na wytrenowaniu sieci neuronowych w taki sposób, aby dyskryminator D przyjmował hasła pochodzące zarówno ze zbioru treningowego, jak również hasła wygenerowane przez generator G. W oparciu o działanie dyskryminatora, sieć neuronowa G, będąca generatorem stara się dopasować parametry w taki sposób, aby odpowiadały one rozkładowi ze zbioru treningowego. Generator z tego modelu, po odpowiednim wytrenowaniu, służy również za generator haseł, które zostaną użyte na bazie danych [21].

Odpowiednie współczynniki z modelu PassGAN opisane zostaną w sekcji implementacyjnej.

III

Dane

8 Źródła danych

8.1 Trenowanie modelu

Zaimplementowany algorytm PassGAN został wytrenowany na zbiorze RockYou [11].

8.2 Sprawdzenie modelu

Skuteczność wytrenowanego modelu została sprawdzona na następujących źródłach danych:

1. Zbiór Badoo [22].
   Zbiór ten składa się z 85 380 860 hashy haseł różnej długości
2. Zbiór Fotolog [23].
   Zbiór ten składa się z 18 789 596 hashy haseł różnej długości
3. Dodatkowo, w przypadku klasycznego ataku słownikowego, za słownik posłużył zbiór RockYou [11]. Zbiór ten składa się z
   (a) 14 344 391 haseł różnej długości
   (b) 9 766 200 haseł długości < 10

IV

Implementacja

9 Łamanie haseł

9.1 Dostępne narzędzia

9.1.1 John the Ripper

John the Ripper (JtR) [24] jest oprogramowaniem, które ma na celu podjęcie próby złamania hasła poprzez obliczenie funkcji skrótu na zadanym ciągu znaków, a następnie porównanie, czy wygenerowany hash odpowiada temu na wejściu. Jeśli hashe te są identyczne, to identyczne powinny być również hasła.

9.1.2 Hashcat

Z powodu, że obliczanie hashy funkcji jest zadaniem relatywnie łatwym, systematycznie próbowano wykorzystać wielowątkowość oferowaną przez nowoczesne karty graficzne. Hashcat [25] jest właśnie takim narzędziem. Zasada jego działania nie różni się znacząco od tej wykorzystywanej przez JtR, jednak narzędzie to wykorzystuje moc obliczeniową karty graficznej, a nie tylko procesora. Obecnie karty graficzne (GPU) są w stanie wykonywać te operacje z prędkością 30 GHz (30 000 000 000 hashy na sekundę), używając do tego algorytmu MD5.

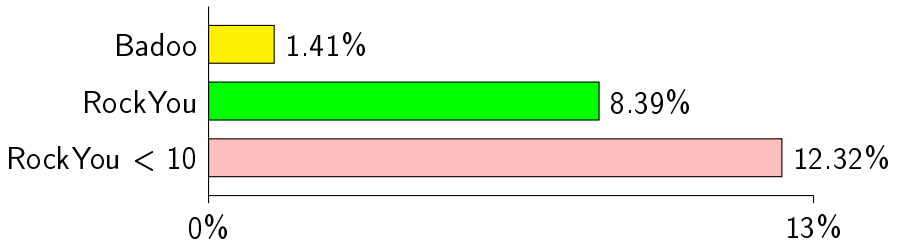
Ze względu na ten fakt, przy dalszych obliczeniach w tym dziale wykorzystane zostało narzędzie Hashcat.

9.2 Atak słownikowy

W przypadku tego typu ataku, za słownik posłużył zbiór RockYou. Obliczenia wykonano przy użyciu narzędzia Hashcat. Jest to jeden z naprostszych ataków oferowanych przez program.

9.2.1 Badoo

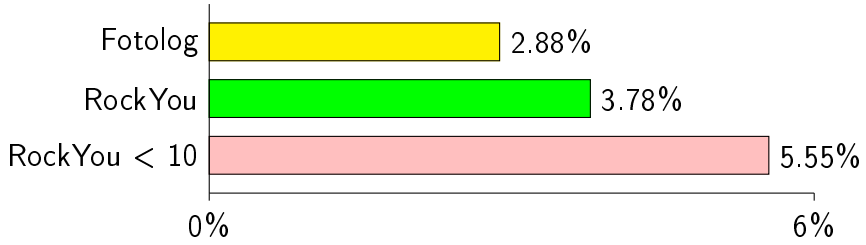
Udało się odgadnąć 1 203 838 haseł. Stanowi to 1.41% zbioru Badoo, oraz 8.39% zbioru RockYou (12.32% przy uwzględnieniu tylko haseł o długości < 10).



Rysunek 4: Zbiór Badoo - atak słownikowy - procent odgatniętych haseł

9.2.2 Fotolog

Udało się odgadnąć 542 364 haseł. Stanowi to 2.88% zbioru Fotolog, oraz 3.78% zbioru RockYou (5.55% przy uwzględnieniu tylko haseł o długości < 10).



Rysunek 5: Zbiór Fotolog - atak słownikowy - procent odgatniętych haseł

10 PassGAN

W następnych podrozdziałach znajduje się opis implementacji sieci PassGAN. Skrypty w języku Python zostały podzielone na dwie części: procedurę trenowania oraz generowania haseł.

10.1 Użyte narzędzia

- Python 3.7.1
- CUDA 10.1
- cuDNN 7.5
- tensorflow-gpu 1.13.1 - wersja wspierająca wykorzystanie kart graficznych do obliczeń

Obliczenia zostały przeprowadzone w oparciu o powyższe narzędzia na systemie Ubuntu 18.04 LTS o następujących podzespołach:

- Procesor - Intel i7-8700K
- Karta graficzna - Nvidia GeForce RTX 2060
- Pamięć RAM - 16GB DDR5

10.2 Reprezentacja zbioru danych

Zbiór danych RockYou użyty w celu wytrenowania modelu składał się z przefiltrowanych haseł o długości maksymalnie 10 [21].

Każda linia w pliku jest pojedyńczym rekordem.

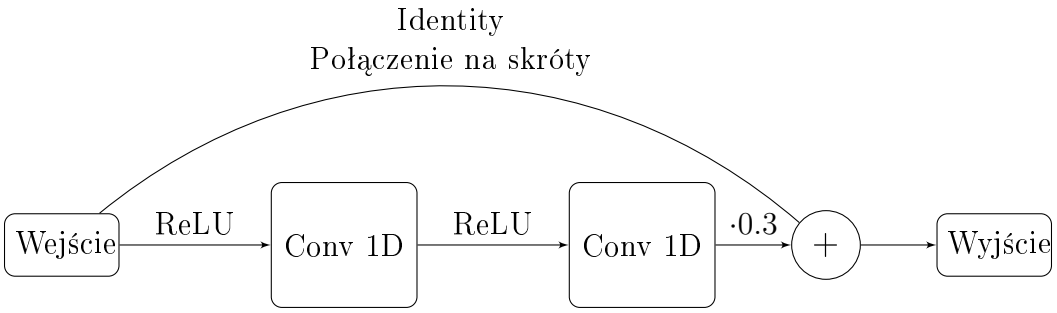
10.3 Pełna implementacja

Pełna implementacji algorytmu PassGAN dostępna jest na platformie GitHub pod adresem `https://github.com/lozovsky/mgr`

10.4 Budowa modelu

10.4.1 Blok szczątkowy

Algorytm swe działanie opiera o sieć IWGAN [20]. Sieć ta z kolei składa się z bloków szczątkowych (ang. Resiual Blocks), których schemat działania jest następujący:



Rysunek 6: Blok szczątkowy - komponent sieci PassGAN

Bloki szczątkowe w sieci PassGAN złożone są z dwóch jednowymiarowych warstw konwolucyjnych połączonych ze sobą za pomocą funkcji aktywacji ReLU - wzmocnionych jednostek liniowych (ang. rectified linear units). Wejście bloku jest funkcją tożsamościową $f(x) = x$, które następnie zwiększane jest o wartość $0.3 \cdot$ wynik działania warstw konwolucyjnych. Wynik tego działania stanowi wyjście działania bloku szczątkowego.

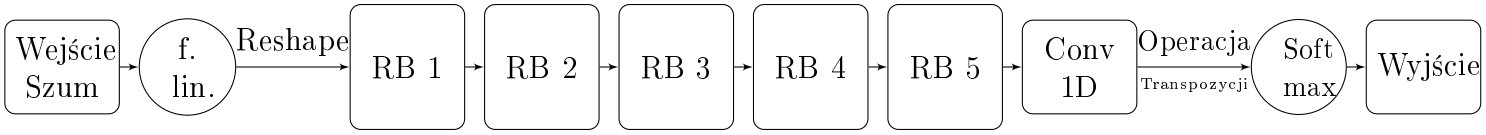
Implementacja Bloku szczątkowego przy użyciu biblioteki TensorFlow wygląda następująco:

```
def ResBlock(name, input, dim):
    return inputs +
        (0.3 * tflib.conv1d.Conv1D
                (name + '.2', dim, dim, 5,
                 tf.nn.relu(tflib.conv1d.Conv1D(
                 name + '.1', dim, dim, 5, tf.nn.relu(inputs)))))
```

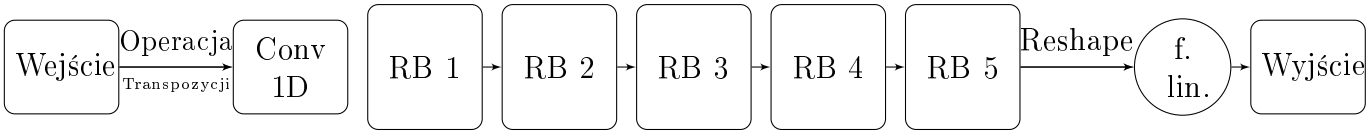
Rysunek 7: Blok szczątkowy - implementacja

W pracy używamy po pięć bloków szczątkowych zarówno dla generatora, jak i dyskryminatora.

10.4.2 Architektura modeli



Rysunek 8: Generator - schemat działania



Rysunek 9: Dyskryminator - schemat działania

Podczas procedury trenowania, dyskryminator D przetwarza hasła ze zbioru treningowego, jak również przykłady wygenerowane przez generator G.

W oparciu o wynik działania D, generator G dostosowuje swe parametry aby wytwarzać przykłady o rozkładzie jak najbardziej zbliżonym do tego ze zbioru treningowego.

Kiedy procedura trenowania jest zakończona, otrzymany w ten sposób model używany jest do generowania propozycji haseł.

Warto również wspomnieć o funkcji aktywacji softmax, wyrażonej wzorem

$$S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

Funkcja ta odpowiada za podanie predykcji modelu jako prawdopodobieństwo (wartości ze zbioru $[0, 1]$). Funkcję tą używamy dla śurowych"wartości z ostatniego neuronu naszej warstwy (ang. logits).

Implementacja tej funkcji jest następująca:

```
def softmax(logits, num_classes):
    return tf.reshape(
            tf.nn.softmax(
                tf.reshape(logits, [-1, num_classes])),
            tf.shape(logits))
```

Rysunek 10: Softmax - implementacja

Pełna implementacja generatora oraz dyskryminatora jest więc następująca:

```
def Generator(noise_size, batch_size, output_length, dimensionality, layer,
    output_dim):

    output = tf.reshape(tflib.linear.Linear(
                'Generator.Input', 128, output_length * dimensionality,
                tf.random_normal(
                shape = [batch_size, noise_size])),
                [-1, dimensionality, output_length])
    for i in range(1, layer):
        output = ResBlock('Generator.\%d' \% i, output, dimensionality)
    output = softmax(tf.transpose(
                tflib.conv1d.Conv1D(
                'Generator.Output', dimensionality, output_dim, 1, output),
                [0, 2, 1]), output_dim)
    return output
```

Rysunek 11: Generator - implementacja

```
def Discriminator(inputs, output_length, dimensionality, layer, input_dim):

    output = tflib.conv1d.Conv1D('Discriminator.Input',
                 input_dim, dimensionality, 1, tf.transpose(inputs, [0,2,1]))
    for i in range(1, layer):
        output = ResBlock('Discriminator.\%d' \% i, output, dimensionality)
    output = tflib.linear.Linear('Discriminator.Output',
                output_length * dimensionality, 1,
                tf.reshape(output, [-1, output_length * dimensionality]))
    return output
```

Rysunek 12: Dyskryminator - implementacja

10.4.3 Parametry

Poniższe parametry zostały użyte w celu wytrenowania sieci PassGAN, a wartości tych parametrów odpowiadają wartościom użytym w artykule [21]

1. batch-size (64)
   Liczba haseł ze zbioru treningowego rozprzestrzenianych do sieci GAN w każdej iteracji algorytmu

2. iterations (199 000)
   Liczba iteracji algorytmu. W każdej iteracji wywoływany jest generator oraz zadana liczna wywołań dyskryminatora.

3. discriminator-iterations (10)
   Liczba wywołań dyskryminatora w jednej iteracji algorytmu.

4. layer (5)
   Liczba warstw cząstkowych dla generatora i dyskryminatora (zgodnie z modelem)

5. dimensionality (128)
   Liczba wymiarów (wag) dla każdej warstwy konwolucyjnej.

6. gradient-penalty (10)
   Współczynnik poprawkowy aplikowany dla gradientu dyskryminatora

7. output-length (10)
   Maksymalna długość generowanych przykładów

8. noise-size (128)
   Rozmiar wektora szumu - ile losowych bitów dodawanych jest do wejścia G.

9. learning-rate (0.001)
   Tempo dopasowywania wag w modelu

10. beta1 (0.5)
    Współczynnik beta1 - opadające tempo obecnej średniej gradientu

11. beta2 (0.9)
    Współczynnik beta2 - opadające tempo obecnego kwaratu gradientu

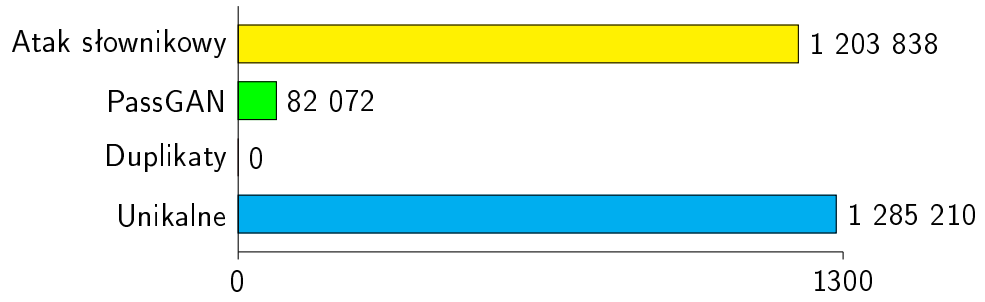
10.5 Generowanie haseł

Mając wytrenowany model wygenerowano plik zawierający 10 000 000 propozycji haseł, które posłużyły za słownik dla programu hashcat. Skuteczność tych propozycji podana została w kolejnym dziale.

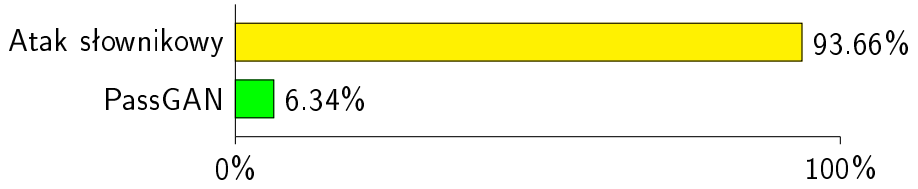
11 Analiza wyników

Plik uzyskany przy użyciu modelu PassGAN użyty został do ataku słownikowego dla tych samych baz hashy haseł jak w przypadku podejścia klasycznego.

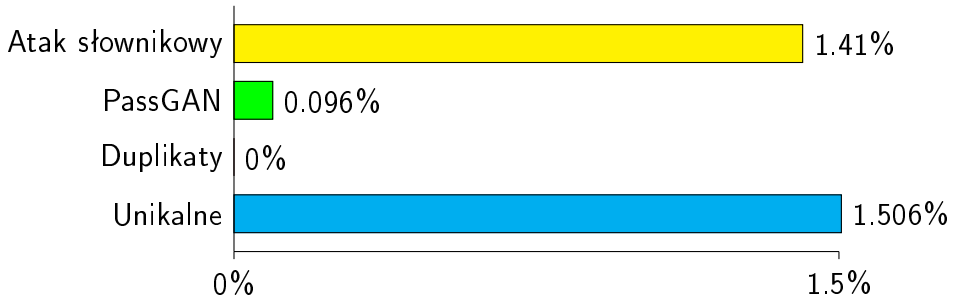
11.1 Zbiór Badoo



Rysunek 13: Zbiór Badoo - liczba odgadniętych haseł

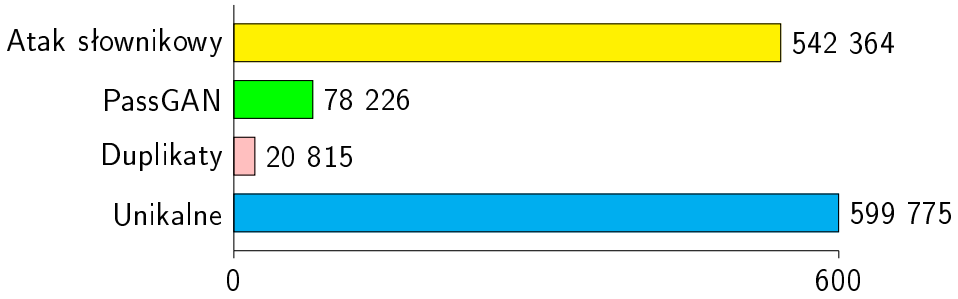


Rysunek 14: Zbiór Badoo - procent unikalnych haseł

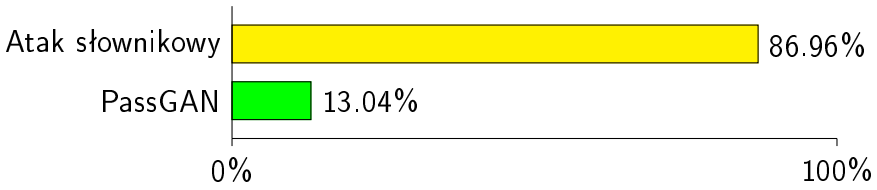


Rysunek 15: Zbiór Badoo - procent odgadniętych haseł

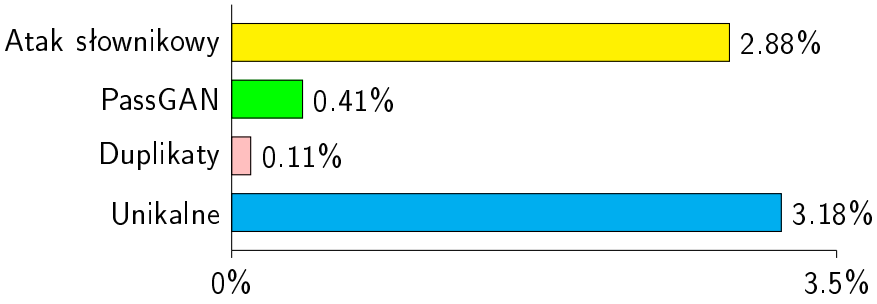
11.2 Zbiór fotolog



Rysunek 16: Zbiór Fotolog - liczba odgadniętych haseł



Rysunek 17: Zbiór Fotolog - procent unikalnych haseł



Rysunek 18: Zbiór Fotolog - procent odgadniętych haseł

Otrzymane wyniki na tle ogółu odgadniętych haseł nie wyglądają zbyt optymistycznie. Warto odnotować, że dla zbioru Badoo nie wystąpiły duplikaty haseł (czyli takie, które znajdowałyby się zarówno w zbiorze RockYou, jak i na liście propozycji algorytmu PassGAN).

W przypadku zbioru Fotolog udało się uzyskać ponad 13% nowych, unikalnych haseł. Wynik ten uważam za zadowalający.

Literatura

[1] Tech Terms. *Password Definition*
https://techterms.com/definition/password

[2] S. Granger. *The Simplest Security: A Guide To Better Password Practices.* Symantec, 2002
https://www.symantec.com/connect/articles/simplest-security-guide-better-password-practices

[3] S. Al-Kuwari, J. H. Davenport, R. J. Bradford. *Cryptographic Hash Functions: Recent Design Trends and Security Notions*
Department of Computer Science, University of Bath, UK, 2011.

[4] H. Feistel. *Cryptography and Computer Privacy.* Scientific American. 228 (5): 15–23, 1973.

[5] P. Oechslin. *Making a Faster Cryptanalytic Time-Memory Trade-Off.* LASEC, 2003.

[6] C. Wille. *Storing Passwords - done right!*
http://www.aspheute.com/english/20040105.asp

[7] D. Akhawe *How Dropbox securely stores your passwords*
https://blogs.dropbox.com/tech/2016/09/how-dropbox-securely-stores-your-passwords/

[8] Password Hashing
https://password-hashing.net/

[9] Argon2
https://github.com/p-h-c/phc-winner-argon2

[10] Kaspersky IT Encyclopedia. *Dictionary attack*
https://encyclopedia.kaspersky.com/glossary/dictionary-attack/

[11] RockYou
https://www.scrapmaker.com/view/dictionaries/rockyou.txt

[12] h2g2. *An Explanation of l33t Speak.* 2002.
https://h2g2.com/edited_entry/A787917

[13] Kaspersky IT Encyclopedia. *Brute-force*
https://encyclopedia.kaspersky.com/glossary/brute-force/

[14] J. Hurwitz, D. Kirsch *Machine Learning - IBM Limited Edition*
John Wiley & Sons, Inc., 2018

[15] A. Smola, S. V. N. Vishwanathan *Introduction to Machine Learning*
Cambridge University Press, 2008

[16] J. Schmidhuber. *Deep Learning in Neural Networks: An Overview*
Istituto Dalle Molle di Studi sull'Intelligenza Artificiale, University of Lugano & SUPSI, Switzerland, 2014
https://arxiv.org/pdf/1404.7828.pdf

[17] R. P. Kostecki. *Wprowadzenie do teorii gier*
https://www.fuw.edu.pl/ kostecki/teoria_gier.pdf

[18] I. J. Goofellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio *Generative Adversarial Nets*
Departement d'informatique et de recherche op'erationnelle, Universite de Montr'eal, Canada, 2014
https://arxiv.org/pdf/1406.2661.pdf

[19] M. Arjovsky, S. Chintala, L. Bottou *Wasserstein GAN*
Courant Institute of Mathematical Sciences, Facebook AI Research, 2017
https://arxiv.org/pdf/1701.07875.pdf

[20] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, A. Courville *Improved Training of Wasserstein GANs*
Montreal Institute for Learning Algorithms, Canada, 2017
https://arxiv.org/pdf/1704.00028.pdf

[21] B. Hitaj, G. Ateniese, P. Gasti, F. Perez-Cruz *PassGAN: A Deep Learning Approach for Password Guessing*
Stevens Institute of Technology, New York Institute of Technology, Swiss Data Science Center, 2019
https://arxiv.org/pdf/1709.00440.pdf

[22] Leak 'Badoo'
https://hashes.org/leaks.php?id=84

[23] Leak 'Fotolog'
https://hashes.org/leaks.php?id=1775

[24] John the Ripper password cracker
https://www.openwall.com/john/

[25] hashcat advanced password recovery
https://hashcat.net/hashcat/