Generative Deep Learning Techniques for Password Generation

David Biesner*[1,2,3], Kostadin Cvejoski*[1,3], Bogdan Georgiev[1,3],
Rafet Sifa[1,2,3], and Erik Krupicka[4]

Fraunhofer IAIS,
University of Bonn,
Competence Center for Machine Learning Rhine-Ruhr (ML2R),
Federal Criminal Police Office, Wiesbaden, Germany

Abstract. Password guessing approaches via deep learning have recently been investigated with significant breakthroughs in their ability to generate novel, realistic password candidates. In the present work we study a broad collection of deep learning and probabilistic based models in the light of password guessing: *attention-based deep neural networks*, *autoencoding mechanisms* and *generative adversarial networks*. We provide novel generative deep-learning models in terms of variational autoencoders exhibiting state-of-art sampling performance, yielding additional latent-space features such as interpolations and targeted sampling. Lastly, we perform a thorough empirical analysis in a unified controlled framework over well-known datasets (RockYou, LinkedIn, Youku, Zomato, Pwnd). Our results not only identify the most promising schemes driven by deep neural networks, but also illustrate the strengths of each approach in terms of generation variability and sample uniqueness.

1 Introduction and Motivation

Most authentication methods commonly used today rely on users setting custom passwords to access their accounts and devices. Password-based authentications are popular due to their ease of use, ease of implementation and the established familiarity of users and developers with the method. [4]

However studies show that users tend to set their individual passwords predictably, favoring short strings, names, birth dates and reusing passwords across sites. [7,9] Since chosen passwords exhibit certain patterns and structure, it begs the question whether it is possible to simulate these patterns and generate passwords that a human user realistically might have chosen.

Password guessing is an active field of study, until recently dominated by statistical analysis of password leaks and construction of corresponding generation algorithms (see Section 2). These methods rely on expert knowledge and analysis of various password leaks from multiple sources to generate rules and algorithms for efficient exploitation of learned patterns.

* Equal contribution

On the other hand, in recent years major advances in machine-driven text generation have been made, notably by novel deep-learning based architectures and efficient training strategies for large amounts of training text data. These methods are purely data driven, meaning they learn only from the structure of the input training text, without any external knowledge on the domain or structure of the data. Major advancements in the field have been fueled by the development in several central directions such as:

1. **The *attention* mechanisms**. Considering a token (word, letter, sentence) within a textual environment, the idea is to develop a flexible notion of *context* that connects the given token with other pieces of the textual environment. Intuitively, this allows the learning model to better grasp the textual structure (e.g. grammar, semantic meaning, word structure, etc), thus leading to eventual improvements in terms of text classification/generation/interpretability. Among others, well-known attention-based examples are given by BERT, ELMO, GPT and various further types of Transformers.
2. **Model architectures and representation capabilities**. Remarkable progress has been made in designing more flexible deep learning structures (CNN-based ResNets and Wide-Resnets, recurrent NNs, adversarial models, etc). The success of these deep neural networks can to a large extent be attributed to their *representation capability*, i.e. they create an appropriate transformation of the data (e.g. compression or a sort of "semantic meaning" extraction) that renders the data easier to handle and solve a given problem. In this regard a central class of deep learning models is given by the so-called ***autoencoders*** whose goal is not only to create a meaningful and useful data representation/transformation (Encoding) but also to be able to go back and reconstruct the initial data from the representation (Decoding). An upshot is that one could *generate new data* by sampling points in the representation space and then decoding back.
3. **Advanced training procedures**. The above tools would not be as efficient, had not it been for the corresponding methods to select (train) the parameters and weights of the neural networks. Among others these include appropriate momentum and annealing-driven stochastic gradient descents, Wasserstein regularization and variational approaches.

In this paper we will continue the exploration of data driven deep-learning text generation methods for the task of password-guessing. While some applications to password guessing already show promising results, most frameworks still can not reach or surpass state-of-the-art password generation algorithms. Ideally, one would attempt to design more efficient password-guessing models aided by neural networks and cutting-edge practices. Our findings and contributions can be summarized as follows:

1. We provide extensive unified analysis of previous as well as novel password guessing models based on deep learning and probabilistic techniques.
2. The collection of architectures based on deep learning exhibits varying performance, with the top-performing models being able to reach sophisticated password generation algorithms in the password recovery task.

3. We show that attention-driven text generation methods (Transformers) can be applied to password guessing with little additional adjustments. We additionally analyse the effect of model pre-training on general language data for the password generation task against training on pure password data.
4. Our novel variational autoencoder (VAE) approach allows more flexible latent representations and outperforms previous autoencoding methods based on Wasserstein training [22]. Moreover, the VAE's performance can be compared to an attention-driven one.
5. The VAE provides a state-of-art password matching performance as well as further sampling possibilities (interpolations, conditional and targeted sampling). However, the password latent space geometry is quite sensitive to training and regularization yielding promising grounds for future investigations in terms of conditional sampling.

2 Related work

Password generation has a long history outside of deep-learning architectures. There are tools available for purely rule-based approaches (Hashcat [1] and JohnTheRipper [3]), which generate password candidates either by brute-force or dictionary attacks, in which a dictionary of words or previously known passwords is augmented by a set of rules, either hand-written or machine generated [2].

Machine-learning based approaches to password guessing may come in their most simple form as regular n-gram Markov Models [23] or more sophisticated approaches like *probabilistic context free grammar* (PCFG) [30], which analyses likely structures in a password training set and applies various generation algorithms based on these observations.

Neural network based password generation has become an active field of study in the recent years. Ranging from relatively simple recurrent neural net (RNN) architectures [21] to recent seminal works applying state-of-the-art text generation methods to password generation: Generative adversarial networks (GANs) [14, 22], Wasserstein Autoencoders [22], and bidirectional RNNs trained with the aid of pre-trained Transformer models [20].

Our work extends this palette of deep learning architectures with the Variational Autoencoder [18] and Transformer-based language models [24]. We additionally offer an extensive, unified and controlled comparison between the both various deep-learning based methods and more established methods mentioned above. This analysis yields a stable benchmark for the introduction of novel models.

3 Models

3.1 GAN

A central idea of adversarial methods is the construction of generative models by game-theoretic means: a "generator" neural network produces data samples, whereas a "discriminator" neural network simultaneously attempts to discern between the real and artificially produced (by the generator) samples. The training of such a system consists in optimizing the performance of both the generator and discriminator (usually, via types of suitably chosen gradient descents and additional regularization). An important tool that smooths out gradients and makes the model more robust is the Wasserstein distance and corresponding cost function: it provides means to efficiently compute discrepancies between two given distributions $\mathbb{P}_0, \mathbb{P}_1$ as:

$$W(\mathbb{P}_0, \mathbb{P}_1) = \sup_{f \in \mathcal{L}_1} \mathbb{E}_{x \sim \mathbb{P}_0} \left[f(x) \right] - \mathbb{E}_{x \sim \mathbb{P}_1} \left[f(x) \right], \tag{1}$$

where $\mathcal{L}_1$ denotes the space of 1-Lipschitz functions. We refer to [12], [5], [13] for further background.

Concerning password guessing and generation our starting point is the well-known PassGAN model proposed in [14]. A further substantial breakthrough in this direction of GAN-based models was given in [22]. The original PassGAN defines a discriminator and generator in terms of residual networks [32] - these are assembled from the so-called residual blocks (e.g. a stack of convolutional neural networks followed by a batch-normalization [16]). A specific feature here is that the input partially bypasses the residual block (shortcutting) and is added to the output of the residual block - the aim is to diminish the effect of vanishing gradients and introduce a form of "memory" across the residual network.

3.2 Representation Learning with Deep Latent Variable Models

Many real world tasks as well as machine learning tasks can be solved very difficult or very easy depending on how the information is represented. Let us take for example the task of dividing the number 180 by 2. This task is straight forward using the Arabic numeral system. Now, we pose the same problem but we use the Roman numeral system to represent the numbers. In this case the task is dividing CLXXX by II. Using the Roman numeral system the task seems much more difficult for a modern educated person [11].

Representation learning is interesting because it provides one particular way to perform unsupervised learning. Unsupervised deep learning algorithms have a main training objective but also learn representations as side product. Naturally arises the question: *What makes one representation better than another?* In the context of machine learning a representation $\mathbf{z}$ is good if it makes the subsequent task easy to solve. One hypothesis out there is that the best representation is the one in which each of the features within the representation corresponds to the underlying factors or causes that generated the observed data. In ideal case each of the features or directions in the features space corresponds to different causes of the data. This kind of representation disentangles the underlying generative factors and they are called *disentangled* representation. There is a large body of deep representation learning research focused on obtaining disentangled representation.

Why representation learning is important in password guessing setting? There has been extensive research on text-password attacks and how people are choosing passwords. Human chosen passwords are not uniformly distributed in the space of passwords (all possible strings). The users tend to pick passwords that are easy to remember or have some personal meaning. Also, every web services has different policies about password construction (length, number of symbol, capital letters and etc.) Therefore, one can conclude that there are different latent factors in constructing a password. If we are able to learn these different factors, it will be possible to generate passwords that are from the same distribution as the human generated passwords.

Variational Auto Encoder (VAE) [18] is a framework for efficient optimization of *deep latent variable models* (DLVM). It comprises of two main components: i) *encoder* and ii) *decoder*. The encoder is stochastic function $\phi : \mathbf{X} \rightarrow \mathbf{Z}$ that maps the input space (passwords) $\mathbf{X}$ to the latent space $\mathbf{Z}$. The decoder is deterministic function that maps a code from the latent space to the input space $\theta : \mathbf{Z} \rightarrow \mathbf{X}$. The model is trained by maximizing the log likelihood

$$\mathcal{L}(\theta, \phi, \mathbf{x^{(i)}}) = -D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}^{(i)})||p_\theta(\mathbf{z})) + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x}^{(i)})}\left[\log p_\theta(\mathbf{x}^{(i)}|\mathbf{z})\right]. \tag{2}$$

The model learns to reconstruct the password $\mathbf{x}$ given to the input by first, mapping the password to a distribution of latent codes $p_\psi(\mathbf{z}|\mathbf{x})$ then we sample from the posterior distribution and we pass the latent code $\mathbf{z}$ to the decoder $p_\theta(\mathbf{x}^{(i)}|\mathbf{z})$. During training a strong prior $p(\mathbf{z})$ is imposed on the learned latent code distribution. Setting the prior can be informed by some previous knowledge that we have about the generative process. However, usually in the VAE framework the prior is set to be centered isotropic Gaussian distribution $p(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$. this is done so later we can easy sample from them and generate new passwords.

The latent space learned by the encoder imposes a geometric connections among latent points that have some semantic similarity in the data space. As a result *similar* points in the data space have latent representation that are close to each other. The notation of *similarity* depends on the modeled data, in the case of password generation it can be based on the structure of the password, the common substring, etc.

Training of VAEs with unmodified objective function (2) often can lead to converging to a undesirable local minimum [6], [26], [19]. This is the case because at the start of the training, the reconstruction term $\log p_\theta(\mathbf{x}|\mathbf{z})$ is weak, i.e., the latent code $\mathbf{z}$ does not contain any useful information about the point $\mathbf{x}$. This will lead to an optimization surface with a local minimum around $q(\mathbf{z}|\mathbf{x}) \approx p(\mathbf{z})$, which is difficult to escape. One possible solution ([6], [26]) to this problem is to use an optimization scheduler where the weight β for the $D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}))$ is annealed for 0 to 1.

Wasserstein Auto Encoder (WAE) [27] is a framework for building generative models using the optimal transport theory (OT) and by minimizing the optimal transport cost [29] between the data (unknown) distribution P_X and a latent variable model P_G. The optimal transport cost measures the distance between two probabilities by posing much weaker structure in comparison to other distance metrics, like for example f-divergence. This is important when we deal with data that is supported on low dimensional manifold in the input space. The WAE objective is defined as

$$D_{WAE}(P_X, P_G) := \inf_{Q(Z|X)\in\mathcal{Q}} \mathbb{E}_{P_X}\mathbb{E}_{Q(Z|X)}\left[c\left(X, G(Z)\right)\right] + \lambda\mathcal{D}_Z(Q_Z, P_Z), \quad (3)$$

where $\mathcal{Q}$ is any nonparametric set of probabilistic encoders, $\mathcal{D}_Z$ is an arbitrary divergence between Q_Z and P_Z, and $\lambda > 0$ is hyper-parameter.

3.3 Transformers

In recent years the transformer, originally applied to machine translation in [28], have become increasingly popular, with transformer-based architectures setting new benchmarks in text generation, machine translation and other NLP tasks.

Transformers rely almost solely on self-attention to process an input text, which considers all pairs of words in the sentence instead of the linear sequence of words. While RNNs may lose the memory of words in the beginning of a sentence rather quickly, transformers are able to capture long-term dependencies between words in a sentence and between sentences.

The self-attention mechanism evaluates attention for each word pair by multiplying their entries in the query, key and value matrices $Q, K \in \mathbb{R}^{n,d_k}$ and $V \in \mathbb{R}^{n,d_v}$ (d_k dimensionality of the query and key, d_v of the value vectors respectively) as

$$\mathbf{Attention}(Q, K, V) = \mathbf{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (4)$$

where the output of the softmax operation provides for each word a probability distribution over all other words in the sequence, which is then used to weight the word values to produce the attention output.

Popular transformer architectures include BERT [10], XLNet [31], Transformer XL [8], and GPT2 [24]. In our work we apply the GPT2 architecture to the password modeling task.

GPT2 is a transformer-based language model, trained on the causal language modeling (CLM) objective, meaning given an incomplete sentence it will try to predict the next upcoming word. A tokenized input sentence is therefore read by a transformer block which outputs a probability distribution p_θ over the vocabulary. For a corpus of tokens $\mathcal{U} = (u_1, \ldots, u_N)$ and a context window $k \in \mathbb{N}$ the loss is then given as

$$\mathcal{L}(\mathcal{U}) = \sum_{i=1}^{N} p_\theta(u_i | u_{i-1}, \ldots, u_{i-k}) \quad (5)$$

To generate text, given a text prompt (e.g. “Hello my ”) the model will start generating text that continues the sentence (“name is GPT2!”). We provide details on the model and training in Section 5.1.

4 Data

4.1 On public datasets

There are several datasets of passwords publicly available. These lists contain passwords that were at some point in time leaked to the public from on certain websites. Leaks contain in rare cases plaintext passwords (e.g. the RockYou leak), but more commonly only password hashes that are then recovered using password guessing methods. Password datasets contain either passwords from a single leak or are aggregated from several leaked sources.

The specific password datasets[1] we employ for training or evaluation are:

- 'rockyou'[2]: 13.0M passwords (116MB), leak of plaintext passwords from a single source,
- 'Have I Been Pwnd V1' ('pwnd')[3]: 319.8M passwords (3.2GB), compilation of multiple leaks of hashed passwords, almost all passwords are recovered and available as plaintext,
- 'linkedin'[4]: 60.1M passwords (619MB), leak of hashed passwords from a single source, most passwords are recovered and available as plaintext,
- Addtional leaks for evaluation: 'myspace' (53k), 'yahoo' (430k), 'youku' (48M), 'seclist' (969k), 'skullsecurity' (6.2M), 'zomato' (6.3M)[5]

Note that there is a certain bias inherent in password datasets. Passwords are (or should be) generally not stored as plaintext but as hashes. Password guessing methods however need plaintext passwords for training. Therefore all available data either belongs to websites which stored their passwords as plaintext (e.g. rockyou) or are recovered password hashes, therefore passwords that were already possible to reconstruct using available methods.

4.2 On segments – splitting passwords into likely subword tokens

Classically, text generation models are based on words or characters. Either a model chose at each step a new likely word from a large vocabulary (eg. $300k$) to generate, or it constructed words character by character, choosing from all available characters (for English text usually around 75 characters, a-z, A-Z, 0-9 and punctuation).

While the first approach is not suitable for password generation, since we only aim to generate one "word" (i.e. password) at a time and would need a vocabulary consisting of all possible passwords in the first place, the character-based approach is an immediate fit to our task. By pre-processing our dataset

[1] Provided link references accessed on 7.12.2020

[2] https://www.kaggle.com/wjburns/common-password-list-rockyoutxt

[3] https://hashes.org/leaks.php?id=70

[4] https://hashes.org/leaks.php?id=68

[5] https://weakpass.com/wordlist/22, https://weakpass.com/wordlist/44, https://hashes.org/leaks.php?id=508, https://hashes.org/leaks.php?id=587, https://weakpass.com/wordlist/50, https://weakpass.com/wordlist/671,

to only contain passwords consisting of the 75 characters we allow (depending on the dataset this removes 0-10% of all entries) we can tokenize each password into a sequence of characters and generate every possible password within the confines of our vocabulary.

However, one could ask whether it is really necessary for the model to take 8 generation steps to generate the sequence `password`, which occurs very regularly in the dataset. It might be beneficial to be able to generate certain common strings (`password`, `123`, `love`) in one step to decrease the length of common sequences since many sequence generation models struggle with remembering context over long distances [15].

Recent NLP models therefore employ a method called byte-pair-encoding [25], which searches a text corpus for the N most common sequences of characters, builds a vocabulary of size N and tokenizes new text using this vocabulary. The vocabulary size N is usually in the range of $30k$, far greater than character based models but smaller than word based models. Given that all relevant single characters are part of the vocabulary this method is then able to tokenize (and therefore encode and generate) every possible string.

In our work we employ this method as follows:

1. Choose training password dataset (rockyou)
2. Split each password into pure segments (ie. containing only characters or only numbers or only punctuation):
   `pass_word!!123` → `pass _ word !! 123`
3. Count all segments and sort by frequency
4. Choose most common $N = 30k$ segments, additionally add all single characters that are not yet in the vocabulary

We end up with a vocabulary of around $30k$ pure segments that are common substrings in the dataset. Using this vocabulary we are then able to tokenize every password in this or other datasets concisely. Further work might include using the exact byte-pair-encoding method described in [25] to extract a vocabulary of statistically significant non-pure substrings.

5 Results

5.1 Experimental Setup

GAN-based models In our setup, we essentially utilize the PassGAN as a standard benchmark - this is well motivated by the previous substantial studies of GAN-based models ([14], [22]). The generator/discriminator are defined as residual neural networks consisting of 6 standard residual blocks followed by a linear projection/softmax function, respectively. Each of the standard residual blocks consists of 2 convolutional layers (with kernel-size 3, stride 1, no dilation) followed by a batch-normalization layer. The generator's latent space (i.e. the input space) is set to 256 inspired by [14].

The PassGAN training was based on some state-of-art practices such as Wasserstein GAN and gradient clipping (cf. [5]), as well as gradient penalty

. . . Barack Hussein Obama II (born August 4, 1961) is an American politician and attorney who served as the 44th president of the United States from 2009 to 2017. A member of the Democratic Party, Obama was the first African-American president of the United States. He previously served as a U.S. senator from Illinois from 2005 to 2008 and an Illinois state senator from 1997 . . .

. . . mummy maryrose jumong imcute fresa energy bacardi yumyum underground shane1 olivia1 navarro brodie bribri anabel 12qwaszx sexy11 pppppp party mario1 juicy corazones smarty selina rebel ferreira bitch123 tomboy sweetlove skittles1 sirena sexy15 jhonny freeman elvira dieguito devin turtle1 sexbomb pink11 oswaldo morangos lavinia carlita adrian1 619619 woaini paintball love4u . . .

Fig. 1: (a) Training data for the pretrained GPT2 model is raw text from various inter sources. (b) Training data for finetuning of GPT2 is passwords from the 'rockyou' dataset contatenated into continuous text.

regularization (cf. [13]). We mostly used a batch-size of 256, a gradient-penalty-hyperparameter λ set to 10 and 10 discriminator iterations per generator step. The preferred gradient descent was based on ADAM [17] with an initial learning rate of $\sim 10^{-4}$ and beta (momentum) parameters $0.5, 0.9$; a fixed-interval annealing with an iteration step of $\sim 10^{6}$ was also used.

WAE/VAE-based models For both the encoder and the decoder we use a CNN with fixed kernel size of 3. The depth and the dilation of the convolution is gradually increase from [1, 2, 4], [1, 2, 4, 8], [1, 2, 4, 8, 16] to [1, 2, 4, 8, 16, 32]. We use cross-validation to pick the best hyper-parameters for the model. The number of channels for the CNN block is 512, the latent dimension of the model $\mathbf{z}$ is chosen from [64, 128, 256]. We use the ADAM [17] with learning rate $\sim 10^{-4}$ and momentum $\beta_1 = 0.5$ and $\beta_2 = 0.9$. The batch size is chosen to be 128 and we also use early stopping. Following [6], we use KL cost annealing strategy. In the case of the WAE we use *maximum mean discrepancy (MMD)* regularizer with RBF kernel.

GPT2-based models The original openly available GPT2 model is trained on a large corpus of internet text. Training data is provided simply as a sequence of raw, unlabeled text that is fed into the model. For training on passwords, one can therefore take a dataset of passwords and construct training data by concatenating shuffled passwords into continuous text (see also Figure (1)). In our report we train two GPT2-based models:

(i) We finetune (i.e. continue the training of) the pre-trained model with our password dataset. We hope that the original training gives the model some background on how language is generally structured as well as a vocabulary of common words. Finetuning on passwords should then give the model an understanding of the structures and the vocabulary of passwords and force it to generate passwords when prompted. We call this model GPT2-Finetuned or GPT2F;

(ii) The other model only uses the architecture of GPT2 and trains a randomly initialized model from scratch. A concern using a pre-trained and finetuned model is whether the model resorts back to generating regular English text when faced with certain prompts, this problem with a model trained from scratch will not appear since all the text it has ever known are passwords.

We train the model using an openly available implementation of GPT2,[6] with the default *gpt2*[7] model as pretrained base and default hyperparameters. The model is therefore a 12 layer, 12 attention-head model with latent dimension 768, maximum sequence length of 1024 and a vocabulary size of 50257. GPT2 applies byte-pair encoding trained on general english text to tokenize text (see Section 4.2).

5.2 Most common generated passwords

In Table (2) we compare the most common passwords generated by our models along with the most commonly generated passwords by the comparison methods. We observe in Table (2a) that each model and training dataset generates a unique set of password candidates. While the GPT2-based models trained on 'pwnd' commonly generate year numbers and short sequences of digits, the equivalent models trained on 'rockyou' produce strings that generally look more like real words that one might commonly find as a password or password substring. The VAE model focuses for both training datasets on names, with the 'rockyou' model generating lowercase names and the 'pwnd' model prefering all-caps strings. For each of these mentioned models we observe that the model trained on 'pwnd' in general seems to produce shorter strings.

However, the WAE reverses the behaviour observed so far. The 'rockyou' version generates an exceeding amount of similar looking numbers, while the pwnd-based model generates all lowercase strings that mostly appear like a concatenation of real words.

Finally we consider the PassGAN model implemented on the 'rockyou' training data. Here we observe a combination of simple number strings containing some variation of `12345` and character strings based on the substring `angel`. In general the most common passwords seem similar to each other, with less variance as observed for the other models.

Table (2c) describes the relative frequency of the most common generated password candidate for our models. We again see a large variance between the models and training data basis. While the PassGAN produces its most common string (`123456`) more every 10000 generations, the VAE trained on 'pwnd' produces its most common string (`MARIA`) only once every 2M generated passwords.

In general both VAE models generate their most common passwords comparatively rarely, while the GPT2-based models all appear in the middle of the field.

[6] https://huggingface.co/transformers/model_doc/gpt2.html

[7] https://huggingface.co/transformers/pretrained_models.html

Model	'rockyou'	'pwnd'
GPT2S	love	2010
	mrs.	love
	baby	1234
	sexy	2000
	girl	2009
	angel	2008
	1992	12345
	1993	2011
	1994	2007
	2007	1992
GPT2F	ilove	1234
	love	2010
	baby	love
	iluv	2000
	sexy	2009
	pink	2008
	memyself	12345
	caoimhe	2011
	cintaku	2007
	jess	1987

Model	'rockyou'	'pwnd'
VAE	leslie	MARIA
	yankee	hilton
	kirsty	SEXY
	jeremy	4678
	claudia	NATA
	gangsta	ALEX
	violet	JOSE
	andrei	BABY
	jennifer	MAMA
	natalie	ANGEL
WAE	08970899	pearlpolina
	08520899	larawilliam
	08970897	manuchandler
	0897	parispolina
	08970852	pearlleft
	08990899	poohilove
	0852	1609polina
	08520897	larapolina
	54040899	glass8159
	08520852	pearlwilliam

Model	rockyou
PassGAN	123456
	12345
	123456789
	1234567
	12345678
	angela
	angels
	angel1
	buster
	128456

(a) Most common generated passwords per model, separated by training dataset.

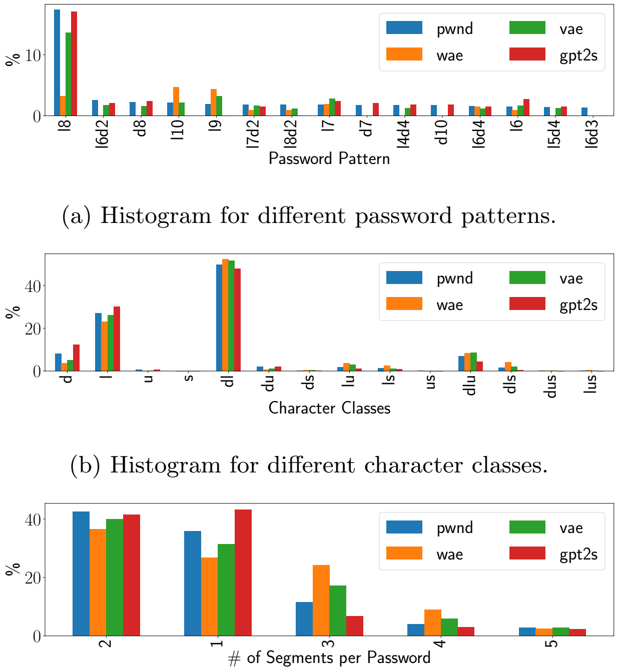
Hashcat – best64		Hashcat – gen2		PCFG		Markov Model		PRINCE	
Freq	Password	Freq	Password	Freq	Password	Freq	Password	Freq	Password
268 648	08	470032		8	zaq12345	2 002 838	12	6	22062206
224 033	ma	59840	1	8	zaq1234	1 697 575	08	4	shie2206
125 357	sa	48516	2	8	zaq121	1 589 461	ma	4	love2206
123 535	10	42536	0	8	aq12345	1 429 925	01	4	june2206
118 000	20	40760	3	8	123wed	1 011 725	10	4	july2206
117 682	01	33257	4	8	123was	974 664	00	4	contrasea
115 167	ch	32531	*	7	zaq123456	924 150	02	4	asdfghjkl
114 823	ja	31886	a	7	tre4567	922 786	09	4	22062536
109 098	ca	30455	5	7	tre456	855 534	99	4	22062534
108 894	09	29998	9	7	sw2121	820 617	an	4	22062533

(b) Most common generated passwords per comparison method trained on 'rockyou' with corresponding frequency for 10^9 generated passwords. Most common password generated by Hashcat gen2 is an empty string.

Model	Relative Frequency
PassGAN (rockyou)	9.9×10^{-3}
GPT2S (rockyou)	2.5×10^{-4}
WAE (rockyou)	1.8×10^{-4}
GPT2F (pwnd)	1.6×10^{-4}
GPT2S (pwnd)	1.4×10^{-4}
GPT2F (rockyou)	1.6×10^{-5}
VAE (rockyou)	1.2×10^{-5}
WAE (pwnd)	5.4×10^{-7}
VAE (pwnd)	4.6×10^{-7}

(c) Relative frequency of most common generated password per model. Sorted by frequency, lower models generate the most common password less often.

Table 2: Analysis of most common generated passwords of our models trained on 'rockyou' and pwnd, as well as the comparison methods evaluated in Table (9).



(a) Histogram for different password patterns.

(b) Histogram for different character classes.

(c) Histogram for the number of segments per password.

Fig. 3: We compare the properties of passwords generated using models trained on the 'pwnd' dataset against the properties of the passwords contained in the 'pwnd' dataset.

Additionally, to estimate if our trained model follow the distribution of the empirical dataset (the training set), we introduce three different statistics:

- *PCFG-like* [30] - password `password!23` is converted to **L8S1D2** where **L** is alpha character, **S** is special symbol character and **D** is a digit;
- *character type composition* - is a representation of the password with respect to character types contained in the password (ex. `Password!23` is represented as **lusd**) where **l** is a lower case letter, **u** is an upper case letter, **s** is a special symbol and **d** is digit;
- the number fragments per password (see Section 4.2).

In Figure (3) are presented the three statistics on passwords generated from models trained on the 'pwnd' dataset. The most common password structure for the ´pwnd' dataset is **l8** (see Figure (3a)) with almost 20% of the whole dataset. The GPT2S model successfully matches the number of passwords with structure **l8**, and the WAE models is performing poorly. For the rest of the password structures the GPT2S and the VAE model perform roughly the same and more or less they match the underlying empirical distribution. The second statistics

Model	**rockyou**	**linkedin**	**pwnd**	**myspace**	**yahoo**	**seclist**	**skullsec**	**youku**	**zomato**
VAE*	44.9%	**21.8%**	15.4%	62.5%	47.3%	57.4%	32.1%	13.8%	20.3%
WAE*	28.2%	12.9%	8.6%	45.7%	34.6%	43.8%	22.3%	10.1%	13.7%
PassGAN*	15.9%	6.8%	4.7%	24.6%	19.0%	30.7%	13.6%	5.1%	8.3%
GPT2F*	**45.1%**	20.3%	14.7%	**65.8%**	**47.6%**	55.3%	31.1%	11.8%	18.6%
GPT2S*	41.7%	18.7%	13.9%	61.1%	45.0%	53.6%	31.0%	13.4%	17.8%
VAE**	26.7%	13.5%	14.6%	44.7%	33.1%	46.2%	24.8%	9.7%	16.5%
WAE**	11.7%	5.2%	3.6%	20.5%	16.7%	26.0%	11.2%	3.6%	6.2%
GPT2F**	36.4%	19.9%	22.1%	57.1%	42.4%	56.5%	34.8%	14.4%	24.3%
GPT2S**	37.6%	20.7%	**22.7%**	58.3%	43.7%	**58.0%**	**36.0%**	**14.5%**	**25.3%**

Table 4: Evaluate the model password-matching performance on all datasets with 10^9 generated passwords. (*) Models trained on the 'rockyou' dataset; (**) Models trained on 'pwnd' dataset. Evaluation is done on the full size of the dataset except for the 'rockyou' and 'pwnd' where the models are evaluated only on the test set (20% of the full size).

presented in Figure (3b) is the distribution of the passwords with respect to the type of characters contained in the password. Half of the passwords approx. 50% in the ´pwnd´ dataset contain digits and lowercase letters (**dl**). All of the models roughly are matching the number of passwords containing **dl**. The GPT2S model generates more passwords containing only digits than the other models including also the empirical data. The last statistics is the distribution over the number of segments per password, i.e., the length of a password. Roughly 42% of the passwords in the 'pwnd' dataset are composed of two segments. From Figure (3c) one can see that GPT2S prefers generating passwords composed of one segment. Whereas, the VAE is roughly following the empirical data distribution and most of the generated passwords are with two segments. As the number of segments grows the difference between the empirical distribution and the distribution of the passwords generated by the models is reducing.

5.3 Password guessing performance

To evaluate the power of our password generation methods we match generated passwords to a predefined test set. Given a dataset of passwords we split into train and test (80% / 20%) and use the train set to train the deep learning text generation model. Once the model is trained we generate a fixed number of passwords (in our experiments up to 10^9). We are interested in the number (both total and as ratio) of generated passwords that appear in the test set.

For the first evaluation we train a model on a train dataset and test on the corresponding test dataset, which comes from the same original distribution of passwords. In practice passwords of interest rarely follow a known distribution one could train a model on. We therefore evaluate our performance by training on the train split of one dataset and evaluating on the test split of another dataset.

Table (4) presents all the deep models trained on the 'rockyou' or 'pwnd' dataset and evaluating on the full 'linkedin', 'myspace', 'yahoo', 'seclist', 'skullsecurity', 'youku' and 'zomato' dataset. In the case of 'rockyou' and 'pwnd' only the test set is used for evaluation. Overall, it is clear from the table that the different training/architecture has a drastic performance impact.

First, focusing on the 'rockyou' trained models in Table (4) we see that the PassGAN benchmark is significantly outperformed by VAE and transformer-based models (up to almost three times on e.g. 'rockyou' and 'linkedin' datasets). On one hand, this demonstrates how effective model selection may lead to substantial matching improvements; on the other hand, one might speculate that the direct application of GAN-based methods is sub-optimal when one works with password datasets with richer latent structure. In contrast, latent models and more sophisticated representation (AEs and attention) techniques seem much more capable of effectively learning a complex latent structure - for a related analysis of more sophisticated GAN-applications we refer to [22]. Further, the table provides evidence that the top performing 'rockyou'-trained models (across most datasets) are our proposed VAE-architecture along with the GPT transformers, where the margin with the second and third competitors (WAE and PassGAN, respectively) is significant.

In terms of the VAE, a couple of important observations are: 1. Interestingly, although having a very different structure, the VAE performs very similarly to the proposed GPT-models, thus suggesting that perhaps some internal password dataset features (e.g. complexity/margins/topology) are crucially guiding the performance of both approaches; 2. Compared to other autoencoding approaches, such as the WAE, it is clear that VAE could perform almost twice as good in terms of matching. This suggests that the WAE, although aiming to provide better reconstruction and training performance, appears to lack the needed flexibility to adapt an efficient latent space representation by forcing the aggregated posteriors to match the fixed prior distribution [27].

A similar analysis of the 'pwnd' trained models in Table (4) can be brought forward, where the overall model performance is reduced in comparison to the 'rockyou' trained models. Here, in contrast, the transformers seem to have a clear advantage over the autoencoding methods, with VAE and WAE being, respectively, second and third in performance. These observations illustrate the effect of the training dataset's generalization ability - the 'rockyou' training appears much richer than the 'pwnd' one and, curiously, renders VAE almost equivalent in performance to the attention-driven solutions.

Note however the effect of training dataset and pretraining procedure on the GPT2-based models. We denote with GPT2F models that were first trained on a large corpus of regular english text and then only finetuned on password data, and denote with GPT2S the corresponding model that was trained from scratch only on password data. We here see that pretraining and finetuning seems to improve the performance on almost all datasets when finetuning happens on the 'rockyou' dataset, for some test datasets we see an increase in multiple percentage points over the model trained from scratch. The opposite effect is happening on the models trained on 'pwnd'. Here we observe the model trained from scratch beating the finetuned model, athough by a smaller margin. Keeping in mind that the 'rockyou' training dataset is significantly smaller than the 'pwnd' dataset (10.1M and 219.6M passwords respectively) we hypothesize that finetuning provides additional information on the general use of language and additional vocabulary that may not be present in smaller dataset but available in the much larger 'pwnd' dataset.

In Table (5) we present a thorough description of the performances of all the models proposed/tested in this paper. The models GPT2F, GPT2S, VAE and WAE are trained on both 'rockyou' and 'pwnd' separately, with exception to the PassGAN that has only been trained on the 'rockyou' dataset.

The results in Table (5), on one hand, provide a thorough overview of matching-performance on 'rockyou' and 'pwnd' in terms of the (guessing) sample size. Moreover, we also illustrate the generation of unique passwords in terms of the sample size, where we emphasize the variability in the model's output. Surprisingly, the top performing solutions in this regard are clearly the autoencoding-based ones (WAE and VAE) trained on 'pwnd' - although falling behind the transformers in terms of matching, the autoencoders seem to provide much greater output variability. Intuitively, the latent space obtained distributions used for generation have a slower decay, whereas the transformers and GANs correspond to much more concentrated sampling distributions.

Finally, in Table (6) we compared the two different pre-processing methods: character and segment based. Lets denote as VAE-char/WAE-char the models trained on the character based pre-processed 'rockyou' dataset and VAE-segment/WAE-segment the models trained on the on the segment based pre-processed 'rockyou' dataset. All the models are evaluated on the 'linkedin' dataset containing 47.9M passwords. One can see from the table that models trained on the character based pre-processed dataset generate larger number of unique passwords. The VAE-char is performing better than the VAE-segment however, the WAE-char performs significantly worse than the WAE-segment model.

5.4 Application of password augmentation rules

As previously mentioned, password recovery tool Hashcat does not only offer efficient hashing of password candidates, but also allows for the application of rules, which can be used to augment and extend a given password list. A rule is an instruction for augmentation of a single password and is applied to each wordlist entry before hashing. For some sample rules see Table (7).

Application of rules greatly increases the number of password candidates to test and extends our method in a reasonable way: the deep-learning model generates base password candidates and further rules make sure that slight permutations of the base candidate are also considered.

Table (8) shows the effect of application of the rule lists *best64*[8] with 64 rules on our generated password lists. We generate 10^8 password with all models trained on the 'rockyou' training split and observe the number of matches in the

[8] https://github.com/hashcat/hashcat/tree/master/rules/best64.rule

Model	Passwords Generated	Unique Passwords	Matched on ‘rockyou’		Matched on ‘pwnd’	
GPT2F (rockyou)	10^6	9.96×10^5	32 156	1.23%	76 423	0.14%
	10^7	8.74×10^6	220 267	8.42%	611 137	1.11%
	10^8	6.78×10^7	692 514	26.5%	2 981 523	5.43%
	10^9	4.57×10^8	1 140 201	43.8%	8 050 063	14.65%
GPT2S (rockyou)	10^6	9.47×10^5	33 952	1.3%	76 160	0.14%
	10^7	8.48×10^6	212 494	8.13%	578 440	1.05%
	10^8	6.62×10^7	637 188	24.4%	2 794 198	5.08%
	10^9	4.54×10^8	1 090 720	41.7%	7 629 127	13.9%
GPT2F (pwnd)	10^6	9.65×10^5	11 130	0.43%	42 942	0.08%
	10^7	9.23×10^6	71 727	2.74%	359 474	0.65%
	10^8	8.52×10^7	359 929	13.8%	2 662 849	4.85%
	10^9	6.76×10^8	952 613	36.4%	12 155 484	22.1%
GPT2S (pwnd)	10^6	9.68×10^5	11 207	0.43%	44 911	0.08%
	10^7	9.27×10^6	74 756	2.86%	376 451	0.68%
	10^8	8.54×10^7	381 804	14.6%	2 798 394	5.09%
	10^9	6.71×10^8	984 225	37.64%	12 494 652	22.7%
PassGAN (rockyou)	10^6	8.50×10^5	19 835	0.76%	47 978	0.09%
	10^7	6.84×10^6	79 731	3.05%	257 882	0.47%
	10^8	4.83×10^7	212 238	8.12%	962 761	1.75%
	10^9	2.95×10^8	415 859	15.9%	2 572 591	4.68%
VAE (rockyou)	10^6	9.93×10^5	26 814	1.03%	63 381	0.12%
	10^7	9.33×10^6	190 371	7.28%	519 099	0.94%
	10^8	7.86×10^7	638 790	24.4%	2 782 128	5.06%
	10^9	5.99×10^8	1 175 420	44.9%	8 456 376	15.4%
VAE (pwnd)	10^6	9.99×10^5	2 362	0.09%	15 093	0.03%
	10^7	9.98×10^6	23 252	0.89%	150 699	0.27%
	10^8	9.82×10^7	183 202	7.01%	1 360 894	2.48%
	10^9	8.82×10^8	698 605	26.7%	8 020 259	14.6%
WAE (rockyou)	10^6	7.41×10^5	7 666	0.29%	21 217	0.04%
	10^7	7.14×10^6	56 903	2.18%	176 971	0.32%
	10^8	6.48×10^7	271 908	10.4%	1 106 844	2.01%
	10^9	6.62×10^8	738 685	28.2%	4 736 310	8.62%
WAE (pwnd)	10^6	7.55×10^5	690	0.03%	3 329	0.01%
	10^7	7.55×10^6	6 476	0.25%	32 318	0.06%
	10^8	7.48×10^7	54 145	2.07%	280 953	0.51%
	10^9	9.50×10^8	306 834	11.7%	1 960 948	3.57%

Table 5: Full evaluation for all trained models. All rockyou/pwnd models trained on the rockyou/pwnd training dataset (80% of the corpus) and evaluated on the test dataset (20% of the corpus).

Preprocessing		**Chars**		**Segment**	
Model	Passwords Generated	Unique Passwords	Matched Passwords	Unique Passwords	Matched Passwords
VAE	10^6	9.93×10^5	0.22%	9.53×10^5	0.20%
	10^7	9.33×10^6	1.80%	8.95×10^6	1.64%
	10^8	7.86×10^7	9.26%	7.42×10^7	8.01%
	10^9	5.99×10^8	24.5%	5.74×10^8	21.4%
WAE	10^6	9.82×10^5	0.01%	7.41×10^5	0.06%
	10^7	9.71×10^6	0.06%	7.14×10^6	0.54%
	10^8	9.52×10^7	0.57%	6.48×10^7	3.51%
	10^9	9.18×10^8	3.30%	6.62×10^8	14.3%

Table 6: Comparing the performance of the models trained on the 'rockyou' dataset using the *character* based or *segment* based pre-processing. Models are evaluated on the 'linkedin' dataset (47.3M passwords).

rule	description	example
`l`	lower case string	`passWord` → `password`
`c`	capitalize string	`passWord` → `PassWord`
`$1`	append character	`passWord` → `passWord1`
`sa@`	switch characters	`passWord` → `p@ssWord`
`c sa@ $1 $2`	combined rules	`passWord` → `P@ssWord12`
`l so0 sa@ ss5`	combined rules	`passWord` → `p@55w0rd`

Table 7: Sample rules for Hashcat. The tool allows basic string manipulations and advanced transformations by chaining rules.

test split before and after application of rules. Comparing M_0 (matches before application of rules) and M_{best64} (matches after application of rules) we observe a significant increase in matches for all models. For comparison we also apply the same ruleset to the 'rockyou' training split itself, which generates 0 matches before application of rules (there are no duplicates in the 'rockyou' dataset) and 9.1×10^5 matches with rules. The training data as basis therefore produces less matches than all models except for the PassGAN. We conclude that the further application of rules to the output of our models is a reasonable strategy to further increase password recovery performance, and that training deep-learning models on the training data split for password generation provides value surpassing the application of predefined rules.

5.5 Comparison to the established methods

In order to compare our models to the established methods from Section 2, we evaluate on a third dataset. We use our training split of 'rockyou' (80%, 10.4M passwords) to generate new passwords using various established methods and

model	N_0	N_0^*	M_0	N_{best64}	M_{best64}
VAE	10^8	7.86×10^7	6.38×10^5	5.18×10^9	12.9×10^5
WAE	10^8	6.48×10^7	2.72×10^5	4.27×10^9	9.6×10^5
GPT2S	10^8	6.62×10^7	6.37×10^5	4.36×10^9	11.2×10^5
GPT2F	10^8	6.78×10^7	6.92×10^5	4.47×10^9	11.0×10^5
PassGAN	10^8	4.83×10^7	2.12×10^5	3.18×10^9	7.0×10^5
‘rockyou’ (train)	1.05×10^7	1.05×10^7	0	6.90×10^8	9.1×10^5

Table 8: Effects of application of Hashcat rules to 10^8 generated passwords from various models trained on ‘rockyou’ training data split when matching the ‘rockyou’ test split. For comparison we apply rules directly to the training dataset. N_0: Number of generated passwords or size of dataset. N_0^*: Number of unique generated passwords or items in the dataset. M_0: Original matches in target dataset. N_{best64}: Number of password candidates when applying *best64* rules to the unique passwords. M_{best64}: Number of matches in target dataset candidates when applying *best64* rules.

Model	Unique Passwords	Matches
3-gram Markov Model	4.35×10^8	4.27×10^6
Hashcat – best64	6.66×10^8	7.26×10^6
Hashcat – gen2	8.49×10^8	2.55×10^6
PCFG v4.1	9.71×10^8	12.52×10^6
PRINCE v0.22	9.99×10^8	1.65×10^6
PassGAN (ours)	2.95×10^8	3.2×10^6
GPT2S	4.54×10^8	8.85×10^6
GPT2F	4.57×10^8	9.60×10^6
VAE	5.99×10^8	10.3×10^6
VAE_S	5.74×10^8	9.06×10^6
WAE	9.18×10^8	1.35×10^6
WAE_S	6.62×10^8	6.10×10^6

Table 9: Results of our evaluation on the ‘linkedin’ dataset (47.3M passwords). All our models were trained on ‘rockyou’ and generated 10^9 passwords, all models above generated 10^9 passwords or the maximum number of possible combinations from the ‘rockyou’ training split.

evaluate on a subset of the ‘linkedin’ dataset (originally 60.7M passwords). We prepare this dataset by removing all entries longer than 12, all entries containing non-ascii characters and all entries that also appear in our ‘rockyou’ training split. We are left with a test set of 47.3M passwords.

For comparison, we train PCFG[9] on a non-unique version of the training split, i.e. passwords appear multiple times in the frequency of the original leak, and generate 10^9 passwords. We use Hashcat to apply two rulesets to the training split of unique passwords. Ruleset *best64* contains 64 rules and generates 6.9×10^8 passwords in total. Ruleset *generated2*[10] contains 65k rules and generates an exceedingly large number of password candidates, of which we sample 10^9 passwords. Both lists are the result of large-scale quantitative evaluations of the effect of various hand-written and machine generated rules on multiple wordlists, password datasets and target hashes. We additionally train a simple 3-gram Markov Model[11] on the unique training split and generate 10^9 passwords. Finally we use the PRINCE algorithm[12] to construct 10^9 passwords of length 4 to 12 from the 'rockyou' training set.

For our models, trained on the 'rockyou' training split we generate 10^9 passwords each and count the matches in the 'linkedin' test data. Table (9) shows the results.

We first observe that all trained models recover a significant amount of passwords from the 'linkedin' test data. Ranging from 1.35M (WAE) to 10.3M (VAE) there is large variance in the performance of the individual models. Interestingly we observe no correlation between number of unique generated passwords and number of matches both in our model and the comparison methods, the trained models with most and least unique passwords (2.95M for PassGAN and 9.18M for WAE_S) match the least number of passwords (3.2M and 1.35M respectively)

Both implementations of VAE and GPT2 respectively achieve very high matching results, with the character-based VAE representing the top-performer with 10.3M matches. Only the probabilistic PCFG algorithm can surpass this model by another 20%. These trained models additionally score higher than all other comparison methods.

5.6 Operations in latent space

Similarity in latent space. The learned latent space by the encoder imposes geometric connections among latent points that have some semantic similarity in the data space. This means that similar points in data space have latent representation that are close to each other. This property can be used also for password generation. Let us assume that we have the password `veronica2296` and we want to generate variants of this passwords. To this end we encode the password `veronica2296` into its latent representation $\mathbf{z}_t$. We parametrize the posterior using the $\mathbf{z}_t$ as mean ($\boldsymbol{\mu} = \mathbf{z}_t$), next we sample latent codes from that region ($\sigma = 0.001$) and we generate passwords $\mathbf{z}_i \sim \mathcal{N}(\boldsymbol{\mu}, \sigma\mathbf{I})$. The results from this task are presented in Table (10). One can see that most of the passwords generated from this region contain the word `veronica` in combination with

[9] https://github.com/lakiw/pcfg_cracker

[10] https://github.com/hashcat/hashcat/tree/master/rules/generated2.rule

[11] https://github.com/brannondorsey/markov-passwords

[12] https://github.com/hashcat/princeprocessor

different number or variants of the name `veronica` ex. `veronico` and a number. This shows that our models have learned semantically meaningful latent space given the training set.

veronica2286	veronica296	verogani2297	veronica2229	veroicata22U
veronica22U6	veronic22259	veroinca2297	verolica2296	verotEic2246
verogama2296	veronica2269	veronica22_9	veronica239	verolica2298
veroga_a2986	veronic2205	veronaxa2269	veronico2259	verolicat269
verosgaj2!98	vertinac2219	veronica249	verincia22T6	veronema2205
veroina22_6	verozica22_	veroszi2246	veronkea2295	verolica2295
veroneza2269	veron_ma2295	verogaxa2299	veronica4298	verote5a229D
veronica20g6	verogala285	verozoca24_7	veronic22#6	veronoca28/5
veronico23D	veronema2276	veronica2296	veronican298	veronicas25
vroricak229	verongic2279	vegaroaka269	veronica2896	vetRonic225

Table 10: Samples with latent representation close to *veronica2296* latent representation.

Conditional password generation. Having latent representation for each password allows us to also do conditional generation. Let ***love*** be a template password. The '*' symbol it is a placeholder for any character defined in the vocabulary. We can condition our model to generate passwords that contain the word 'love' in the middle, with three random characters as prefix and suffix. In Table (11) are presented some conditionally generated samples. For a further thorough analysis of conditional password sampling in terms of EM-based algorithms we refer to [22]. In particular, in this regard our VAE model exhibits a competitive performance as seen in Table (11).

9alolove71u	nublove85/9	miblovenv11	cetlovesder
licloverrs9	siclove00me	riglover2k	biolove121
hicloven3ke	failoveye4	n2ulovemswo	inudlove12
lyaloveji8	gemloveso1	irolovesor	Vealover.v
ltelovejr*	vatlover10	mejlovey4u	sewalover79

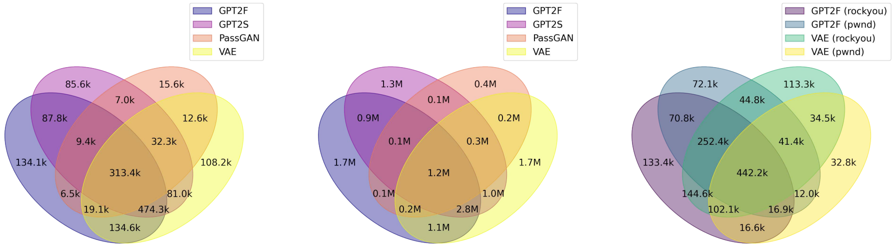
Table 11: Conditional generation of passwords. We condition the generation on ***love***.

Interpolation between two passwords. Another possibility with the VAE models is to do interpolation between two passwords. Let us assume we have the password A and the password B. We use the encoder to obtain the latent representations $\mathbf{z}_A$ and $\mathbf{z}_B$ for A and B respectively. Next, we sample latent representations from the linear path former by $\mathbf{z}_A$ and $\mathbf{z}_B$ and using the decoder we obtain the corresponding passwords. The process of sampling new passwords in this way is called *interpolation*. In Table (12) are given some example interpolations. Each column in the table is an example interpolation. The top row is the password A and the bottom row is the password B. In the middle are the passwords generated from the latent representations sampled form the linear path between A and B.

remington223	pepegrillo16	newthanakorn	musikALE0991	DIMAYUGAluna	fressikarosa
rtninton2131	popegrillo16	nevwhanfaror	muwik0134902	DIMAYUYAUMEM	krezsickiess
Atnitton2133	petckrllole7	new_hanaqira	muwIE1250894	DIMuYuYnuMAY	kryesikarish
ntn123to2131	hoperillel62	newsafarhrig	memIE1284989	GIMAYuYuuakH	kynesiha2308
rmnm2et21213	agterillo652	18stahankoua	mT0130281948	VUHuwuNAuMMe	kynayiha2s09
nnnmot121032	ktanlilel657	18s7tafabano	K72636481934	WHfluvusue85	cynayoshi_09
nnd231523465	allininal505	129swokfaton	172639451658	Tofluousuck9	cy2ayueh2990
ontit1321642	allinllang08	1287tockokno	172639293483	WofdmousuS86	clyayosh1090
andiew123456	alliannan680	12960ocktint	172839481614	moodmous6885	clya23081099
andrew123456	allianna0685	1297bucktown	172839456123	woodmouse598	clya23081990

Table 12: Example interpolations. Each column is one interpolation where the top row is the start password A and the last row is the end password B. In between are generated passwords.

5.7 Intersection of generated passwords



(a) Models trained on 'rockyou' matched on 'rockyou' test set.

(b) Models trained on 'rockyou' matched on 'pwnd' test set.

(c) Models trained on 'rockyou' or 'pwnd' matched on 'rockyou' test set.

Fig. 13: We compare the passwords matched by various models on different datasets to analyse whether the output of different models is sufficiently different. Numbers given in thousands or millions.

Table (2) showed that different model architectures qualitatively seem to generate vastly different passwords, however the question remains if different architectures and different training datasets actually lead to a quantitatively different generation and match different subsets of the test dataset. Take for instance the two GPT2-based models trained on the 'rockyou' dataset. Tables (2) and (2c) show qualitatively a different distribution on generated passwords, however Table (5) tells us that both models match around 40% of the passwords in the 'rockyou' test dataset. Figure (13) therefore analyses the intersection of passwords matched on the 'rockyou' and 'pwnd' test dataset between the models.

Figure (13a) shows the intersections of the models GPT2F, GPT2S, PassGAN and VAE, trained on the 'rockyou' dataset and matched on the 'rockyou' training set. We see that while there is a large overlap between matched passwords, with 313k passwords appearing in every model output, each model still has a significant amount of unique passwords. In this respect there are large differences between the models. While the GPT2F model recovered 134k password no other model could recover, our PassGAN implementation only had 16k unique found passwords, and we find the largest intersection in the passwords that all models except PassGAN could recover. All models in total matched around 1.5M passwords, or 30% more than the best single model (VAE with 1.18M recovered passwords).

The effect is even more pronounced when evaluating the same models, trained on the 'rockyou' dataset, tested on the 'pwnd' dataset in Figure (13b). We again see a similar behaviour as in Figure (13a), with all models generating a significant amount of unique passwords matched on the test set. Again the GPT2 models and the VAE generate more unique found passwords than the PassGAN (1.3M – 1.7M against 0.4M passwords), and the intersection between the three non-PassGAN models represent the largest subset of found passwords (2.8M passwords). In total the models generated 13.1M unique passwords found in the 'pwnd' data, or 55% more than the best single model (VAE with 8.45M recovered passwords).

Finally Figure (13c) analyses the effect of training data on the matched passwords. We compare the GPT2F and VAE models, both trained on 'rockyou' and pwnd. While now the largest subset lies in the intersection of all four models (442k), again many passwords in the 'rockyou' test set are only found by one model. The models trained on 'rockyou' unsurprisingly offer a better performace than the models trained on pwnd. While the 'rockyou' models together recovered 391k passwords that none of the 'pwnd' models could generate, the 'pwnd' models together only found 117k unique passwords. Similar to the analysis of Figure (13a), all models recover a total of 1.5M passwords, or 30% more than the best single model (VAE (rockyou) with 1.18M recovered passwords).

We conclude that there is a significant overlap in recovered passwords from each model, which we do not find surprising given that the models share common architectures or training datasets and the best models alone already recover over 40% of the entire test set. However the diagrams also reveal that many models find a large amount of unique passwords, and the set of all password generations outperforms any single model by a large margin. Each model therefore offers unique password recovery functionality and could be considered in a password recovery task.

6 Conclusion and Future Work

The present work illustrates various deep learning password generation techniques. Conducting a thorough unified analysis we discuss password-matching capabilities, variability and quality of sampling and robustness in training. On one hand, we bridge and extend previous methods based on attention schemes, GANs and Wasserstein autoencoding; on the other hand, we provide a promising novel approach based on Variational Autoencoders that allows for efficient latent space modeling and further sampling mechanisms. Lastly, we hope our work will facilitate and provide benchmark lines for further deep learning and ML practitioners interested in the field of password guessing.

In terms of further investigation, the application of deep learning techniques to password generation poses further intriguing questions on the interplay between classical probabilistic methods and neural networks, where one would ultimately hope to construct more efficient and reliable domain-inspired password representation schemes - e.g. based on carefully crafted fragmentations.

Acknowledgment

This project was funded by the Federal Ministry of Education and Research (BMBF), FZK: 16KIS0818. The authors of this work were supported by the Competence Center for Machine Learning Rhine Ruhr (ML2R) which is funded by the Federal Ministry of Education and Research of Germany (grant nos. 01|S18038B, 01|S18038C). We gratefully acknowledge this support.

References

1. Hashcat – advanced password recovery. https://hashcat.net/hashcat/. Accessed: 2020-12-07.
2. Hashcat raking generated2.rule. https://github.com/evilmog/evilmog/wiki/Hashcat-Raking—generated2.rule. Accessed: 2020-12-07.
3. John the ripper password cracker. https://www.openwall.com/john/. Accessed: 2020-12-07.
4. Troy hunt: Here's why [insert thing here] is not a password killer. https://www.troyhunt.com/heres-why-insert-thing-here-is-not-a-password-killer/. Accessed: 2020-12-07.
5. M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. In *34th International Conference on Machine Learning, ICML 2017*, 2017.
6. S. R. Bowman, L. Vilnis, O. Vinyals, A. M. Dai, R. Jozefowicz, and S. Bengio. Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349*, 2015.

7. K. Chanda. Password security: An analysis of password strengths and vulnerabilities. *International Journal of Computer Network and Information Security*, 8:23–30, 07 2016.
8. Z. Dai, Z. Yang, Y. Yang, J. G. Carbonell, Q. V. Le, and R. Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *CoRR*, abs/1901.02860, 2019.
9. M. Dell'Amico, P. Michiardi, and Y. Roudier. Password strength: An empirical analysis. pages 1 – 9, 04 2010.
10. J. Devlin, M. W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference*, 2019.
11. I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.
12. I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, 2014.
13. I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville. Improved training of wasserstein GANs. In *Advances in Neural Information Processing Systems*, 2017.
14. B. Hitaj, P. Gasti, G. Ateniese, and F. Perez-Cruz. Passgan: A deep learning approach for password guessing. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11464 LNCS:217–237, sep 2017.
15. S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.
16. S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *32nd International Conference on Machine Learning, ICML 2015*, 2015.
17. D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
18. D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.
19. D. P. Kingma and M. Welling. An introduction to variational autoencoders. *arXiv preprint arXiv:1906.02691*, 2019.
20. H. Li, M. Chen, S. Yan, C. Jia, and Z. Li. Password guessing via neural language modeling. In X. Chen, X. Huang, and J. Zhang, editors, *Machine Learning for Cyber Security*, pages 78–93, Cham, 2019. Springer International Publishing.
21. W. Melicher, B. Ur, S. M. Segreti, S. Komanduri, L. Bauer, N. Christin, and L. F. Cranor. Fast, lean, and accurate: Modeling password guessability using neural networks. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 175–191, Austin, TX, Aug. 2016. USENIX Association.
22. D. Pasquini, A. Gangwal, G. Ateniese, M. Bernaschi, and M. Conti. Improving password guessing via representation learning. *42nd IEEE Symposium on Security and Privacy (Oakland)*, 2021.
23. L. Rabiner and B. Juang. An introduction to hidden markov models. *IEEE ASSP Magazine*, 3(1):4–16, 1986.
24. A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Gpt2. *Open AI*, 2019.
25. R. Sennrich, B. Haddow, and A. Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.

26. C. K. Sønderby, T. Raiko, L. Maaløe, S. K. Sønderby, and O. Winther. How to train deep variational autoencoders and probabilistic ladder networks. In *33rd International Conference on Machine Learning (ICML 2016)*, 2016.
27. I. Tolstikhin, O. Bousquet, S. Gelly, and B. Schoelkopf. Wasserstein auto-encoders. *arXiv preprint arXiv:1711.01558*, 2017.
28. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.
29. C. Villani. *Topics in optimal transportation*. Number 58. American Mathematical Soc., 2003.
30. M. Weir, S. Aggarwal, B. de Medeiros, and B. Glodek. Password cracking using probabilistic context-free grammars. pages 391–405, 05 2009.
31. Z. Yang, Z. Dai, Y. Yang, J. G. Carbonell, R. Salakhutdinov, and Q. V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. *CoRR*, abs/1906.08237, 2019.
32. S. Zagoruyko and N. Komodakis. Wide Residual Networks. In *British Machine Vision Conference 2016, BMVC 2016*, 2016.