Advances in Password Recovery Using Generative Deep Learning Techniques

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

[1] Fraunhofer IAIS, Sankt Augustin, Germany
Kostadin.cvejoski@iais.fraunhofer.de
[2] University of Bonn, Bonn, Germany
[3] Competence Center for Machine Learning Rhine-Ruhr (ML2R), Dortmund, Germany
[4] 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, MySpace, 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 [10]. However studies show that users tend to set their individual passwords predictably, favoring short strings, names, birth dates and reusing passwords across sites [16,17]. 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 Sect. 2). These methods rely on expert knowledge and analysis

D. Biesner and K. Cvejoski—Equal contribution.

I. Farkaš et al. (Eds.): ICANN 2021, LNCS 12893, pp. 15–27, 2021.
https://doi.org/10.1007/978-3-030-86365-4_2

of various password leaks from multiple sources to generate rules and algorithms for efficient exploitation of learned patterns.

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 of new architectures and mechanisms, advanced representation capabilities and training procedures.

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) our 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; (4) we additionally analyse the effect of model pre-training on general language data for the password generation task against training on pure password data; (5) our novel variational autoencoder (VAE) approach allows more flexible latent representations and outperforms previous autoencoding methods based on Wasserstein training [26]; (6) the VAE provides a state-of-art password matching performance as well as further sampling possibilities (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 John-TheRipper [4]), 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 [27] or more sophisticated approaches like *probabilistic context free grammar* (PCFG) [31], 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 [25] to recent seminal works applying state-of-the-art text generation methods to password generation: Generative adversarial networks (GANs) [20,26], Wasserstein Autoencoders [26], and bidirectional RNNs trained with the aid of pre-trained Transformer models [24].

Our work extends this palette of deep learning architectures with the Variational Autoencoder [23] and Transformer-based language models [28]. 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

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, which provides means to efficiently compute discrepancies between two given distributions. We refer to [14,18,19] for further background.

Concerning password guessing and generation our starting point is the well-known PassGAN model proposed in [20]. The 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 [21]).

Deep Latent Variable Models. The *Variational Auto Encoder* (VAE) [23] is a framework for efficient optimization of *deep latent variable models* (DLVM). It comprises of two main components: *encoder* and *decoder*. The encoder is a 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 evidence lower bound (ELBO)

$$\mathcal{L}(\theta, \phi, \mathbf{x}^{(\mathbf{i})}) = -D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}^{(i)})||p_\theta(\mathbf{z})) + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x}^{(i)})}\left[\log p_\theta(\mathbf{x}^{(i)}|\mathbf{z})\right]. \quad (1)$$

The model learns to reconstruct the password $\mathbf{x}$ given to the input by first, mapping the password to a distribution of latent codes $q_\phi(\mathbf{z}|\mathbf{x})$, then sampling from the posterior distribution and passing the latent code $\mathbf{z}$ to the decoder $p_\theta(\mathbf{x}^{(i)}|\mathbf{z})$. During training a strong prior $p_\theta(\mathbf{z})$ is imposed on the learned latent code distribution. Usually in the VAE framework the prior is set to be centered isotropic Gaussian distribution $p_\theta(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$, which enables us to later easily sample from the prior 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 may be based on the structure of the password, like a common substring.

Transformers. In recent years the transformer, originally applied to machine translation in [30], has become increasingly popular, with transformer-based architectures setting new benchmarks in text generation, machine translation and many other natural language processing 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 $\textbf{Attention}(Q, K, V) = \textbf{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$, 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.

In our work we apply the GPT2 [28] architecture to the password modeling task. GPT2 is a transformer-based language model, trained on the causal language modeling 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 $\mathcal{L}(\mathcal{U}) = \sum_{i=1}^{N} p_\theta(u_i|u_{i-1}, \ldots, u_{i-k})$.

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 Sect. 4.2.

4 Results

4.1 Data

There are several datasets of passwords publicly available. These lists contain passwords that were at some point in time leaked to the public from 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. We apply the same preprocessing to all datasets: Removing duplicates, removing passwords <4

and >12 characters and removing passwords containing characters other than letters, numbers and punctuation `.!?* $_#&/+`.

The specific password datasets we employ for training or evaluation are: ‘rockyou’ (13.0M passwords, [7]); ‘Have I Been Pwnd V1’/‘pwnd’ (274.8M, [3]), ‘linkedin’ (60.1M, [5]), ‘myspace’ (53k, [6]), ‘yahoo’ (430k, [11]), ‘youku’ (48M, [12]), ‘seclist’ (969k, [8]), ‘skullsecurity’ (6.2M, [9]), ‘zomato’ (6.3M, [13]). All counts after preprocessing. Note that all datasets are sets of unique passwords. Since the main task is not accurate learning of the training data distribution but generation of new passwords we hypothesize that training on unique passwords improves the guessing performance. Benchmarks on the VAE-model have shown a slight improvement over training on the original non-unique password leak.

For training we employ splits of the rockyou (80%, 10.4M training/20%, 2.6M testing) and pwnd (80%, 219.8M training/20%, 54.9M testing).

4.2 Experimental Setup

GAN-Based Models. In our setup, we essentially utilize the PassGAN as a standard benchmark - this is well motivated by previous substantial studies of GAN-based models [20,26]. 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 [20].

The PassGAN training is based on state-of-art practices such as Wasserstein GAN and gradient clipping (cf. [14]), as well as gradient penalty regularization (cf. [19]). We used a batch-size of 256, a gradient-penalty-hyperparameter $\lambda = 10$ and 10 discriminator iterations per generator step. The preferred gradient descent was based on ADAM [22] with an initial learning rate $\eta = 10^{-4}$ and momentum parameters $\beta_1 = 0.5, \beta_2 = 0.9$; a fixed-interval annealing with an iteration step of 10^6 was also used.

Note that [26] report improvements of the GAN-architecture and training for password generation by adding additive noise to the input, replacing the residual blocks with deeper residual bottleneck blocks and applying batch normalization to the generator. We compare to the results reported in their paper in Sect. 4.4, Table 2.

Variational Autoencoders. 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 [22] with learning rate $\eta = 10^{-4}$ and momentum $\beta_1 = 0.5$, $\beta_2 = 0.9$. The batch size is chosen to be 128 and we also use early stopping. Following [15], we use KL cost annealing strategy.

Table 1. Most common generated passwords per model, separated by training dataset.

Model	'rockyou'	'pwnd'	Model	'rockyou'	'pwnd'	Model	'rockyou'	'pwnd'	Model	'rockyou'
	`leslie`	`MARIA`		`love`	`2010`		`ilove`	`1234`		`123456`
	`yankee`	`hilton`		`mrs.`	`love`		`love`	`2010`		`12345`
	`kirsty`	`SEXY`		`baby`	`1234`		`baby`	`love`		`123456789`
	`jeremy`	`4678`		`sexy`	`2000`		`iluv`	`2000`		`1234567`
VAE	`claudia`	`NATA`	GPT2S	`girl`	`2009`	GPT2F	`sexy`	`2009`	PassGAN	`12345678`
	`gangsta`	`ALEX`		`angel`	`2008`		`pink`	`2008`		`angela`
	`violet`	`JOSE`		`1992`	`12345`		`memyself`	`12345`		`angels`
	`andrei`	`BABY`		`1993`	`2011`		`caoimhe`	`2011`		`angel1`
	`jennifer`	`MAMA`		`1994`	`2007`		`cintaku`	`2007`		`buster`
	`natalie`	`ANGEL`		`2007`	`1992`		`jess`	`1987`		`128456`

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. 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 expect 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 will not appear with a model trained from scratch since all the text it has ever known are passwords. We call this model GPT2-Scratch or GPT2S.

We train the model using an openly available implementation of GPT2,[1] with the default *gpt2*[2] model as pre-trained 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 [29].

4.3 Analysis of Generated Passwords

In Table 1 we compare the most common passwords generated by our models along with the most commonly generated passwords by the comparison methods.

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

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

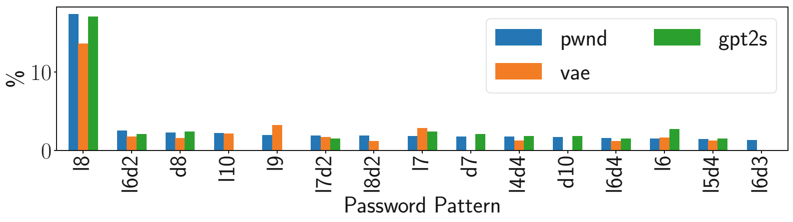

Fig. 1. Statistics on password structure for 'pwnd' dataset and passwords generated by models trained on 'pwnd'.

We observe 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. Finally, for the PassGAN model implemented on the 'rockyou' training data. 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 when trained on 'rockyou'.

Additionally, to estimate whether our trained model follow the distribution of the empirical dataset (the training set), we introduce *PCFG-like* statistics [31] – password `password!23` is split into pure segments as **L8S1D2** where **L** is alpha character, **S** is special symbol character and **D** is a digit.

The most common password structure for the 'pwnd' dataset is **l8** (see Fig. 1) with almost 20% of the whole dataset. The GPT2S model successfully matches the number of passwords with structure **l8**. For the rest of the password structures the GPT2S and the VAE model perform roughly the same and approximately match the underlying empirical distribution.

4.4 Password Guessing Performance

To evaluate the power of our password generation methods we match generated passwords to a predefined test set. As mentioned in Sect. 4.1, 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.

Table 2 presents all the deep models trained on the 'rockyou' or 'pwnd' dataset and evaluating on the full 'linkedin', 'myspace', 'yahoo', 'seclist',

Table 2. 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). Pasquini [26] trained PassGAN on a different 80/20 'rockyou' split and report 23.3%.

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%
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%
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%**

'skullsecurity', 'youku' and 'zomato' dataset. In the case of 'rockyou' and 'pwnd' only the test set is used for evaluation.

First, focusing on the 'rockyou' trained models in Table 2 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). The improved GAN-architecture and training from [26] reports a 23.3% recovery performance for a model trained on rockyou evaluated on rockyou. This is a clear improvement over the PassGAN benchmark from [20] but still does not reach the latent variable or transformer architectures. 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.

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.

A similar analysis of the 'pwnd' trained models in Table 2 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 method.

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.

4.5 Comparison to Established Methods

In order to compare our models to the established methods mentioned in Sect. 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 evaluate on a subset of the 'linkedin' dataset (originally 60.7M passwords). Additionally to the preprocessing mentioned in Sect. 4.1 we remove all entries that also appear in our 'rockyou' training split. We are left with a test set of 47.3M passwords.

Table 3. 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. (*) Numbers taken from [20] for comparison, trained and evaluated on different data splits.

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
FLA*	7.4×10^8	8.29×10^6
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

For comparison, we train PCFG[3] 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*[4] 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[5] on the unique training split and generate 10^9 passwords. Finally we use the PRINCE algorithm[6] to construct 10^9 passwords of length 4 to 12 from the 'rockyou' training set.

[3] https://github.com/lakiw/pcfg_cracker.
[4] https://github.com/hashcat/hashcat/tree/master/rules/generated2.rule.
[5] https://github.com/brannondorsey/markov-passwords.
[6] https://github.com/hashcat/princeprocessor.

Additionally we compare to the FLA [25] experiments done in [20]. Note that the model is trained on a 'rockyou' split of 9.9M passwords, evaluated on 'linkedin' test set of 40.6M passwords, both only containing passwords $\leq$10 characters.

For our models, we train on the 'rockyou' training split, generate 10^9 passwords each and count the matches in the 'linkedin' test data. Table 3 shows the results. We first observe that all trained models recover a significant amount of passwords from the 'linkedin' test data. Ranging from 3.2 (PassGAN) to 10.3M (VAE) there is large variance in the performance of the individual models. For these models we additionally observe a correlation between number of unique generated passwords and number of matches.

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%. The VAE and GPT2 models additionally score higher than all other comparison methods.

4.6 Operations 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 ($\boldsymbol{\mu} = \mathbf{z}_t$), sample latent codes from that region, $\mathbf{z}_i \sim \mathcal{N}(\boldsymbol{\mu}, \sigma\mathbf{I})$ with $\sigma = 0.001$, and then generate passwords by passing the latent codes to the decoder. The results from this task are presented in Table 4. One can see that most of the passwords generated from this region contain the word `veronica` in combination with different number or variants of the name `veronica` (e.g. `veronico`) and a number. This shows that our models have learned semantically meaningful latent space given the training set.

Table 4. Latent space models allow for conditioning generation on prior information. **Left**: Samples with latent representation close to the latent representation of `veronica2296`. **Right**: Conditional generation of passwords. We condition the generation on `***love***`.

veronica2286	veronica296
veronica22U6	veronic22259
verogama2296	veronica2269
veroga_a2986	veronic2205
verosgaj2!98	vertinac2219
veronica2229	veroicata22U
veroneza2269	veron_ma2295

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

Having latent representation for each password allows us to also do conditional generation. Let `***love***` be a template password, where `*` 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 by encoding `***love***` (representing the `*` by an `UNK` character) and sampling from the region in latent space. In Table 4 we present some conditionally generated samples. For a further thorough analysis of conditional password sampling in terms of EM-based algorithms we refer to [26].

5 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 previously established methods based on attention schemes and GANs; 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.

Acknowledgement. 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 07 Dec 2020
2. Hashcat raking generated2.rule. https://github.com/evilmog/evilmog/wiki/Hashcat-Raking---generated2.rule. Accessed 07 Dec 2020
3. Have i been pwnd v1. https://hashes.org/leaks.php?id=70. Accessed 07 Dec 2020
4. John the ripper password cracker. https://www.openwall.com/john/. Accessed 07 Dec 2020
5. Linkedin leak. https://hashes.org/leaks.php?id=68. Accessed 07 Dec 2020
6. Myspace leak. https://weakpass.com/wordlist/22. Accessed 07 Dec 2020
7. Rockyou leak. https://weakpass.com/wordlist/90. Accessed 07 Dec 2020
8. Seclist compilation. https://weakpass.com/wordlist/50. Accessed 07 Dec 2020

9. Skullsecurity compilation. https://weakpass.com/wordlist/671. Accessed 07 Dec 2020
10. 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 07 Dec 2020
11. Yahoo leak. https://weakpass.com/wordlist/44. Accessed 07 Dec 2020
12. Youku leak. https://hashes.org/leaks.php?id=508. Accessed 07 Dec 2020
13. Zomato leak. https://hashes.org/leaks.php?id=587. Accessed 07 Dec 2020
14. Arjovsky, M., Chintala, S., Bottou, L.: Wasserstein generative adversarial networks. In: 34th International Conference on Machine Learning, ICML 2017 (2017)
15. Bowman, S.R., Vilnis, L., Vinyals, O., Dai, A.M., Jozefowicz, R., Bengio, S.: Generating sentences from a continuous space. arXiv preprint arXiv:1511.06349 (2015)
16. Chanda, K.: Password security: an analysis of password strengths and vulnerabilities. Int. J. Comput. Netw. Inf. Secur. **8**, 23–30 (2016)
17. Dell'Amico, M., Michiardi, P., Roudier, Y.: Password strength: an empirical analysis, pp. 1–9 (2010)
18. Goodfellow, I.J., et al.: Generative adversarial nets. In: Advances in Neural Information Processing Systems (2014)
19. Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., Courville, A.: Improved training of wasserstein GANs. In: Advances in Neural Information Processing Systems (2017)
20. Hitaj, B., Gasti, P., Ateniese, G., Perez-Cruz, F.: PassGAN: a deep learning approach for password guessing. In: Deng, R.H., Gauthier-Umaña, V., Ochoa, M., Yung, M. (eds.) ACNS 2019. LNCS, vol. 11464, pp. 217–237. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-21568-2_11
21. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. In: 32nd International Conference on Machine Learning, ICML 2015 (2015)
22. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
23. Kingma, D.P., Welling, M.: Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114 (2013)
24. Li, H., Chen, M., Yan, S., Jia, C., Li, Z.: Password guessing via neural language modeling. In: Chen, X., Huang, X., Zhang, J. (eds.) ML4CS 2019. LNCS, vol. 11806, pp. 78–93. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30619-9_7
25. Melicher, W., et al.: Fast, lean, and accurate: modeling password guessability using neural networks. In: 25th USENIX Security Symposium (USENIX Security 2016), Austin, TX, pp. 175–191. USENIX Association, August 2016
26. Pasquini, D., Gangwal, A., Ateniese, G., Bernaschi, M., Conti, M.: Improving password guessing via representation learning. In: 42nd IEEE Symposium on Security and Privacy (Oakland) (2021)
27. Rabiner, L., Juang, B.: An introduction to hidden Markov models. IEEE ASSP Mag. **3**(1), 4–16 (1986)
28. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I.: Gpt2. Open AI (2019)
29. Sennrich, R., Haddow, B., Birch, A.: Neural machine translation of rare words with subword units. arXiv preprint arXiv:1508.07909 (2015)
30. Vaswani, A., et al.: Attention is all you need. In: Advances in Neural Information Processing Systems (2017)

31. Weir, M., Aggarwal, S., de Medeiros, B., Glodek, B.: Password cracking using probabilistic context-free grammars, pp. 391–405 (2009)
32. Zagoruyko, S., Komodakis, N.: Wide residual networks. In: British Machine Vision Conference 2016, BMVC 2016 (2016)