PGGAN: Improve Password Cover Rate Using the Controller

Xiaozhou Guo\textsuperscript{1,2}, Yi Liu\textsuperscript{12}, Kaijun Tan\textsuperscript{12}, Min Jin\textsuperscript{12} and Huaxiang Lu\textsuperscript{1234*}

\textsuperscript{1} Institute of Semiconductors, Chinese Academy of Sciences, Beijing 100083, China
\textsuperscript{2} University of Chinese Academy of Sciences, Beijing 100089, China
\textsuperscript{3} CAS Center for Excellence in Brain Science and Intelligence Technology, Chinese Academy of Sciences, Beijing 200031, China
\textsuperscript{4} Semiconductor Neural Network Intelligent Perception and Computing Technology Beijing Key Lab, Beijing 100083, China
*Corresponding author’s e-mail: luhx@semi.ac.cn

Abstract. Password generation model based on generative adversarial network usually has the problem of high duplicate rate, which further leads to low cover rate. In this regard, we propose PGGAN model. It sets up an additional controller network which is similar to the discriminator in the aspect of structure and function. The discriminator and the controller respectively learn the measure between the distribution of generated password with the real password distribution and the uniform distribution, and then use two measures to teach generator meanwhile. By changing the activation function and loss function of the controller, different measure functions can be selected. The experimental results show that compared with GAN, our PGGAN performs better both in cover rate and duplicate rate. Moreover, Wasserstein distance usually has a better effect to the other measure in model. Specifically, PGGAN with Wasserstein distance can increase the cover rate by 3.57\% and reduce the duplicate rate by 30.85\% on rockyou dataset.

1. Introduction
With the increasing demand for information security, password gradually becomes an effective form of authentication for people due to their ease of memory and use. Also, it is easier to deploy for the programmer, so password would remain one of the most important authentication methods in the future. Otherwise, there may be lots of vulnerability factors in the password\cite{1}, so it can lead to passwords being cracked by attackers.

Some password generative models are proposed to guess password. Early password generative models rely on heuristic rules set by experts, such as replacing lowercase letters in passwords with uppercase letters, changing words, but they have a limit in universality. Recently, models based on machine learning dominate the password guessing task. Markov model\cite{2} has advantages both in speed and effect. \cite{3} improves performance in terms of expression ability by expanding the model layer. \cite{4-5} both use the enumeration method and implement generation in descending order approximately. PCFG model based on statistical machine learning carries on deep excavation to the password pattern\cite{6}. It splits any password into letter, number, special character part, and \cite{7} conducts semantic mining on the letter string. \cite{8} adds personal information including birthday, email and telephone number, so that could crack more passwords.
Machine learning model based on neural network, also known as a deep learning model, has stronger representation ability. It has a wide range of applications in natural language processing[9]. [10] firstly tries to generate passwords using recurrent neural network framework. RNN generates a character at each time step and uses the generated character as the input for the next time step. [11] substitutes RNN with LSTM, which alleviates the problem of gradient disappearance. [12] proposes PG-RNN, and gets competitive results on multiple datasets. [13] introduces GAN(Generative Adversarial Network) to generate passwords first of all. PassGAN uses the one-hot vector to encode character, and then constructs the corresponding encoding matrix for each password. PassGAN directly captures the probability distribution of the password encoding matrix. When generating passwords, model just needs to provide the generator with Gaussian random noise.

GAN is powerful and potential for a variety of generation tasks, such as image[14], speech[15], video[16]. However, GAN usually suffers from the problem of mode collapse, which means that GAN produces a large number of duplicate samples. Accordingly, [17] expands the generator to enhance the ability to predict, so as avoid falling into the fixed patterns. In [18], for the discriminator, a gradient penalty term is imposed in the neighborhood of the training sample, so as to leave the local equilibrium point. [19-20] enhance generator’s ability by modifying network structure. Yet mode collapse has not been completely solved.

There are two reasons for GAN to generate a lot of duplicate passwords. For one reason, GAN in essence maps the normal distribution to the probability distribution of the password, so the generator needs to use the random sampling algorithm when generating samples, but this algorithm would inevitably produce duplication. For another reason, the problem of mode collapse would push the generator to usually generate some specific high probability passwords, for example 123456, while the other passwords are hard to arise. In addition, the number and proportion of duplicate passwords would both increase with the increase of generation number. Duplicate passwords not only waste resources, but also reduce the cover rate to the test set.

In order to reduce the duplicate rate and improve the cover rate, we propose the PGGAN model. When the probability of password is uniformly distributed, the duplicate rate can be minimized in expect. Therefore PGGAN not only makes generator learn the real probability distribution of the password, but also makes the learned distribution close to the uniform distribution. These above goals are in conflict with each other, so PGGAN should find a balance. Additionally PGGAN approaches the uniform distribution by using a network named controller, which is applied to learn the distance between probability distributions. We try a variety of measure distances including KL divergence, inverse KL divergence, JS divergence and Wasserstein distance. The experiment results in rockyou and CSDN dataset show that PGGAN has advantages in cover rate and duplicate rate.

The structure of the paper is as follows. First, GAN is introduced. Then network structure, loss function and training algorithm of PGGAN would be explained in detail in section 3. The experiment set and related results are discussed in section 4. Finally, some conclusions about the PGGAN model are covered in section 5.

2. GAN

![Figure 1. GAN model.](image)

GAN consists of two neural networks: discriminator $D$ and generator $G$, as shown in Figure 1. The essence of GAN is to make the probability distribution of the generative passwords $p_g(x)$ as close as possible to the probability distribution of the passwords in the training set $p_{data}(x)$. The function of the discriminator is similar to logistic regression. For any sample $x$, the discriminator will give the
probability $D(x)$ that the sample comes from the training set. Therefore, the perfect discriminator should output probability 1 for the sample from the training set and probability 0 for the sample generated by the generator. For the generator, it tries to capture the pattern in the training set, generates a sample and sends the sample to the discriminator. It is better to make the sample deceive the discriminator, so that the discriminator mistakenly thinks that the sample comes from the training set and outputs probability 1. Through this adversarial generation process, GAN does not explicitly model $p_{data}(x)$, but samples satisfying $p_{data}(x)$ can also be obtained.

In the discriminator $D$, it tries to learn the JS divergence between $p_g(x)$ and $p_{data}(x)$, so its loss function is
\[
\max_{\theta_D} E_{x \sim p_{data}} \log[D(x)] + E_{x \sim p_g} \log[1 - D(x)]
\]
(1)

Wherein $\theta_D$ represents the parameters in the discriminator. The generator wishes to narrow the learned distance, so its loss function is
\[
\min_{\theta_G} E_{z \sim p_z} \log[1 - D(G(z))]
\]
(2)

Wherein $\theta_G$ represents the parameters in the generator. Other GANs also follow the principle of discriminator learning distance and generator reducing distance, but they use different distances.

3. PGGAN

Figure 2. PGGAN model.

PGGAN is a deep generative model to generate passwords, which is composed of generator $G$, discriminator $D$ and controller $C$. The generator accepts Gaussian random noise $z$ as input, and then outputs the generated password $G(z)$. The discriminator takes the generated password or the real password of the training set as input, and outputs a scalar whose value is used to indicate differences of samples. We denote the probability distribution of passwords in the training set as $p_{data}(x)$. For the controller, it receives the generated password or the password in the uniform set as input, and also outputs a scalar value. We denote this distribution in the uniform set as $p_{uni}(x)$. In addition, the generated password also satisfies a probability distribution $p_g(x)$, which is defined implicitly by the generator. The PGGAN network is depicted in figure 2. The task of the discriminator is to drive the distribution $p_g(x)$ to approach $p_{data}(x)$, while the controller push $p_g(x)$ to the uniform distribution $p_{uni}(x)$.

In the discriminator of PGGAN, the network is mainly composed of ResBlock. In ResBlock, the backbone structure is ReLU activation-1D convolution-ReLU activation -1D convolution, and a short cut is added to prevent gradient from disappearing and increase the depth of network, as shown in Figure 3. First, the input password is operated by 1D convolution layer, then through 5 ResBlock in turn, and through a fully connected layer, eventually $D$ output a scalar value, which is shown in Figure 3. $D$ hopes to learn the Wasserstein distance between $p_{data}(x)$ and $p_g(x)$, so its loss function $L_D$ is
\[
\min_{\theta_D} -E_{x \sim p_{data}} [D(x)] + E_{x \sim p_g} [D(x)] + \lambda_p \left\| \nabla_D D(x_p) \right\| -1 \right|^2
\]
(3)

Where $\lambda_p$ is the coefficient of regularization term, and $x_p$ is a penalty sample which could be got by means of linear interpolation, for example $x_p = \alpha x_{data} + (1 - \alpha)x_g$. The regularization term makes the discriminator satisfy 1-Lipschitz restriction approximately.
For the controller $C$, the network structure is basically the same as the discriminator, as sketched in Figure 3. Here, the controller can choose different metrics to measure the distance between $p_{\text{true}}(x)$ and $p_{\text{gen}}(x)$, and using different metrics may need to adjust the activation function of the output layer. If using Wasserstein distance, the output layer does not need activation function, and loss function is similar to $D$

$$\min_{\theta_C} - \mathbb{E}_{x \sim p_{\text{true}}} [C(x)] + \mathbb{E}_{x \sim p_{\text{gen}}} [C(x)] + \lambda_p \left\| \nabla_x C(x_p) \right\| - 1 \right]^2$$  \hfill (4)

The coefficient of regularization term $\lambda_p$ and the method of acquiring penalty sample are the same as the discriminator. If using f-divergence, we could design different distance functions. For JS divergence, the activation function of the last layer is sigmoid: $f(x) = 1/(1 + e^{-x})$ and the loss function $L_C$ is

$$\min_{\theta_C} - \mathbb{E}_{x \sim p_{\text{true}}} \log[C(x)] + \mathbb{E}_{x \sim p_{\text{gen}}} \log[1 - C(x)]$$ \hfill (5)

For KL and JS mixing divergence (we call it the mixing divergence), the activation function is also sigmoid and the loss function is the same as above. For $\chi^2$ divergence, the last layer is the fully connected layer and does not need the activation function. Its loss function is

$$\min_{\theta_C} \mathbb{E}_{x \sim p_{\text{true}}} [C(x) - 1]^2 + \mathbb{E}_{x \sim p_{\text{gen}}} [C(x) + 1]^2$$ \hfill (6)

As to the generator $G$, ResBlock is still the most important part. For an input noise $z$, it first passes through the full connected layer, then $G$ send it to 5 ResBlocks successively. The sample is convoluted again and finally get generated password using softmax layer and argmax operation. $G$ has to reduce the two distance learned by $D$ and $C$ at the same time, so when $C$ uses Wasserstein distance its loss function is

$$\min_{\theta_G} \mathbb{E}_{x \sim p_{\text{z}}} [-D(G(z))] + \lambda_p \mathbb{E}_{x \sim p_{\text{z}}} [-C(G(z))]$$  \hfill (7)

where $\lambda_p$ is the equilibrium parameter. We can adjust the effect of the two tasks on the generator by controlling the value of the parameter $\lambda_p$. Larger parameter values mean that the generator tends to reduce the duplicate rate, while smaller parameter values make the generator more committed to fitting the true distribution of passwords.

If controller $C$ using JS, mixing or $\chi^2$ divergence, the corresponding loss functions $L_G$ are respectively

$$\min_{\theta_G} \mathbb{E}_{x \sim p_{\text{z}}} [-D(G(z))] + \lambda_p \mathbb{E}_{x \sim p_{\text{z}}} [-\log(1 - C(G(z)))]$$ \hfill (8)

$$\min_{\theta_G} \mathbb{E}_{x \sim p_{\text{z}}} [-D(G(z))] + \lambda_p \mathbb{E}_{x \sim p_{\text{z}}} [\log C(G(z))]$$ \hfill (9)

$$\min_{\theta_G} \mathbb{E}_{x \sim p_{\text{z}}} [-D(G(z))] + \lambda_p \mathbb{E}_{x \sim p_{\text{z}}} [C(G(z)) - 1]^2$$ \hfill (10)

The training algorithm of PGGAN is basically the same as GAN, except that the training of the controller is additionally added. The controller and the discriminator have a similar role, just the distance metric they need to learn is different. So when training PGGAN, we treat them in the same
way, which means they have the same training timing, number of training times and optimization
algorithm. On the one hand, only when the discriminator and controller learn the distance well, can
they provide the correct learning direction for the generator. On the other hand, if the discriminator
and controller are trained too well, the gradient of the generator may disappear. Therefore, in each
iteration, the discriminator and controller are trained \( n \) times and the generator is trained once. In
summary, the training algorithm is shown in Table 1.

4. Experiments
In order to verify the effect of PGGAN model, we choose rockyou and CSDN public data sets for
experiments. We use the first 80% of the data set as the training set and the last 20% as the test set. For
rockyou data set, it contains 21315685 passwords, of which 8274727 are unique passwords. The train
set contains 17052548 passwords, and 6909743 passwords are unique. The test set includes 4263137
passwords, and the number of unique passwords is 2174626. For CSDN data set, there are 4500371
passwords, and 2905686 passwords are unique. Its train set has 3600296 samples, of which 2352305
passwords are unique. In the test set, it includes 900075 passwords, while exists 655284 unique
passwords. Our experiment hardware environment is Xeon gold CPU, and the acceleration graphics
card used is 1080ti. The software environment is Ubuntu 14 operating system, python 3.6, and the
deep learning library is pytorch1.4.

| Table 1. PGGAN training algorithm |
|----------------------------------|
| **PGGAN training algorithm**      |
| while not converge:              |
| freeze the generator parameters \( \theta_G \) |
| for \( i=1:n_D \)              |
| train the discriminator by \( L_D \) |
| for \( i=1:n_C \)              |
| train the controller by \( L_C \) |
| freeze the discriminators parameters \( \theta_D \), the controller parameters \( \theta_C \) |
| train the generator by \( L_G \) |

Here we need to illustrate some super parameters and detail used in the experiment. The network
structure of the generator, discriminator and controller is shown in the third part. Each character in the
password is encoded in the form of one hot, and the length of the encoding vector is 70. The
dimension of the input Gaussian noise is 128, and in ResBlock, the dimension of feature map keeps
128. During the training process, the batch size is 128. The optimization algorithm of the generator,
discriminator and controller are Adam, and the learning rate is 0.0001, \( \beta_1 \) and \( \beta_2 \) are 0.5 and 0.9
respectively. The number of training times for the discriminator and controller (i.e. \( n_D \) and \( n_C \)) is 5.
Both in discriminator and controller’s loss function, regularization coefficient \( \lambda_p \) is 10, and in the
generator’s loss function, we set equilibrium parameter \( \lambda_p \) 0.1. We make random data set, which is
composed of two parts: one is the unique passwords in the public data set, such as rockyou, CSDN,
and the other is a randomly generated string set. In order to avoid using passwords of the test set for
training, we discard these passwords which exist in test set. Eventually, we get a random data set
including 500W nonduplicate passwords.

In the contrast experiment, the PassGAN model is the pure model without the controller. In
PGGAN model, KL divergence, mixing divergence, \( \chi^2 \) divergence and Wasserstein distance all are
tried. After the training, we use these models to generate 10\(^6\) passwords, and compare them from two
aspects of cover rate (\( CR \)) and duplicate rate (\( DR \)).

In the rockyou dataset whose results are shown in Table 2, compared with PassGAN, PGGAN
performs better both in cover rate and duplicate rate. In terms of cover rate, PGGAN increases 2.02%
on average. Among them, Wasserstein distance who has the best performance, improves 3.57%. For
the duplicate rate, PGGAN significantly decreases 22.73% averagely, and the biggest gap achieves
30.85%. In the CSDN dataset, PGGAN has less performance improvement which is depicted in Table
3. The average cover rate of PGGAN is increased by 1.22% and the duplicate rate is decreased by 17.33%. The biggest differences are 1.50% and 21.14% respectively. Totally, PGGAN model using Wasserstein distance in controller performs better than the other distance in whatever dataset. Experimental results show the effectiveness of our method. The introduction of the controller makes the generator not only learn the distribution of passwords, but also prevent it from falling into mode collapse. Thus, the generator not only learns how to generate passwords better, but also does not generate a large number of repeated passwords. The reduction of the duplicate rate gives the generator opportunity to try more unique passwords, thereby contributing to the improvement of the cover rate. From another point of view, uniform distribution can be considered as a kind of noise with a small amplitude, which can improve the diversity of training data and improve the generation effect. In addition, compared with KL divergence, mixing divergence and JS divergence, Wasserstein distance has better mathematical properties, for example its derivative can be obtained almost everywhere and the derivative is not zero, so it is better for training, and finally helps to advance the performance.

|                | PassGAN | JS divergence | mixing divergence | \(\chi^2\) divergence | Wasserstein distance |
|----------------|---------|---------------|-------------------|------------------------|---------------------|
| CR             | 5.75%   | 7.48%         | 6.23%             | 8.03%                  | 9.32%               |
| DR             | 88.17%  | 72.24%        | 69.55%            | 62.64%                 | 57.32%              |

Table 2. CR and DR in rockyou testset.

|                | PassGAN | KL divergence | mixing divergence | JS divergence | Wasserstein distance |
|----------------|---------|---------------|-------------------|---------------|---------------------|
| CR             | 1.06%   | 2.05%         | 2.49%             | 2.02%         | 2.56%               |
| DR             | 88.38%  | 75.65%        | 75.42%            | 65.24%        | 67.88%              |

Table 3. CR and DR in CSDN testset.

5. Conclusions
In this work, a more powerful password generative model PGGAN based on adversarial generative network has been implemented. PGGAN adds the controller which is used to learn the distance between password distribution and uniform distribution. The two measures learned by the discriminator and controller are used to optimize the generator, so that the problem of high duplicate rate can be alleviated, and the cover rate can be improved naturally. Experimental results on rockyou and CSDN datasets show the effectiveness of PGGAN. Using different metrics may lead to different degrees of performance improvement, and our experiments show that Wasserstein distance is the best. Since PGGAN model outperforms the traditional method, applying PGGAN as an efficient password guessing model is a promising research direction.

Acknowledgments
This work is supported by the CAS Strategic Leading Science and Technology Project (A class) Superconducting Computer Development XDA18040400 High Technology Project 31513070501, in part by National Natural Science Foundation of China U19A2080, in part by National Natural Science Foundation of China U1936106, in part by the Science and Technology Innovation Special Zone Project (1916312ZD00902201). This work is funded by the Beijing Academy of Artificial Intelligence, we would like to express our gratitude.

References
[1] Stobert E, Biddle R. The password life cycle: User behavior in managing passwords[C] //Proc of SOUPS2014. New York: ACM, 2014: 243-255
[2] Narayanan A, Shmatikov V. Fast dictionary attacks on passwords using time-space trade-off[C]//Proceedings of the 12th ACM Conference on Computer and Communications Security Alexandria, USA, 2005: 364-372.

[3] Tansey W. Improved models for password guessing[R]. 2011.

[4] Durmuth M, Angelstorf F. OMEN: Faster password guessing using an ordered markov enumerator[C]//Proceedings of International Symposium on Engineering Secure Software and Systems, Milan, Italy, 2015: 119-132.

[5] Wang D, Zhang Z, Wang P, et al. Targeted online password guessing: An underestimated threat[C]//Proc of ACM CSS 2016, New York, USA, 2016: 1-13.

[6] Weir M, Aggarwal S, Medeiros B D, et al. Password cracking using probabilistic context-free grammars[C]//Proceedings of the IEEE Symposium on Security and Privacy, Oakland, USA, 2009: 391-405.

[7] Veras R, Collins C, Thrope J. On the semantic patterns of passwords and their security impact[C]//Proceedings of the 21st Network and Distributed System Security Symposium, San Diego, USA, 2014: 1-16.

[8] Li, Yue, Haining Wang, and Kun Sun. "A study of personal information in human-chosen passwords and its security implications." IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications. IEEE, 2016.

[9] Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems. 2017.

[10] Melicher W, Ur B, Segreti S M, et al. Fast, lean, and accurate: Modeling password guessability using neural networks[C]//Proc of USENIX Security, 2016.

[11] Xu, Lingzhi, et al. "Password guessing based on LSTM recurrent neural networks." 2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC). Vol. 1. IEEE, 2017.

[12] Nanjun T, Huaxiang L U, Min J, et al. PG-RNN: a password-guessing model based on recurrent neural networks[J]. CAAI Transactions on Intelligent Systems, 2018.

[13] Goodfellow I J, Pouget-Abadie J, Mirza M, et al. Generative Adversarial Networks[J]. Advances in Neural Information Processing Systems, 2014, 3:2672-2680.

[14] Karras, Tero, et al. "Analyzing and improving the image quality of stylegan." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2020.

[15] Pascual, Santiago, Antonio Bonafonte, and Joan Serra. "SEGAN: Speech enhancement generative adversarial network." arXiv preprint arXiv:1703.09452 (2017).

[16] Liang, Xiaodan, et al. "Dual motion GAN for future-flow embedded video prediction." Proceedings of the IEEE International Conference on Computer Vision. 2017.

[17] Metz, Luke, et al. "Unrolled generative adversarial networks." arXiv preprint arXiv:1611.02163 (2016).

[18] Kodali, Naveen, et al. "On convergence and stability of gans." arXiv preprint arXiv:1705.07215 (2017).

[19] Hardy, Corentin, Erwan Le Merrer, and Bruno Sericola. "Md-gan: Multi-discriminator generative adversarial networks for distributed datasets." 2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS). IEEE, 2019.

[20] Xian, Yongqin, et al. "f-VAEGAN-D2: A feature generating framework for any-shot learning." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2019.