Research and Development Project Report

Plausible password generation using generative models

Animesh Bohara (160050040)
Eashan Gupta (160050045)

Guide: Prof. Abir De

Abstract

There has been considerable work done on password guessing, i.e. given a dataset of passwords used by people, to train a model to guess a password in minimum number of tries. There are also various metrics used to evaluate a password for its security. In this project, we attempt the problem of password generation. Any generated password that our model outputs should be easily memorable, but not easily guessable. For this purpose, we first come up with suitable definitions for memorability and guessability and provide justification for the same. Then we also try different models and evaluate them based on the above metrics.

1 Introduction

A preferable password for any user should be sensible enough for them to remember easily while also being strong enough so that others might not be able to guess it easily. The various metrics derived so far to evaluate passwords only measure their strength. Furthermore, these methods are mostly derived using models used to crack passwords. In this project, we first try to evaluate a password in terms of the ease of remembering it- *memorability*- along with its strength-*guessabilty*. We believe that evaluating a password based on these metrics should be sufficient. Next, we attempt the problem of password generation to generate new memorable as well as not so easily guessable passwords. We use majorly three methods to explore the problem- recurrent neural networks, variational autoencoders and grammar variational autoencoders.

2 Related Work

Following is a survey based on earlier literature on the various metrics to evaluate a password given some data. We also give a brief overview of the methods used.

2.1 How Does Your Password Measure Up? The Effect of Strength Meters on Password Creation

In [Ur et al., 2012], the authors undertake an empirical discussion of password features (number of uppercase characters, digits, symbols, etc.) v/s the visual cues and feedback provided during password creation. The statistical differences between control groups are measured using p-values, Kruskal-Wallis test and Mann-Whitney U test.

To calculate the *guessability* of a set of passwords, they simulate an attacker in 3 different scenarios (number of guesses attacker can make - 500 million, 50 billion and 5 trillion). Comparisons are made on the percentage of passwords guessed in a dataset (control group).

For the case of *memorability*, they don't provide a quantification. Instead, they observe that there is no significant difference in the memorability of the password even if the user logs in for the second time, seconds after the first login, or 2 days after the first login.

2.2 Measuring Password Guessability for an Entire University

In [Mazurek et al., 2013], the authors access the existing passwords of CMU's students, faculty and staff securely.

First they assign guess numbers to each password using modified *Weir's algorithm*, which relies on password cracking using a probabilistic context-free grammar. Then, they apply cox regression on password guessability to get relations between different factors (like number of digits, lowercase alphabets, etc.) and guessability.

2.3 Measuring Real-World Accuracies and Biases in Modeling Password Guessability

The authors in [Ur et al., 2015] work on measuring guessability more effectively. They claim that using only guessability from only one password-cracking tool is insufficient. Instead, we should use different tools along with custom configurations in parallel.

2.4 Fast, Lean, and Accurate: Modeling Password Guessability Using Neural Networks

Guessability measurement using guessing tools like John the Ripper, Hashcat, etc. takes a large amount of time. The authors in [Melicher et al., 2016] develop a RNN-based approach which is sufficiently low latency and lightweight to be run client-side, for e.g. in a browser. They use transference learning during training the neural network. They also make use of Monte Carlo simulations to estimate the guess numbers of passwords.

2.5 Targeted Online Password Guessing: An Underestimated Threat

In [Wang et al., 2016], the authors use PII (personally identifiable information)- information of person from different accounts- to guess passwords of a particular user. They classify the user information and develop methods to guess passwords based on the data available for each class. The paper in particular describes four algorithms to guess passwords given a target.

First, given minimal information (name, age) they divide passwords into tags L(letter), D(digit), B(birthday), N(name) etc. Then, a CFG for language of passwords is constructed using these, also known as Probabilistic CFG. Another algorithm as described in the paper uses sister passwords from sister sites to guess password at another site. Other users' passwords from both sides are used and matched to create a map. Then modifications via both PCFG and structure-level rules are studied. A third algorithm combines the information and methods from the two to guess passwords. The fourth algorithm uses all the information available. It uses Bayesian models to compute probability of selecting a password and estimate its probability.

These methods have not been used by us so far in the project but may be used later to model guessability, and to study how PII is stored and used.

2.6 Beyond Credential Stuffing: Password Similarity Models using Neural Networks

In [Pal et al., 2019], the authors work on generating similar new passwords for a user given at least one of their other passwords. The generated passwords are used for attacks to empirically evaluate the performance of the model. The authors use various methods to generate new passwords given leaked passwords.

First they use *sequence-to-sequence* (seq2seq) algorithms such as those used in NLP to generate a new password. This method, however, does not outperform previous approaches to the problem.

Next they describe a *password-to-path* (pass2path) algorithm which tries to predict the modifications done by users to their passwords to generate other passwords. The model uses three transformations- insertion, deletion and substitution of characters- among passwords and the number of transformations taken is the edit distance between the two passwords. The authors model similarity between passwords w and $\bar{w}$ as the conditional probability $P(w|\bar{w})$ of user choosing the password w given a previous password $\bar{w}$. An encoder-decoder model is used to capture the above probability and then used to generate passwords with lower edit distance. This approach performs well on the dataset (with 2 passwords of 1.4 billion people) used.

The authors also use the above model to introduce personalized password strength meters (PPSMs) to measure security of passwords given previously compromised passwords. These are defined using password similarity via embeddings.

2.7 Reasoning Analytically About Password-Cracking Software

Most metrics measuring guessability of a password effectively count the number of tries required to guess by the trained model. These models take a lot of time to train and generate a probabilistic model which is then used to generate passwords and try them. In [Liu et al., 2019], the authors remark that real-world hackers seldom use such time-consuming methods and instead use tools like John the Ripper and Hashcat. The authors then reason analytically about such popular password cracking tools to give methods to efficiently measure the strength of a password instead of running the tools to get the number of tries. First, they propose an analytical approach to modeling transformation-based password guessing. Password cracking tools leverage the insight that passwords tend to differ in small and predictable ways. Thus, the authors use methods such as rule inversion- to invert a rule on the target password- and guess counting- to approximate the number of guesses required and get an order of magnitude- to measure the strength of the password. Second, they provide optimisation algorithms for the tools to generate the passwords more efficiently.

3 Our work

In this project, we remark that a "good" password should not be easily guessable but still be very memorable. Thus, we first describe our interpretations/definitions of the two metrics.

Following that are some models which we have experimented on to generate passwords. We have tried different approaches followed by evaluations of these approaches based on the metrics.

3.1 Datasets

3.1.1 Rockyou dataset

This is a list of cleartext passwords ordered by their frequency, which was obtained by a data breach of the company *RockYou* in 2009. It contains 14.3 million unique passwords.

3.1.2 1.4 Billion Password database

This is a database of 1.4 billion cleartext passwords found on the dark web. The sources of these passwords are unclear. Some of the passwords in this database are still in use by people.

3.2 Metrics

3.2.1 Memorability

The memorability score of a word w is calculate as follows. Let $\mathcal{V}$ be the vocabulary (Top 100k English words in our case).

$$\begin{aligned} v &= \underset{v \in \mathcal{V}}{\operatorname{argmin}}\, editDistance(v, w) \\ memorability &= 1 - \frac{editDistance(v, w)}{min(len(v), len(w))} \end{aligned}$$

Here $editDistance(a, b)$ denotes the minimum number of operations needed to go from the word a to the word b. An operation can consist of addition, deletion or substitution of a character. The intuition for the above score is that English words are easily memorable, and the more we deviate from these, the less memorable our password becomes. The second step is normalizing the score, so that the range of the guessability and memorability scores remain same ($[0, 1]$).

Though this definition of memorablity is justifiable, it is still imperfect and has scope for improvement. First, the above score is highly dependent on the dictionary of popular words used. Thus, popular words

such as proper nouns may be missed out. It is also required that the dictionary be constantly updated to keep up with the trends. Second, this definition does not take care of patterns e.g. "12345678", "111111", "abcdef", "asdfgh", "password@123" etc. Such patterns may not be included in the dictionary but are still very memorable. Third, the above method also does not take care of phrases comprised of multiple words and popular abbreviations e.g. "ILoveU", "ILoveYou" etc. Fourth, the method is case insensitive and it is not defined if "password" is more memorable than "PaSsWoRd".

Future work on this metric may be to take care of the above problems. One method may be to add more patterns in the dictionary and not just popular words. Another would be to use the average edit distance of N nearest neighbours for the score instead of the minimum edit distance.

3.2.2 Guessability

We use the guessability score from the meter by [Ur et al., 2017]. We modify it slightly to accept the input password as a GET parameter, and call this HTML page from a Python script.

The authors in [Ur et al., 2017], use a neural network to model the number of guesses. In addition, they also use pre-defined heuristics which have performed well empirically to evaluate password strength. We use this meter as our "un"-guessability score as stronger the password, less likely it is to be guessed. It may be possible to find better meters but for the course of this project, we have used this as our baseline.

3.3 Next character prediction using RNNs

If we view the problem at a character level, we can go around generating the password character-by-character. RNNs have traditionally been used for this purpose. Given the characters of a word $c_1, c_2, ...c_k$, we want to derive a probability distribution for c_{k+1}, over the vocabulary (which we have taken as the ASCII character set in our case, along with two $[START]$ and $[END]$ tokens). During the generation phase in the model, we sample a character from this distribution, appending it to the word until we encounter an $[END]$ token. The first character of a word is always fixed as the $[START]$ token. The RNN we used in our case is LSTM. While LSTM may be an overkill for our problem, but it converges faster than SimpleRNN and GRU, despite having more parameters to train. Training is done on a subset of the Rockyou dataset.

3.3.1 Training and Generation

A $[START]$ token is added at the start of the word. Then the word is padded with $[END]$ tokens at the end till the length of the word becomes equal to MAXLEN (50 in our case). Then the characters of the word are one-hot encoded. This is the input to our model. The model consists of an LSTM layer, followed by a Softmax of the outputs, which is treated as a probability distribution. The loss is computed as the *categorical crossentropy* of this output distribution with input left-shifted by 1 position.

In the generation phase, a *temperature* parameter is used to control the randomness of the output. The output of the LSTM layer, called as *logits*, are divided by *temperature*, before the Softmax layer.

In Figure 1, we observe that the memorability of the generated words decreases on average with increasing temperature. This might be due to the fact that more temperature brings more randomness to the output, making it less memorable. But an interesting observation is the guessability remains almost the same at any temperature. This means that a model with smaller temperature is better because of the higher memorability. But in case of smaller temperatures, because the randomness is less, we get many outputs which are present in the training set, as seen in Figure 2.

3.4 Variational Recurrent Autoencoder

In the previous LSTM model, we attempted to generate the word, one character at a time. Instead, we could view the problem as being given a dataset of passwords, which come from some distribution, and we have to generate a new password from the same distribution. Generative models are used for exactly this purpose. In the case of the Rockyou dataset, the distribution has the property of having high memorability scores. This is because the passwords in Rockyou dataset are ordered by frequency, and thus are highly memorable.

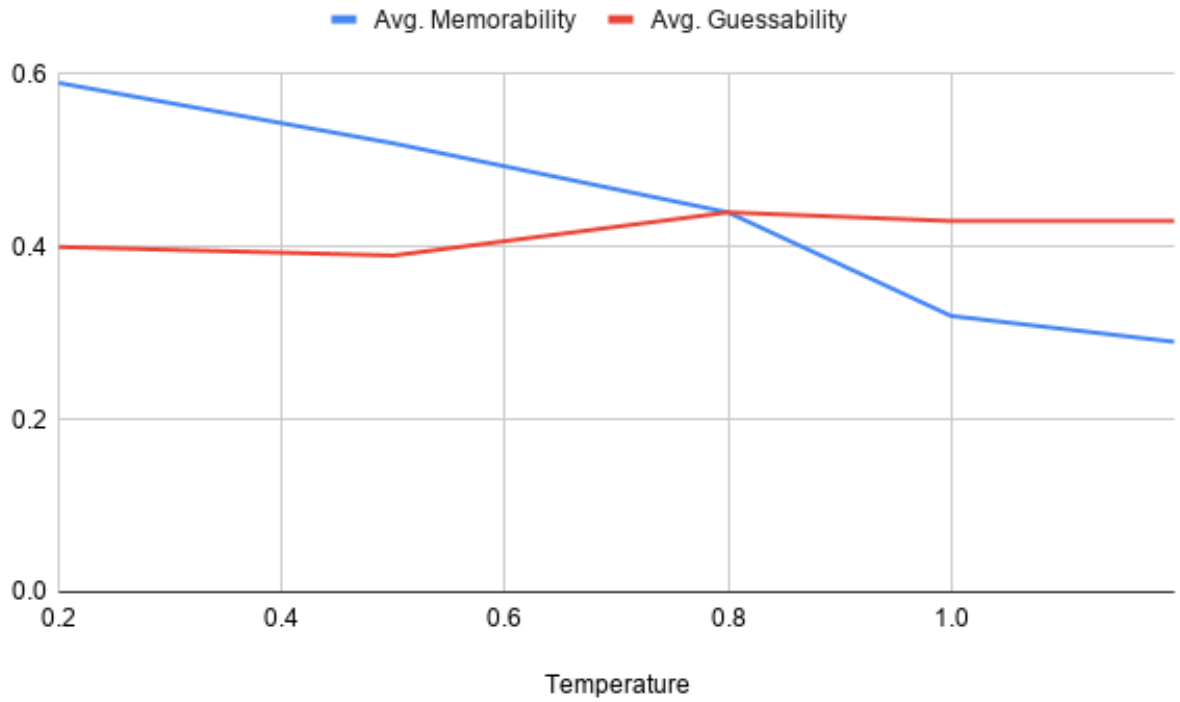

Figure 1: Average Memorability and Guessability v/s Temperature for LSTM model

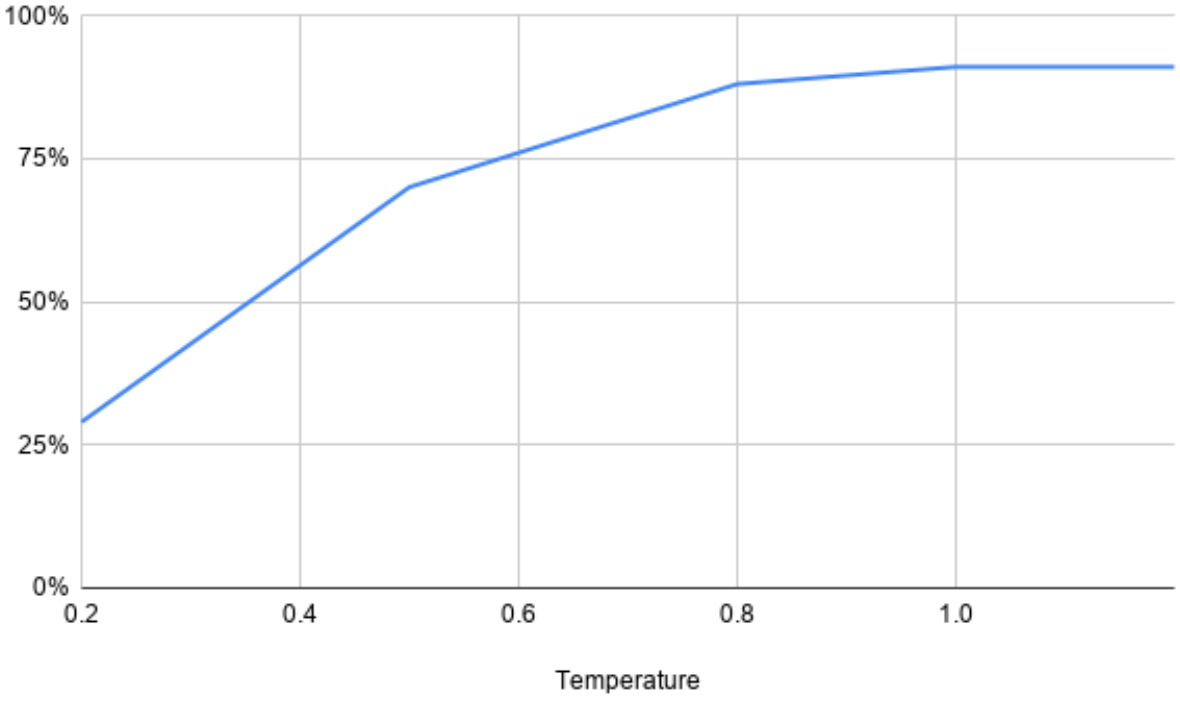

Figure 2: The percentage of generated passwords not in training set v/s Temperature

3.4.1 Training and Generation

An RNN is used as the encoder and decoder in the autoencoder, as it summarizes the input as its internal state (encoder), and then is able to reconstruct a variable length input word given an initial state input (decoder). The type of RNN used is GRU because of slightly lesser number of parameters than LSTM. The input provided is similar to the previous model.

Additionally, an input mask is applied to the encoder to consider the hidden state outputs until the first $[END]$ token is encountered. The application of such a mask is necessary because otherwise during backpropogation, our RNN suffers from the *Vanishing Gradient* problem. The final hidden state of the encoder is passed through linear layers to get the mean and standard deviation of the latent variable. Then a latent variable is sampled from the gaussian distribution described by the above mean and standard deviation using the "reparametrization trick". This sampled latent variable is passed as input to the decoder. The decoder additionally takes as input the previous character output by itself. In case of training, this is the actual previous character in the word.

The loss is taken as the reconstruction loss along with the KL divergence of the distribution of the latent variable and the latent prior, which is assumed to be standard gaussian. In standard VAE models, during the generation phase the latent variable is sampled from the prior, and passed as input to the decoder. We got better results by sampling two random inputs w_1, w_2 from the training dataset, sampling latent variables z_1, z_2 from the distribution described by the encoder outputs of w_1 and w_2, and passing the latent variable $z = 0.5z_1 + 0.5z_2$ to the decoder.

During generation, the output of the decoder at each step is taken as the character with the maximum probability. While this is not an accurate method as we want the word which has the highest probability given a particular state, this is a good preliminary heuristic. In future, we can try using a *k-beam search* heuristic for the same.

3.5 Grammar Variational Autoencoder

Variational autoencoders have been widely used for generative modelling. But generative modeling of discrete data such as arithmetic expressions and molecular structures still poses significant challenges for VAEs. [Kusner et al., 2017] use Grammar VAE to model such structures based on the observation that discrete data can be represented as a parse tree from a context-free grammar.

We try to use similar modeling methods as Grammar VAE to generate passwords. Based on the previous background experience, we observe that passwords are generally modifications of words and it may be possible to capture these modification patterns. We also believe that such modification patterns may be well represented in a context-free grammar. Hence, we attempt this approach.

In this VAE model, we first fix a context-free grammar G to represent passwords. This grammar is unambiguous and covers the entire space of possible passwords. Any password P is first converted to its parse tree based on G. Then by an infix traversal over the tree, we get a unique order of production rule applications from G. Note that if the start symbol of G is applied with this order of applications, it would again generate P. Next, this order list is one-hot encoded based on all production rules. This one-hot encoding is then passed to a Variational Autoencoder to learn the rule orderings. The output from the decoder is a probability distribution over production rules which is then used to produce a valid ordering of rules using masks. Applying this ordering on start symbol of G gives a new generated password.

3.5.1 Training and Generation

For training our model, we used grammar G with start symbol $Smile$:

```
Smile -> Chain
Chain -> Chain BranchedAtom
Chain -> BranchedAtom
BranchedAtom -> Letter
BranchedAtom -> Digit
BranchedAtom -> SpecialChar
Letter -> Lower
```

```
Letter -> Upper
Nothing -> None
Digit -> '0' | '1' | ... | '9'
Lower -> 'a' | 'b' | ... | 'z'
Upper -> 'A' | 'B' | ... | 'Z'
SpecialChar -> '@' | ... | '~'
```

The training was done on a subset of Rockyou dataset. First the passwords were encoded to one-hot vectors. The loss used was a sum of the binary cross-entropy loss and the KL divergence. We trained various encoders and decoders with the dataset :

1. The first model used for training used 3 Convolution layers with ReLU activation, followed by a Dense layer to calculate the mean and variance. The decoder used a Dense layer followed by 3 GRU layers and then valid orderings are sampled using masks. This was the exact model as trained by thhe authors in [Kusner et al., 2017].

2. We simplified the above model and used only one GRU layer in the encoder followed by Dense layer. The decoder also used a Dense layer followed by one GRU layer. This was done based on the assumption that a GRU in encoder might help us capture the order of production rules used.

3. We further experimented on models by adding Convolution layers and input mask in GRU to the above model.

3.5.2 Observations and Conclusion

Figure 3 shows a comparison of scores for various models based on Grammar VAE trained. Following the above experiments, we observe that the models start generating passwords which have more digits in them. This is particularly so as we increase the latent dimensions. Thus, the memorability scores of such samples are also very low. As shown in Figure 3, higher latent dimensions have very low memorability. The guessability scores are better than those generated in earlier models. We believe that this increased occurrences of digits may be due to our choice of grammar G. In G, the number of rules "Digits -> '0' | '1' | ... | '9' " are lower than others thus they might be getting more probabilities and appearing more. The results improve with higher latent dimensions but there are still samples with a lot of repetition of characters. The best results are from the model which has both GRU and Convolution layers in the encoder along wih use of masks for the GRU.

For future work, we believe that there is a lot of scope for improvement in Grammar VAE models used to generate passwords. First, we have only used a simple naive grammar for the tests. We believe that if we use a grammar with more patterns and rules following popular heuristics, we may be able to capture them more effectively. An example would be to have phrases, words and letters, all as terminals of the CFG instead of only letters. Though, there would still be a problem to determine rules in case of ambiguous grammar. Second, changing model parameters and structure might also be helpful. Changing the latent dimensions and other parameters might help as well.

4 Conclusion

Table 1 shows a comparison of some of the models and methods experimented in this project.

The average memorability scores of VAE are higher, and with a higher standard deviation, which shows that the VAE model consistently outputs more passwords with good memorability scores than the LSTM model, while the guessability score of the VAE model is lower than that of the LSTM model. This can be attributed to the training dataset we are using. The training dataset used in both models is the Rockyou dataset, which contains password with high memorability scores, and not so good guessability scores. As the VAE model outputs a password from this distribution, it is natural that the generated password will be expected to have a good memorability and not-so-good guessability score.

Compared to these, the Grammar VAE model generate passwords with high guessability scores i.e. they are not easily guessed. Consequently, they also have lower memorability. Additionally, by observing the

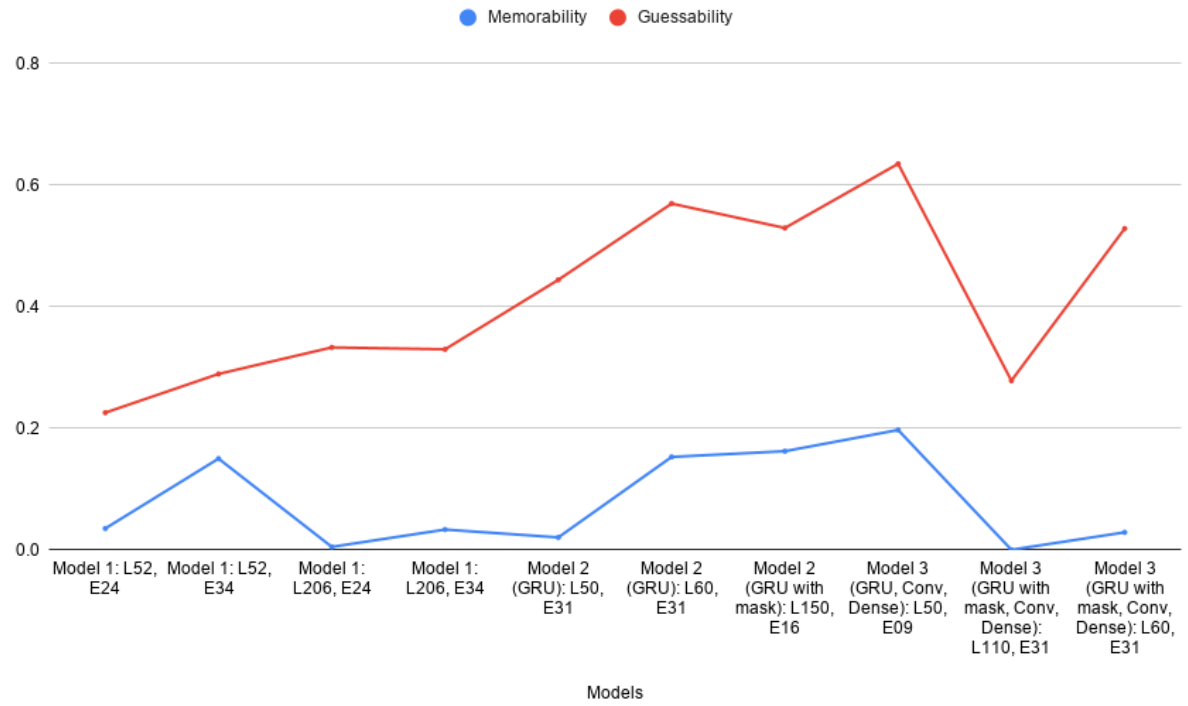

Figure 3: Average Memorability and Guessability for various Grammar VAE models. LX, where X is the latent dimension used. EY, where Y is the number of epochs trained. Follow the model number from 3.5.1. The deep NN layers mentioned in brackets are the NN layers used in encoder.

Model	**Details**	**Epochs**	**Training Dataset size**	**Average Memorability**	**Average Guessability**	**Percent passwords not in training set**
LSTM	Temperature = 0.2	15	10 million	0.59 ± 0.15	0.40 ± 0.05	29
VAE	GRU encoder with masking, GRU decoder	600	10000	0.66 ± 0.25	0.30 ± 0.03	30
Grammar VAE	Model 3 (GRU with mask): Latent Dimension = 150	16	100000	0.162 ± 0.191	0.530 ± 0.088	0
Grammar VAE	Model 3 (GRU, Conv, Dense): Latent Dimension = 50	9	100000	0.197 ± 0.232	0.634 ± 0.289	0

Table 1: Average Scores of sample passwords generated by various models with different parameters

samples generated, it seems that the passwords tend to have more occurrences of digits in some cases which can be attributed to the context-free grammar being used. Thus, the model may be useful in generating unique stronger passwords.

5 Future Work

Apart from training and testing the above models with different hyperparameters, we can try some more methods. *Reinforcement Learaning* methods such as *Monte Carlo Tree Search* or hybrid methods such as *Reward Augmented Maximum Likelihood* estimator can be tried. The guessability and memorability can also be modelled by neural networks, and this can be used as the discriminator network in a *GAN*.

References

Matt J Kusner, Brooks Paige, and José Miguel Hernández-Lobato. Grammar variational autoencoder. *arXiv preprint arXiv:1703.01925*, 2017.

Enze Liu, Amanda Nakanishi, Maximilian Golla, David Cash, and Blase Ur. Reasoning analytically about password-cracking software. pages 380–397, 05 2019. doi: 10.1109/SP.2019.00070.

Michelle L Mazurek, Saranga Komanduri, Timothy Vidas, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, Patrick Gage Kelley, Richard Shay, and Blase Ur. Measuring password guessability for an entire university. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 173–186, 2013.

William Melicher, Blase Ur, Sean M Segreti, Saranga Komanduri, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. Fast, lean, and accurate: Modeling password guessability using neural networks. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 175–191, 2016.

B. Pal, T. Daniel, R. Chatterjee, and T. Ristenpart. Beyond credential stuffing: Password similarity models using neural networks. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 417–434, 2019.

Blase Ur, Patrick Gage Kelley, Saranga Komanduri, Joel Lee, Michael Maass, Michelle L. Mazurek, Timothy Passaro, Richard Shay, Timothy Vidas, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. How does your password measure up? the effect of strength meters on password creation. In *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*, pages 65–80, Bellevue, WA, 2012. USENIX. ISBN 978-931971-95-9. URL https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/ur.

Blase Ur, Sean M. Segreti, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, Saranga Komanduri, Darya Kurilova, Michelle L. Mazurek, William Melicher, and Richard Shay. Measuring real-world accuracies and biases in modeling password guessability. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 463–481, Washington, D.C., August 2015. USENIX Association. ISBN 978-1-939133-11-3. URL https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/ur.

Blase Ur, Felicia Alfieri, Maung Aung, Lujo Bauer, Nicolas Christin, Jessica Colnago, Lorrie Faith Cranor, Henry Dixon, Pardis Emami Naeini, Hana Habib, Noah Johnson, and William Melicher. Design and evaluation of a data-driven password meter. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, page 3775–3786, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450346559. doi: 10.1145/3025453.3026050. URL https://doi.org/10.1145/3025453.3026050.

Ding Wang, Zijian Zhang, Ping Wang, Jeff Yan, and Xinyi Huang. Targeted online password guessing: An underestimated threat. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 1242–1254, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450341394. doi: 10.1145/2976749.2978339. URL https://doi.org/10.1145/2976749.2978339.