On using Transformer as Password Guessing Attacker

Simo Ryu
cloneofsimo@korea.ac.kr
학번 : 2019330001 이름 : 류시모

October 2020

Abstract

In this paper, we show one can perform password guessing attacks with the Transformer model, with small number of parameters and training time. We study on the effect of temperature on generative stage, and how pre-training impacts generalization ability of the model. Source code for this project is available at https://github.com/cloneofsimo/PGAT.

1 Introduction

If an adversary wishes to break into an encrypted database, numerous cracking methods can be performed. As passwords are one of the standardized, de-facto method for authentication, brute-forcing every possible password candidates to break through authentication can be one of the promising methods. However, since ideal cardinality of random string is infeasible large, many state-of-the-art methods to produce candidates try to sample from the true distribution of password space. As multiple leaks of password databases suggested that cardinality of passwords tends to be orders of magnitude smaller than just random strings. One explanation is that a password users choose is often combinations of characters that are easy to remember. Tools such as HashCat [6] and John the Ripper [10] therefore tries to mimic the distribution with policy-based string generation, such as concatenation, random mutation of characters, and markov-chain based events. Other methods are also studied and implemented widely.

As neural network proved its strong performance in multiple tasks, novel approaches such as PassGAN [9] had been developed to produce password guesses based on deep

learning and Generative Adversarial Networks (GAN). Their work proved that deep learning based methods are promising approaches for password guessing. However as their work rely purely on password datasets this, which we later argue, may be idealistic.

On the other side, significant improvements had been developed on wide range of NLP tasks. This had been mainly due to incredibly large pre-trained language models. As pre-trained language models showed various state-of-the-art results, recent work by Brown et al. [3] introduced GPT-3, huge pre-trained LM with 175 billion parameters. Their work showed that very large language model is capable of few-shot learning. Another recent work by T.Schick and H. Schutze [18] has also demonstrated that small LMs are also capable of few-shot learning, when designed with capability of rapid exploitation of linguistic patterns.

Building on the idea that pre-trained LM being capable of multiple, wide ranging tasks, we investigate on the capabilities of pre-trained LM as a password guesser. Also, we will have a close look at the effect of temperature parameter during password generation. Lastly, we will handle how pre-training stage can effect the generalization ability of the Transformer.

2 Related Works

2.1 Pre-training for Language Tasks

Feature embedding based methods, both neural and non-neural methods had been widely studied.

Perhaps one of the most famous pre-training method for language tasks started from work of Brown et al. [2], is word embedding. On many modern NLP systems, using such word embedding is largely beneficial , rather than learning from scratch[21]. Much the same idea had been generalized to sentence embeddings [11] and paragraph embeddings [12].

On top of feature-embedding based methods, many works attempted self-supervised learning on NLP systems : pre-trained on language corpus and then fine-tuned for specific tasks. One notable example of this work would be by Howard and Ruder [11], which leverages pre-training and fine tuning for text classification tasks.

Yet, recent works with transformer based pre-training methods had achieved the most successful cases for transfer learning on NLP tasks [5, 3, 17].

2.2 Transformer

Transformers were first proposed by Vaswani et al. [22] as architecture for machine translation. When Transformers appeared, transfer-learning on many NLP systems became much more robust and universal. From then, large Transformer-based language models had been frequently used with pre-training on large corpus dataset, then fine-tuned on specific tasks. Consequently large LMs had proved themselves as many state-of-the-art architecture. As mentioned above, GPT-3 is also a decoder - only Transformer [3], and is also pre-trained on large language corpus. Another notable model is BERT [5], which is essentially pre-trained model achieved by denoising self-supervision task.

Very recent work [1] suggested that pre-trained transformers are also capable of vision classification tasks. Authors showed that treating one image as patches of small images, and simply pre-training standard transformer architecture is enough to attain near state-of-the-art results on various classification tasks.

To briefly explain the mechanics of the Transformer, it consists of Multi-Head Self-Attention (MHA), that allows the model to obtain information on multiple representational, positional subspaces. Generally, MHA is defined as

$$\text{MHA}(Q, K, V) = Concat(\text{head}_1, \text{head}_2, \text{head}_3, ...\text{head}_h)W^O$$

where $Q, K, V \in \mathbb{R}^{t \times d_m}$ are embedding matrices. Each head is defined as:

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) = \text{softmax}[\frac{QW_i^Q(KW_i^K)^T}{\sqrt{d_m}}]VW_i^V$$

where $W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{d_m \times d}, W^O \in \mathbb{R}^{hd \times d_m}$ are learned parameters. If one wishes to make Masked-self-attention, we treat Q, K, V with no difference, i.e., $Q = K = V$. As we wish to use masked-self attention, for the rest of the paper there will be no distinguishment of embedded matrices Q, K, V.

2.3 Non-Neural Password Guessing Attackers

Even though password authentication is de-facto standard method of authentication method, there are numerous situations where password guessing attacks can be a threat. Mainly, until this day there are many cases where hashed database of passwords are leaked, in which case adversary can sample top-k most likely candidates of password guesses, hashes them to check with the hashed database. There are also legal cases where password guessing is necessary. Multiple approaches, both neural base and non-neural based, had been developed and studied for password guessing attacks.

2.3.1 Markov Based Models

First proposed by [16] Markov model tries to predict the probability of the next character in a password based on the previous characters. Researchers found that using 6-gram Markov model with additive smoothing is often optimal for English-language passwords [14].

2.3.2 Mangled Wordlist Methods

The most widely used method is mangled wordlist methods. The method consists of transforming an element from a set of word (dictionary) into a password by set of "mangling rules". Mangling rules are then designed with hand, such as changing character 'e' into '3' or, concatenating two strings randomly. Even though there are no statistical learning element in this method, in many cases this produced great password guesses, which made HashCat [6] and John The Ripper [10] widely popular.

2.4 Neural Password Guessing Attackers

After various promising results regarding neural networks have published, there had been many researches attempting to model the password spaces via neural networks. Few works consider regression : where neural network tries to evaluate the strength of the password [15] , while others, explores in the direction of this paper, tries to generate password guesses explicitly.

2.4.1 Neural Networks as Password Strength Evaluators

Notable first attempt to leverage the power of neural network to model password-strength is from Ciaramella et al. [4].Novel works such as Melicher et al. [15] tries similar approach for similar tasks, but focuses more on making the model light-weight to be able to deploy on browsers.

2.4.2 Neural Networks as Password Guessing Attackers

On the other hand, some researches to explicitly use neural networks for password guessing attacks had been around. Hitaj et al.[9] proposed PassGAN, and argued that such direction of using neural network could be novel approach for password-guessing attacks. However, many of these approaches attempts without prior knowledge of user's information, which is not ideal if there are any that can be robustly used for such task. Accordingly, recent work by Zhou et al. [24] attempts to tackle this unused stack of

Model Name	n_{params}	n_{layers}	d_{model}	d_{ff}	n_{heads}	Learning Rate
Small	0.62M	3	128	256	4	2.0×10^{-4}
Medium	3.21M	4	256	512	4	9.0×10^{-5}
Large	15.87M	5	512	1024	4	3.0×10^{-5}

Table 1: **Model hyperparameters, learning rate, batch size during training**

information on the users, such as nickname, birthday, phone number, etc. They proposed password guessing model TPGXNN that combines user's personal information to generate relevant password guesses. More recent work by He et al. [8] uses transformer to tackle the same problem. Despite using the transformer as main architecture, our proposal is a single decoder-only Transformer model, that is independent of user's information, and tries to mimic the distribution of passwords.

3 Transformers are Good Password Guessing Attackers

We first explore rather it is possible to use Transformer model to deploy password-guessing attacks.

3.1 Model Architecture

We use the Decoder-Only-Transformer model similar to that of proposed by [17]. We tried to keep every condition similar, including Layer Normalization, dropout and positional embedding. To understand how the model size effect the quality and efficiency of the model, we prepared three different models, Small, Medium and Large. Table 1 introduces specific parameters for model architecture. Here, n_{params} is number of total parameters, n_{layers} is number of MHA layers in the transformer layer, d_{model} is the number of units in each bottleneck layer, d_{ff} is number of units in feed-forward layer after MHA layer, n_{heads} is number of self-attention heads in each MHA layers.

One thing to consider is that, despite having model name "Large", the number of parameters is less than 12M, which is nowhere even close to big language models such as BERT [5], or GPT-3 [3], where the number of parameters are several orders of magnitude larger. Moreover, it has been shown that large Transformers seem to be under-fitted, and extremely overparametrized models seem to generalize on multiple tasks, despite being under-fitted. Thus, it is reasonable to ask : "why not try even larger language models?". In many practical cases, password guessing attacks are performed in parallel and exhaustively. Thus, computational resources are always on short. Thus ultimate goal of

neural network based password guessing attack isn't the accuracy of the model, they are ultimately "performance per parameters", or "performance per resource". Therefore even though having larger language model might increase the performance of the model, but might lead to overall less efficiency for the task.

As a generating method, we sample randomly from the softmax of the logit from the head layer. We choose not to use top-k truncated sampling because passwords have more stochastic behavior, and restricting to top-k must mean restricting the size of the space the generator will cover. In this sampling process, temperature parameter T plays important rule on shaping sampled distribution. We will investigate this in section 4.

3.2 Preparing Dataset

From certain websites such as [7] that are capable of providing leaked password datasets, we prepared Rockyou dataset. The dataset was clearly identical to the dataset that the authors of PassGAN [9] used. We split into train, test dataset just as they did, keeping the same number of testset. Refer to Table 2 for more detailed numbers.

3.3 Training

For better training, we used learning rate range test (LRRT) first introduced by [19] to retrieve the optimal learning rate. Refer to table 1 for precise values. As larger batch size improves training, we conducted LRRT with batch size of 256, and trained the model with batch size of 256 also.

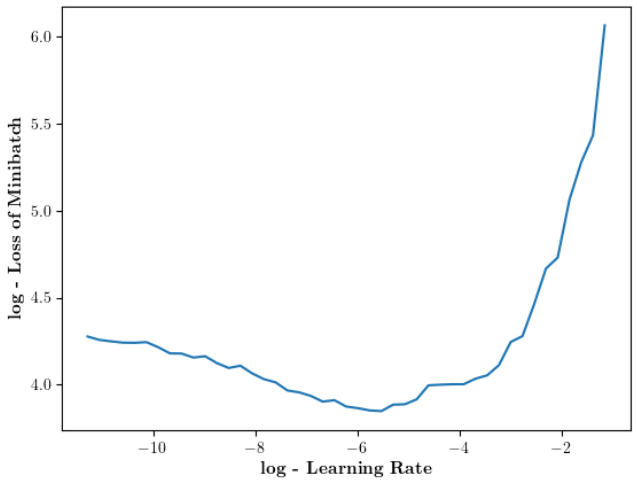

Figure 1: **Plot of Learning Rate Range Test performed on Small model**

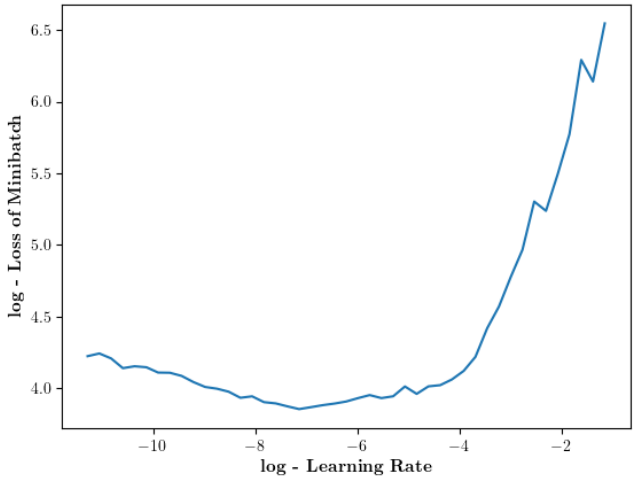

Figure 2: **Plot of Learning Rate Range Test performed on Medium model**

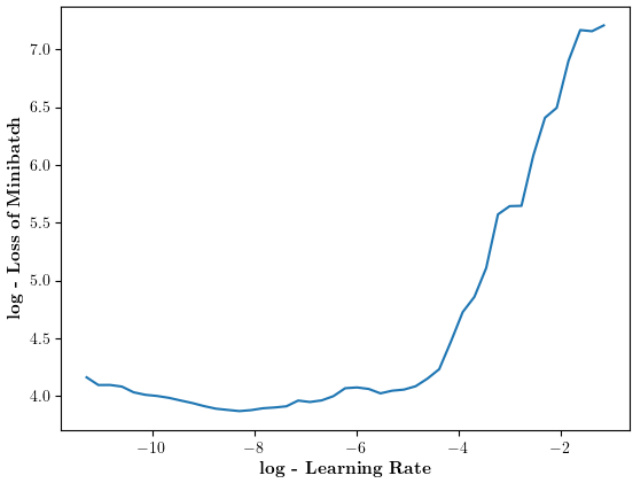

Figure 3: **Plot of Learning Rate Range Test performed on Large model**

Furthermore, we train with cycling learning rate, that was introduced in [20]. We used step size of 1000 for both increasing step and decreasing step, with 5 times learning rate range (thus, starting from learning rate of l to $5l$, within 1000 steps, and decreasing back to l in another 1000 steps), with triangular cycle. We used AdamW optimizer [13] with weight decay of 10^{-10} . To see how the model is training, we test the quality of the model on each checkpoints, where each checkpoints were set between 2000 iterations. This way, we would be checking on every point with lowest learning rate, thus expect most overfitted model of each points. On any case, we would stop the training process when model's performance is on plateau. as it seems more training iteration doesn't hurt the performance, even though it doesn't benefit either. Thus to have consistence comparisons we stop the training on one epoch for all rest of the experiments.

3.4 Evaluation

In password guessing attacking scenario, main interest can be mainly thought of three metrics : Sample time, number of unique samples, and number of correct passwords per sample. We will treat sampling time with less of a priority in our case since authors of PassGAN already [9] suggested that it the "slow candidates" that might make the answer rather than the fast, but less expressive and more heuristic ones. We are also interested in number of unique samples, because we think they suggest another important factor

of the scenario : that password attackers must somehow span all space in some way. We believe that this is the main weakness of model such as [9] and [15], where one does not have the expressiveness of the model at hand.

To have a fair comparison with PassGAN, we tried to keep every evaluation step close to that of PassGAN.

	Transformer (Ours)	PassGAN
Number of Training Samples	5.0 M	9.9M
Number of Training Iterations	4.9 M	12.8M
Number of layers	3 $\sim$5	6
Number of Test Samples	1.9M	1.9M

Table 2: **Comparisons between our Transformer model versus PassGAN**

As explained earlier we created checkpoints every 2000 iteration step and evaluate their performance at every checkpoint. At each checkpoints, we sampled 10^5 passwords to see the performance.

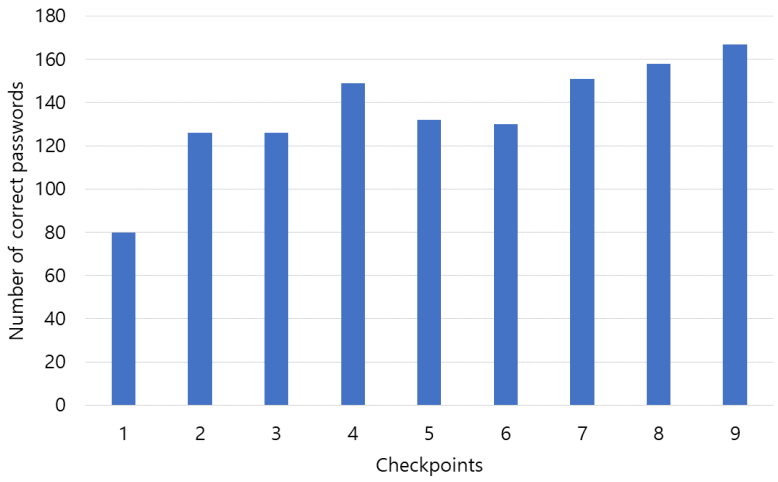

Figure 4: **Number of correct samples per each checkpoints during training, Small model**

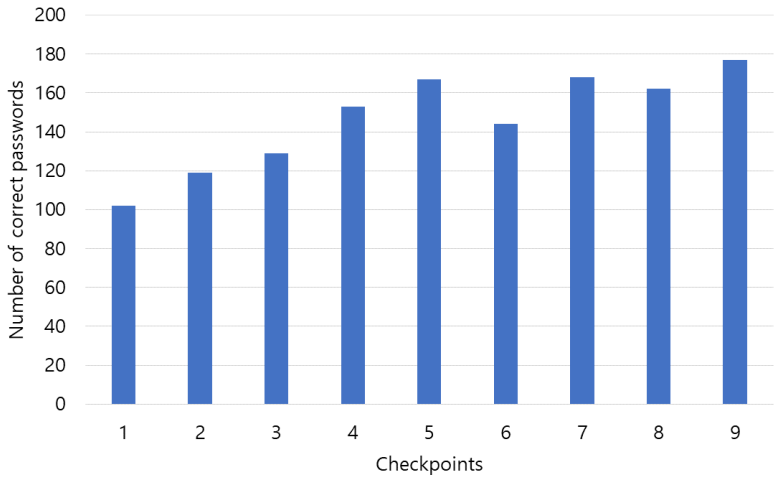

Figure 5: **Number of correct samples per each checkpoints during training, Medium model**

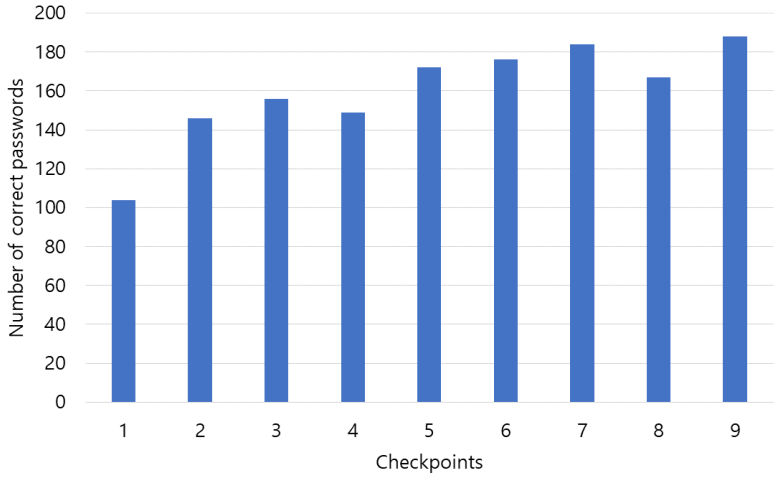

Figure 6: **Number of correct samples per each checkpoints during training, Large model**

These results show that disregarding the model size, within 16 checkpoint model seems to reach its maximum performance. In case of medium model, when we sampled 10^6, we noticed the same behavior : concluding that the model indeed does reach each

it's maximal performance under 1 epoch.

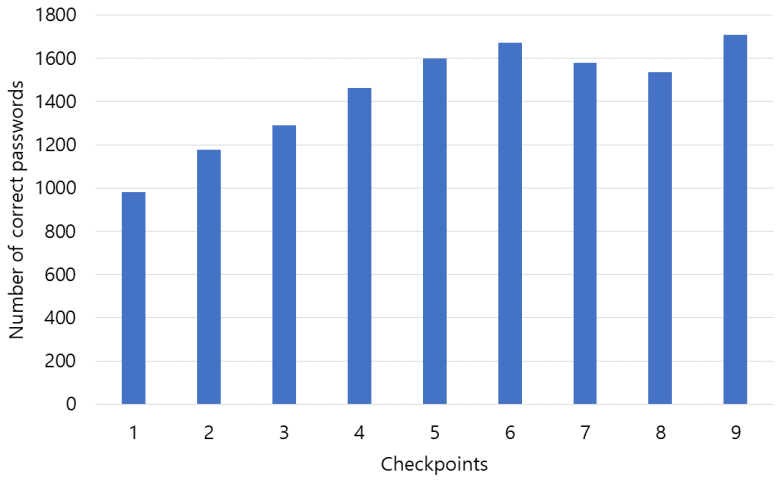

Figure 7: **Number of correct samples per each checkpoints during training, Medium model, sampled 10^6 password candidates.**

Finally, having tested on the testset, we find that single Transformer model is indeed competitive with other Neural Network approaches, mainly PassGAN and RNN based models [15].

Model Name	$k = 10^4$	$k = 10^5$	$k = 10^6$
Small	139	1579	13909
Medium	187	1819	16158
Large	203	1937	17485
PassGAN	103	957	7543

Table 3: **Comparisons between different model under number of correct passwords per sample size k**

In some cases, Transformer model performed 2 time better than PassGAN on just number of correct samples. Moreover, we find that Transformers outperform PassGANs with respect to number of unique samplings. We discuss the factor that might play a great role in number of unique guesses in section 4.

Model Name	$k = 10^4$	$k = 10^5$	$k = 10^6$
Small	9994	99840	989183
Medium	9991	99807	988673
Large	9995	99841	988758
PassGAN	9738	94400	866972

Table 4: **Comparisons between different model under number of unique candidates per sample size** k

Note that with PassGAN, number of unique passwords decreased about 15%, while with our Transformer model less than 2% drops.

4 Effect of Temperature

4.1 Empirical Results

In this section we investigate on the effect of temperature during sampling. Seeing the results from section 3, we see that Medium model is enough to check the validity of any experiments we conduct. Thus, we use medium model to study what the temperature value would do to the model's performance. Temperature value was varied from $2^{-1.9}$ to $2^{1.8}$. We found that in general having high temperature decreases overall accuracy, and having small temperature also decreases overall accuracy.

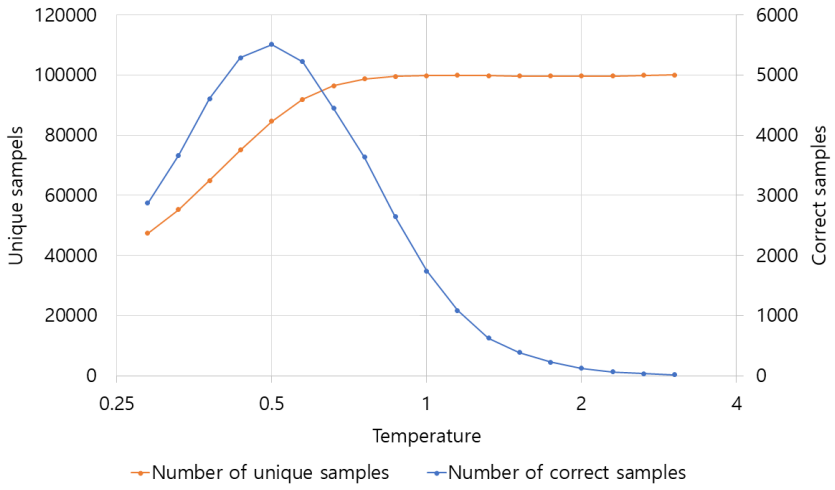

Figure 8: **Number of correct samples and unique samples for each temperature value, Medium model, sampled 10^5 password candidates.**

Then, to how the different temperature effect the performance during training, we sampled with $T = 0.43$ on medium model. We see that with different temperature underfitting nor overfitting doesn't seem to be a problem.

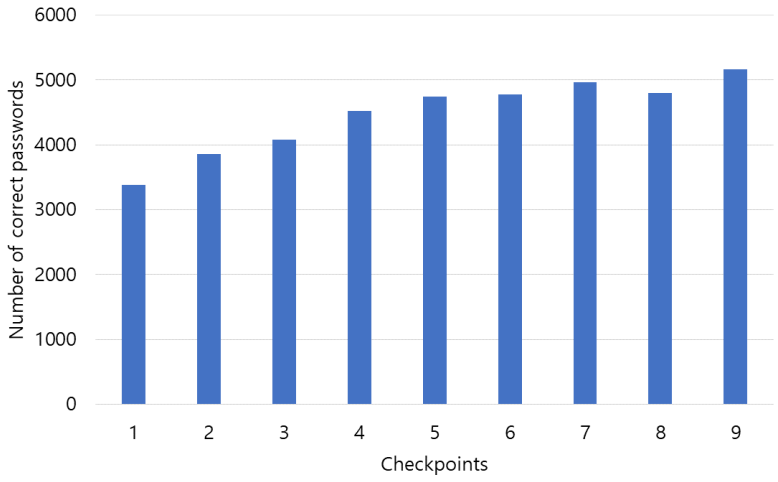

Figure 9: **Number of correct samples per each checkpoints during training, Medium model, T = 0.43, sampled 10^6 password candidates.**

Interestingly, we find that number of unique sample per each checkpoints seem to behave rather radically during training.

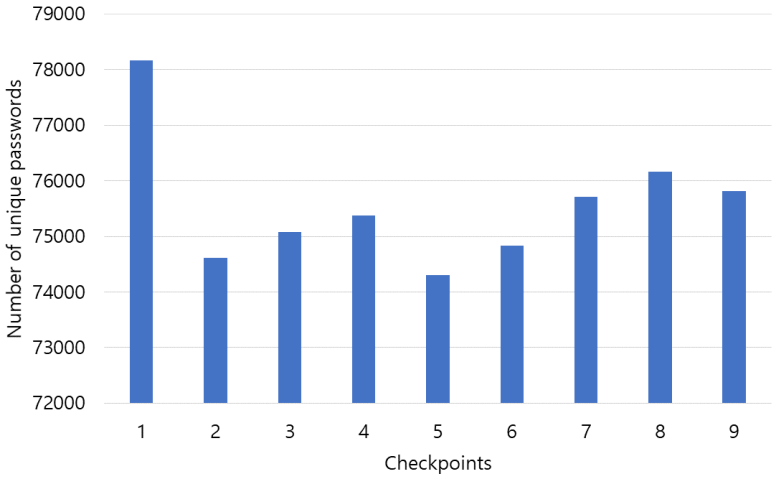

Figure 10: **Number of unique samples per each checkpoints during training, Medium model, T = 0.43, sampled 10^6 password candidates.**

One can clearly see that having different temperature drastically changes performance. Especially, having high temperature increases number of unique samples, but also decreases overall performance. Our initial guess was that having different temperature would just be another hyperparameter that would effect the sampling efficiency, but having tested on ”best temperature per sample size”, we realized there was much deeper phenomena happening.

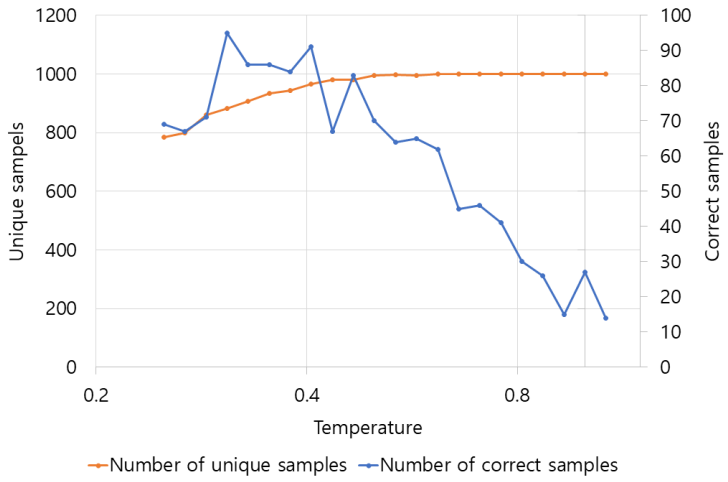

Figure 11: **Number of correct samples and unique samples for each temperature value, Medium model, sampled** 1000 **password candidates.**

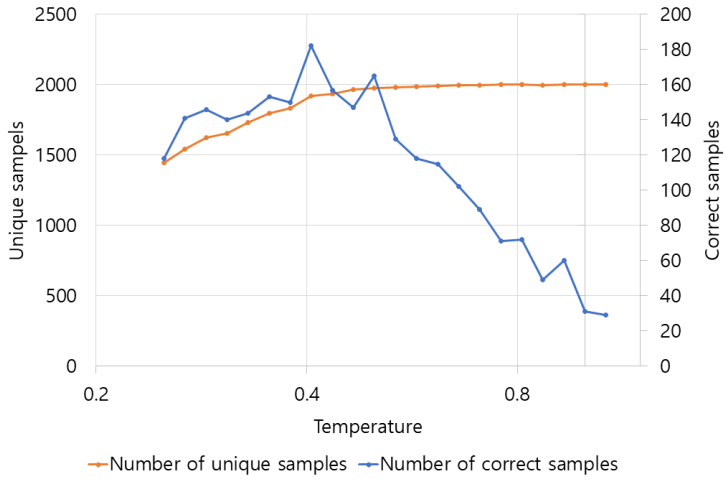

Figure 12: **Number of correct samples and unique samples for each temperature value, Medium model, sampled** 2000 **password candidates.**

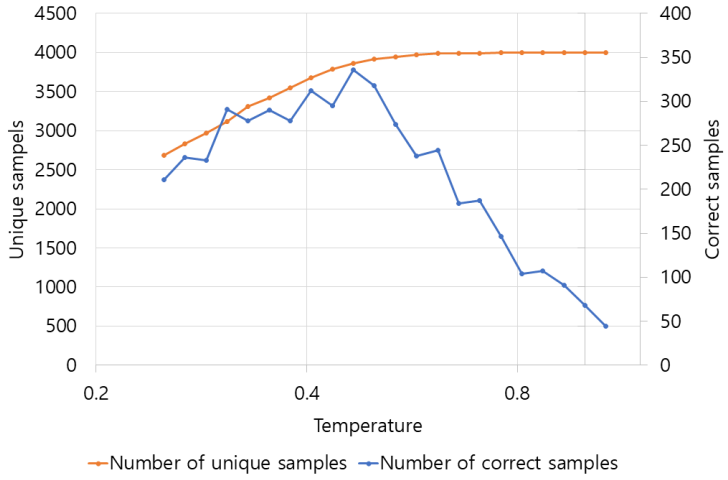

Figure 13: **Number of correct samples and unique samples for each temperature value, Medium model, sampled** 4000 **password candidates.**

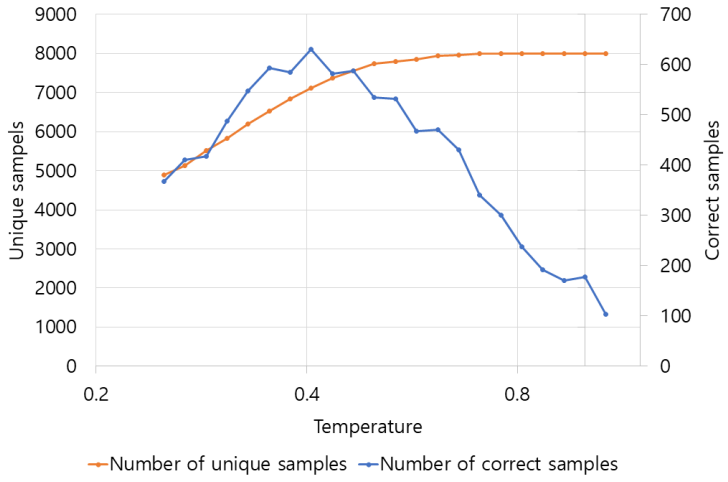

Figure 14: **Number of correct samples and unique samples for each temperature value, Medium model, sampled** 8000 **password candidates.**

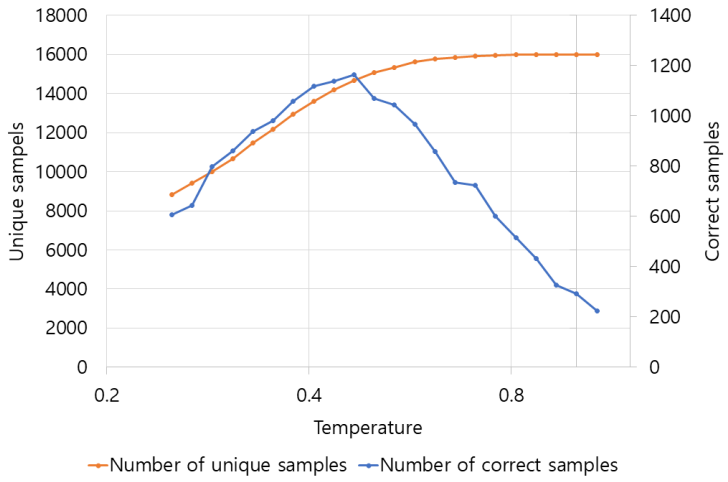

Figure 15: **Number of correct samples and unique samples for each temperature value, Medium model, sampled** 16000 **password candidates.**

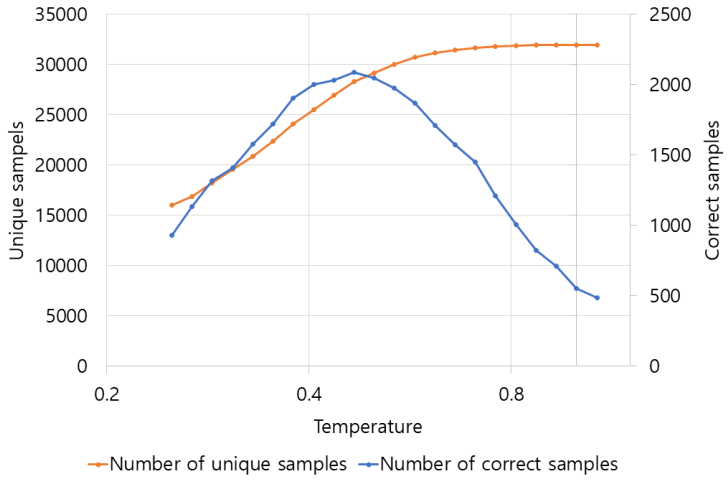

Figure 16: **Number of correct samples and unique samples for each temperature value, Medium model, sampled** 32000 **password candidates.**

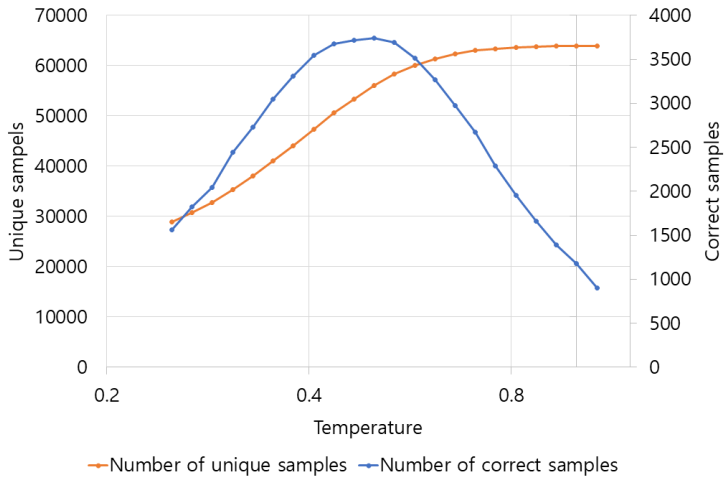

Figure 17: **Number of correct samples and unique samples for each temperature value, Medium model, sampled** 64000 **password candidates.**

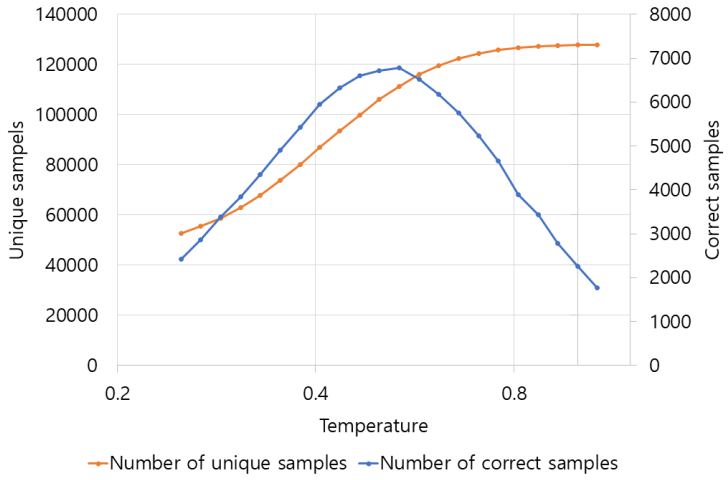

Figure 18: **Number of correct samples and unique samples for each temperature value, Medium model, sampled** 128000 **password candidates.**

From these results we see that number of samples are almost irrelevant to the general behavior of the temperature per each test cases. First is that low temperature causes to sample specifically ”good samples”, making other passwords unable to be sampled from

model. Second is that high temperature causes to sample rather unlikely passwords, thus making performance lower. As we discussed earlier, having precise model does not always mean having best best model in case of password guessing attack. scenario. Thus, having low temperature is clearly unidealistic in many cases, as by definition low temperature reduces the possible space that can be generated by the language model.

Another side of the phenomena was that ”best temperature” for the sample increased as the number of sample increased. For example, on Figure 11, best temperature seems to be around 0.3, which is very different from Figure 18, where best temperature seems to be around 0.55. In other words, ”bump” of the distribution seems to shift as the number of sample increases. We can explain this phenomena with the same framework : if the model samples more candidates, it should have higher temperature to sample more extensively. This effect can be viewed in different way as follows. We view the result respect to unique candidates per sample : Per each temperature of interest, see the ratio of unique number as the sample size increases. Indeed, looking at the result at ! cite figure, one can clearly see that not only does having low temperature provides us with less unique samples,**the ratio of unique samples per sample decreases as well.** Thus it might be unsafe to say that temperature $T = 0.43$ would be best choice for all sample size, as higher might seem like worse choice for smaller sample size, but it would not be the case for larger sample size.

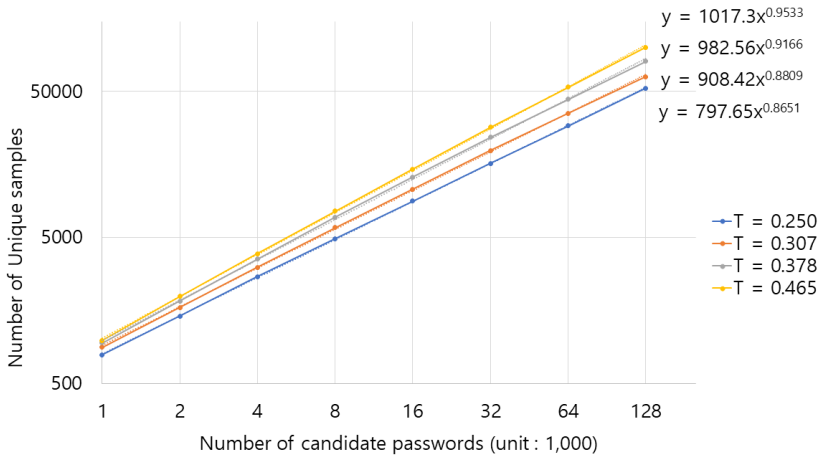

Figure 19: **Number of unique samples for each sample size, Medium model, for various temperature**

Another reformulation of this phenomena is that number of duplicate candidates seems to be somewhat quadratic to the sample size. Refer to the figure below for the specific values.

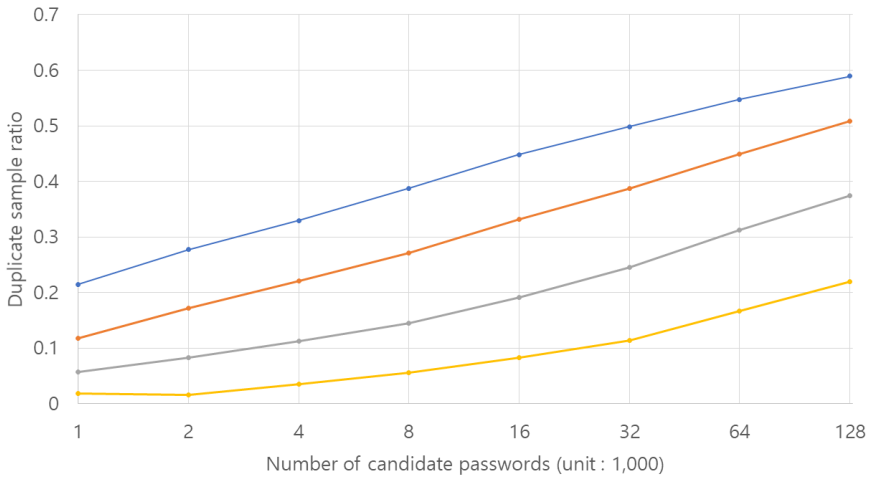

Figure 20: **Ratio of duplicate candidates for samples, Medium model, for various temperature**

4.2 Explanation

We can explain this behavior with elementary probability theory. Consider all possible strings as S, which there exists probability measure $\mathbb{P}$ assigned to each strings as "probability of a person using this string as a password". Since the number of passwords is finite, the probability becomes countable measure. We can restrict to very small set, such as the dataset above and argument still holds. Our Transformer than samples each string randomly according to inaccurate model of the internal structure -the neural network-, where it attempts to find the logit of probability of next character using all characters before it. Denote this probability as $\mathbb{P}_{\mathbb{T}}$, which can be thought of a Markov chain model. Since the probability assigned itself isn't stochastic, we can factorize the probability that string s is sampled by the Transformer in a standardized way

$$\mathbb{P}_{\mathbb{T}}(S = s_0, s_1, ...s_n) = \mathbb{P}_{\mathbb{T}}(S_n = s_n | S_{0,n-1} = s_0, s_1, ..s_{n-1}) \times \mathbb{P}_{\mathbb{T}}(S_{0,n-1} = s_0, s_1, ...s_{n-1})$$

Which is than recursively defined. Since neural network is deterministic, we have fixed logit function l, thus,

$$\mathbb{P}_{\mathbb{T}}(S_n = s_n | S_{0,n-1} = s_0, s_1, ...s_{n-1}) \propto \exp(l(S_n = s_n | S_{0,n-1} = s_0, s_1, ...s_{n-1})/T)$$

Therefore using recursive argument, heuristically, we can say

$$\log \mathbb{P}_{\mathbb{T}}(S = s_0, s_1, ...s_n) \approx l(S = s_0, s_1, ...s_n)/T$$

Now if the Transformer samples k elements, K, the number of unique candidates the Transformer samples would be

$$X = \mathbb{E}(\sum_{s \in S} 1_{s \in K})$$

which is than by linearity of expectation,

$$X = \sum_{s \in S} \mathbb{E}(1_{s \in K}) = \sum_{s \in S} \mathbb{P}(s \in K)$$

Notice that

$$\begin{aligned}
\mathbb{P}(s \in K) &= 1 - \mathbb{P}(s \notin K) \\
&= 1 - (1 - \mathbb{P}_{\mathbb{T}}(S = s))^k \\
&\approx k\mathbb{P}_{\mathbb{T}}(S = s) - \frac{k(k-1)}{2}\mathbb{P}_{\mathbb{T}}(S = s)^2
\end{aligned}$$

Thus we have

$$\begin{aligned}
X &\approx k \sum_{s \in S} \mathbb{P}_{\mathbb{T}}(S = s) - k^2 \sum_{s \in S} \mathbb{P}_{\mathbb{T}}(S = s)^2 \\
&= k - O(k^2 \mathbb{E}(\mathbb{E}(1_{S=s})^2)) \\
&= k - O(k^2 (Var(\mathbb{E}(1_{S=s})) + \mathbb{E}(\mathbb{E}(1_{s=S}))^2)) \\
&= k - O(k^2 Var(\mathbb{E}(1_{S=s})) + k^2/N) \\
&= k - O(k^2 Var(\text{softmax}(\mathbb{P}_{\mathbb{T}}/T)))
\end{aligned}$$

This equation explains the results we got. Mainly, ratio of duplicate candidate grows linearly as number of sample increases. Also, such ratio grows slower if the variance of the probability distribution is small, which is what happens when temperature is high.

5 Using Pre-trained Transformer

In this section, we will describe our findings on effect of pre-training when using Transformer as password guessing attacker.

5.1 Model Architecture

As simmilar to before, we used transformer that was introduced in the first GPT paper [17]. The following hyperparameters are only difference.

5.2 Preparing Dataset

To pre-train the model on general English corpus, we needed set of words that included general nouns, that doesn't have much replications, and that somehow spans large informative language structure. At first we collected dataset resembling that of a CommonCrawl, but soon to realize that it was an overkill for such a task. This is because our task leverages very little information of natural language, and differ drastically from sentence modeling. Thus, we can argue that it is more reasonable to pre-train on vocablary themselves rather than sentences, as entirety of vocabulary of English would also contain lots of compound-words that exhibits many natural language features. Thus we prepared such dataset from [23].

For fine-tuning and testing how pre-trained Transformer generalizes to different datasets, we prepared 4 different password datasets: leaked passwords from platform Chegg, Mathway, Ashleymadison, and LinkeIn. The follwing table includes detailed information on each datasets.

Dataset index	Name of the platform	Number of unique passwords
1	Chegg	4.99M
2	Mathway	3.34M
3	Ashleymadison	2.68M
4	LinkedIn	5.00M

5.3 Training and Testing

To see rather pre-training helps, we first train the Medium model to the Wikipedia vocabulary dataset, and fine-tune on the Rockyou dataset, the same train, test set split as done in section 3. Every other training parameters and conditions were identical, including pre-training stage. We simply trained with treating words as passwords.

Also as we are interested in generalization effect of pre-training, we would fine-tune on the ith data and test on j data for $i \neq j$. To see rather pre-training benifits, we make a comparison with classical case. That is, just training on i th dataset and testing on jth dataset.

5.4 Results

Unlike our expectation, pre-training Medium model on Wikipedia vocabulary dataset and fine-tuning on Rockyou dataset didn't yield better result. Result on training on i th dataset and testing on j th dataset is on Table 5.

	Train 1	Train 2	Train 3	Train 4
Test 1		5.9	4.8	0.7
Test 2	44.9		33.1	8.1
Test 3	143.5	70.0		12.8
Test 4	0.69	2.6	2.6	

Table 5: **Result of Not Pre-training : Number of correctly guessed passwords per each dataset (units : 1,000)**

Similarly, pre-training on Wikipedia vocabulary dataset and fine-tuning on i th dataset, and testing on j th dataset results as described in Table 6.

	Train 1	Train 2	Train 3	Train 4
Test 1		6.6	5.6	0.8
Test 2	80.2		35.1	15.6
Test 3	148.2	73.0		21.3
Test 4	0.8	2.7	3.0	

Table 6: **Result of Pre-traing : Number of correctly guessed passwords per each dataset (units : 1,000)**

As one can clearly see in this case, pre-training certainly benefits every 4 model, under evaluation on every other different datasets.

6 Conclusion

In this paper, we introduced the question rather sampling method from Transformer model could improve upon the previous neural network methods, such as PassGAN [9] and RNN-based models [15]. We proposed that with character level embedding and sampling-based generation, Transformers outperforms previously claimed models with small number of parameters and training time. Also, we studied how the temperature parameter in sampling could effect and potentially benefit the generative method, using empirical results combined with probabilistic modeling of the results. Finally, we found

that pre-training is often beneficial for generalization, but not often the case with local datasets.

References

[1] Anonymous. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *Submitted to International Conference on Learning Representations*. under review. 2021. URL: https://openreview.net/forum?id=YicbFdNTTy.

[2] Peter F Brown et al. "Class-based n-gram models of natural language". In: *Computational linguistics* 18.4 (1992), pp. 467–480.

[3] Tom B Brown et al. "Language models are few-shot learners". In: *arXiv preprint arXiv:2005.14165* (2020).

[4] Angelo Ciaramella et al. "Neural network techniques for proactive password checking". In: *IEEE Transactions on Dependable and Secure Computing* 3.4 (2006), pp. 327–339.

[5] Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[6] *HashCat*. 2020. URL: HashCat.%202017.%20https://hashcat.net..

[7] *Hashes.org*. 2020. URL: https://hashes.org/leaks.php.

[8] Shen He et al. "Research on Password Cracking Technology Based on Improved Transformer". In: *Journal of Physics: Conference Series*. Vol. 1631. 1. IOP Publishing. 2020, p. 012161.

[9] Briland Hitaj et al. *PassGAN: A Deep Learning Approach for Password Guessing*. 2019. arXiv: 1709.00440 [cs.CR].

[10] *John the Ripper*. 2020. URL: https://www.openwall.com/john/.

[11] Ryan Kiros et al. "Skip-thought vectors". In: *Advances in neural information processing systems*. 2015, pp. 3294–3302.

[12] Quoc Le and Tomas Mikolov. "Distributed representations of sentences and documents". In: *International conference on machine learning*. 2014, pp. 1188–1196.

[13] Ilya Loshchilov and Frank Hutter. "Decoupled weight decay regularization". In: *arXiv preprint arXiv:1711.05101* (2017).

[14] Jerry Ma et al. “A study of probabilistic password models”. In: *2014 IEEE Symposium on Security and Privacy*. IEEE. 2014, pp. 689–704.

[15] William Melicher et al. “Fast, lean, and accurate: Modeling password guessability using neural networks”. In: *25th {USENIX} Security Symposium ({USENIX} Security 16)*. 2016, pp. 175–191.

[16] Arvind Narayanan and Vitaly Shmatikov. “Fast dictionary attacks on passwords using time-space tradeoff”. In: *Proceedings of the 12th ACM conference on Computer and communications security*. 2005, pp. 364–372.

[17] Alec Radford et al. *Improving language understanding by generative pre-training*. 2018.

[18] Timo Schick and Hinrich Schütze. “It’s Not Just Size That Matters: Small Language Models Are Also Few-Shot Learners”. In: *arXiv preprint arXiv:2009.07118* (2020).

[19] Leslie N Smith. “A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay”. In: *arXiv preprint arXiv:1803.09820* (2018).

[20] Leslie N Smith. “Cyclical learning rates for training neural networks”. In: *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2017, pp. 464–472.

[21] Joseph Turian, Lev Ratinov, and Yoshua Bengio. “Word representations: a simple and general method for semi-supervised learning”. In: *Proceedings of the 48th annual meeting of the association for computational linguistics*. 2010, pp. 384–394.

[22] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems*. 2017, pp. 5998–6008.

[23] *Wikipedia-Summary-Datset*. 2020. URL: `https://github.com/tscheepers/Wikipedia-Summary-Dataset`.

[24] Huan Zhou, Qixu Liu, and Fangjiao Zhang. “Poster: An Analysis of Targeted Password Guessing Using Neural Networks”. In: *Proceedings of the 2017 IEEE Symposium on Security and Privacy (IEEE S&P 2017)*. 2017.