PassGAN: A Deep Learning Approach for Password Guessing

(Briland Hitaj, Paolo Gasti, Giuseppe Ateniese, Fernando Perez-Cruz)

Original paper appeared in NeurIPS 2018 Workshop on Security in Machine Learning (SecML'18)

Yohannes B. Bekele
March 2020

Instructor
Dr. Mahmoud N Mahmoud

Outline

- *Introduction*
- *Literature Review*
- *System Model*
- *Experiment Setup*
- *Training & Testing*
- *Evaluation*
- *Shortcomings & Performance Enhancements*
- *Problem Mitigation*
- *Conclusion*

Passwords and why they matter

First and most important line of defense for security

Many users reuse their passwords

Data breaches can impact a number of sites

When databases are breached, stolen passwords are usually hashed

Adversaries cannot directly access the information

Password guessing

Identifying weak passwords when they are stored in hashed form

Password Guessing Approaches

Traditional Password guessing

Ad-hoc and based on intuitions on how users choose passwords

- *John the Ripper [1]*
- *HashCat [2]*

- An exhaustive brute-force attack
  - Given the set of characters[a-z], [A-Z], [0-9] with password length up to 8,
    - With 10^8 passwords/sec it takes 25 days.
    - With 10^9passwords/sec, it takes 60 hours
- Dictionary-based attack
  - Hash comparison
- Rule-based approach on top of the dictionary list.

[1]. John the Ripper. 2017. http://www.openwall.com/john/.
[2]. HashCat. 2017. https://hashcat.net.

Password Guessing Approaches

Data-driven Password Guessing

Based on deep learning

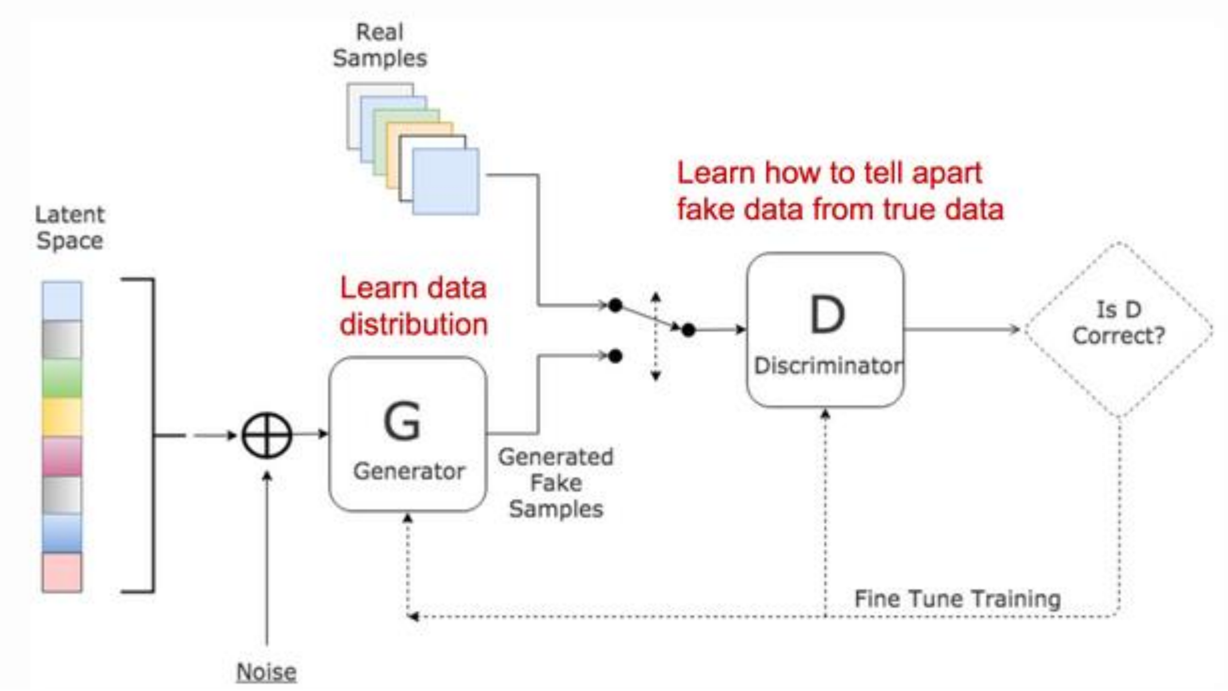
- *PassGAN*
- *FLA*

- **Why?**
  - Capture a large variety of properties and structures
  - No priori knowledge needed
- **How?**
  - Two steps:
    - Train a deep neural network
    - Generate new samples that follow the same distribution.
- **What?**
  - Generative Adversarial Networks or Recurrent Neural Networks

Literature Review(1/7)

Generative Adversarial Networks (GANs)

- Two active processes:
  - discriminating & generating
- Generator tries to deceive the discriminator by imitating the input data
- Discriminator tries to determine which of the inputs are from the actual data and which are from the generator



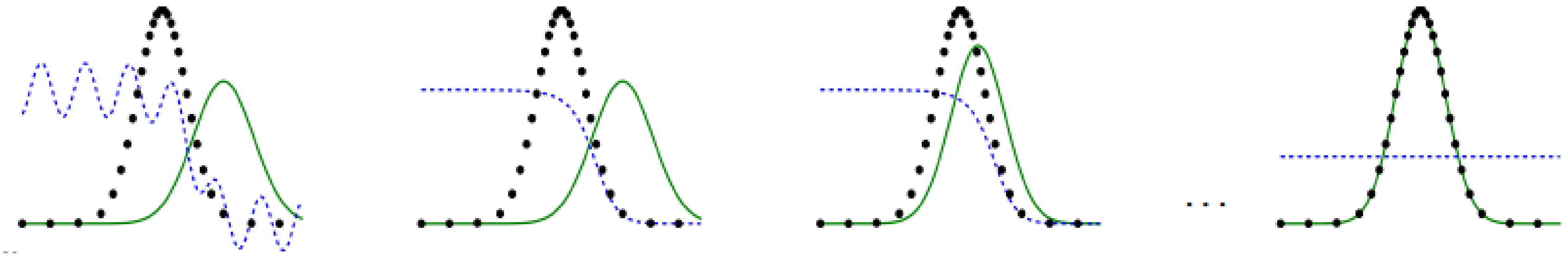
- Becomes better in creating similar outputs to the dataset as it iterates more

https://pathmind.com/wiki/generative-adversarial-network-gan

Literature Review(2/7)

Expressed mathematically

$$\min_G \max_D L(D,G) = \mathbb{E}_{x\sim p_r(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))]$$



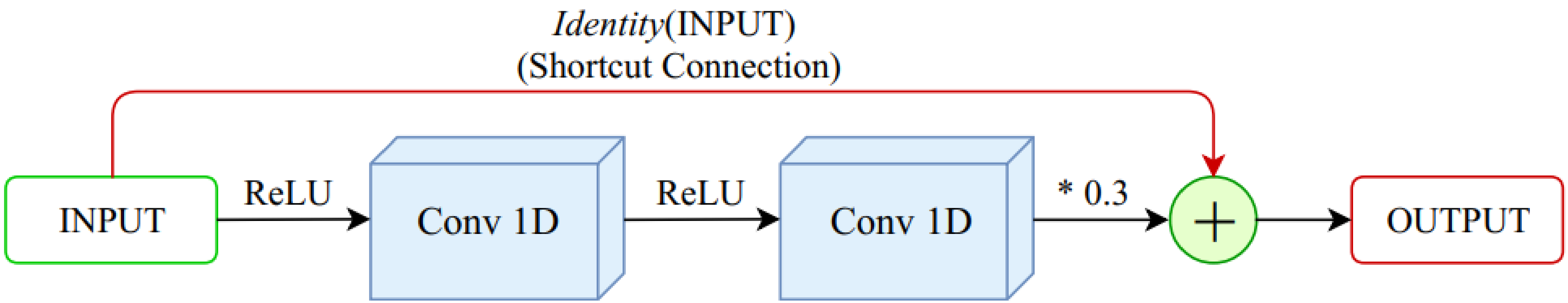
GANs are trained by simultaneously updating the discriminative distribution (***D*, blue, dashed line**) so that it discriminates between samples from the data generating distribution (**black, dotted line**) from those of the **g**enerative distribution G (green, solid line).

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In Advances in neural information processing systems. 2672–2680.

Improved Training of Wasserstein GANs (IWGAN)

- In GANs, initially the training error decreases as the number of layer increases. However, after reaching a certain number of layers, training error starts increasing again.

- ResNet [6] :- includes "shortcut connection" between layers.



- ResNet Network Converges faster compared to plain counter part of it.

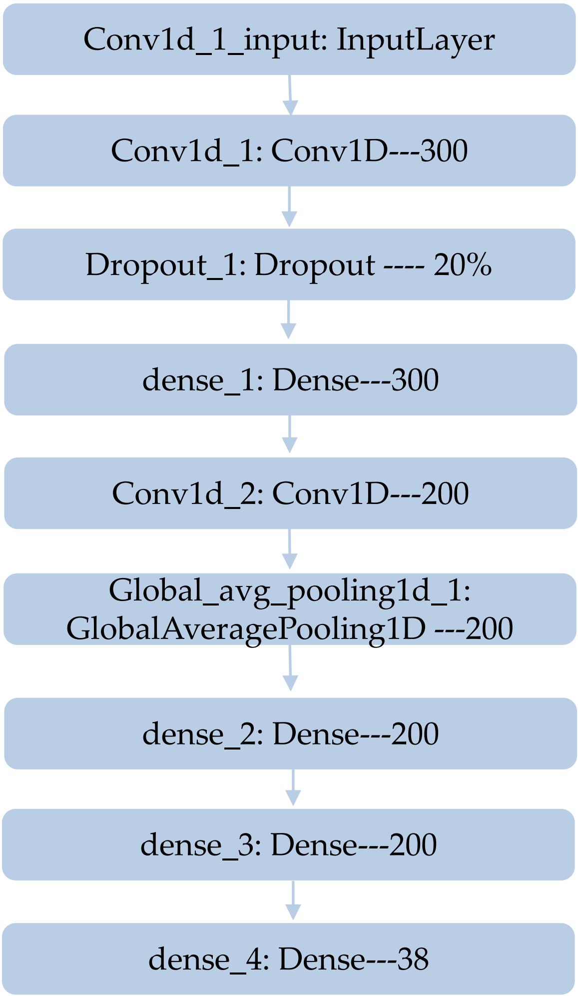
[6]. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), 770–778.

Literature Review(4/7)

Name/ Authors	Approach	Methodology
JtR [1]	Password guessing (on CPU)	• Exhaustive brute force attacks; • Dictionary-based attacks; • Rule-based attacks • Markov-model-based attacks
HashCat [2]	Password guessing (on GPU)	Same as JtR
Olsen [3]	Password generation	CNN
Melicher et al. [4]	Password strength estimation	Based on RNN, LSTM
Lingzhi Xu et al. [5]	Password generation	LSTM

A Machine Learning Approach to Predicting Passwords [3]

Research question	How can machine learning models be used for password cracking
Approach	Uses Convolutional Neural Networks
Methodology	➢ For building and training keras framework was used ➢ Relu and softmax activation functions ➢ 38 output neurons represent each character in character set including a line-terminator. ➢ Trained on the Rockyou dataset
Results	The final validation accuracy is 41.3% and the final training accuracy is 45.4%
Drawback	Slow password generation ➢ 14,000,000/100 = 140,000sec ≈ 39hours



[3]. Olsen, Christoffer (2018). „A Machine Learning Approach to Predicting Passwords“. MA thesis. Technical University of Denmark. Available from: https://www.researchgate.net/profile/Georg_Knabl/publication/328719001_Machine_Learning-driven_Password_Lists/links/5bdd8266299bf1124fb6f4d9/Machine-Learning-driven-Password-Lists.pdf

Literature Review(6/7)

A Fast, Lean, and Accurate: Modeling Password Guessability Using Neural Networks [4]

Research question

Using ANNs to model text passwords' resistance to guessing attacks and explore how different architectures and training methods impact NNs' guessing effectiveness.

Approach

Uses Recurrent Neural Networks

Methodology

- Two different recurrent architectures of RNN are used namely LSTM and refined LSTM models
- The models typically used three LSTM recurrent layers and two densely connected layers for a total of five layers.
- Keras library and neocortex browser implementation of neural networks.
- Testing data from Mechanical Turk (MTurk) and 000webhost

Results

This approach outperforms traditional generation methods in terms of recognized password policies and at guess numbers above 10^{10}.

[4]. William Melicher, Blase Ur, Sean M Segreti, Saranga Komanduri, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. 2016. Fast, Lean, and Accurate: Modeling Password Guessability Using Neural Networks.. In USENIX Security Symposium. 175–191.

Literature Review(7/7)

Password guessing based on LSTM recurrent neural networks [5]

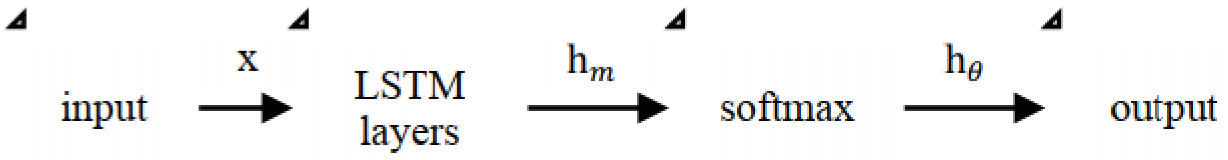
Research question How to use Recurrent Neural Networks for password guessing?

Approach Uses Recurrent Neural Networks

Methodology

- The basic ideas of the password guessing model include:
  - The probability distribution of x(t) can be predicted by the NN when using x(1), x(2), …,x(t-1) as sequence inputs
  - Next character can be decided by a selection algorithm according to probability distribution
- The model contains 2 hidden LSTM layers, 256 neurons per LSTM layer
- The LSTM model is trained by 30 million Rockyou passwords, test with Rockyou test set (2.6 million passwords), Myspace dataset (MS) and Facebook dataset (FB).

Results The generated 3.4 billion passwords could cover 81.52% of the remaining Rockyou dataset.

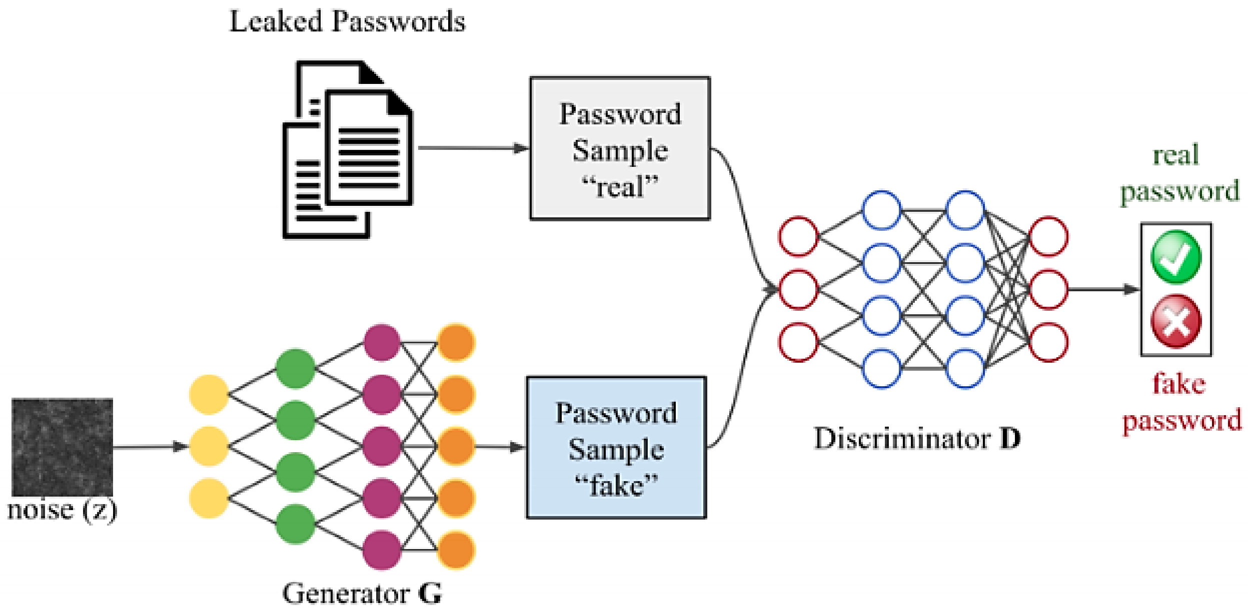


[5]. Lingzhi Xu, Can Ge, Weidongg Qiu, Zheng Huang, Zheng Gong, Jie Guo, and Huijuan Lian. Password guessing based on lstm recurrent neural networks, July 2017

System Model(1/2)

- Uses IWGANs to learn the distribution of real passwords from password leaks, and to generate password guesses.

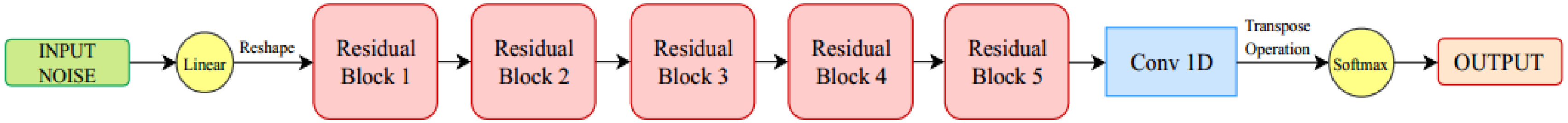
- Two Significances
  - *Can be an efficient and accurate password cracking tool*
  - *Offers a distinct advantage in being able to create passwords indefinitely*



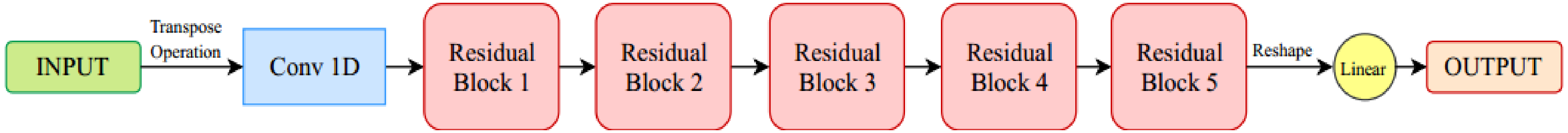
http://www.cs.tufts.edu/comp/116/archive/fall2018/achen.pdf

System Model(2/2)

Detail system model



(a) Generator Architecture, *G*



(b) Discriminator Architecture, *D*

Experiment Setup (1/8)

Hardware

- 64GB RAM, 12-core
- 2.0 GHz Intel Xeon CPU
- NVIDIA GeForce GTX 1080 Ti GPU with 11GB of global memory

Software

- TensorFlow version 1.2.1 for GPUs
- Python version 2.7.12
- Ubuntu 16.04.2 LTS

Parameters

- Batch size = 64
- Number of iterations = 199,000
- Number of discriminator iterations per generator iteration = 10
- Model dimensionality = 5*128
- Gradient penalty coefficient (λ) = 10
- Output sequence length = 10
- Size of the input noise vector (seed) = 128 FPN
- Parameters for Adam optimizer

    Learning rate = 0.0001

    Coefficient $\beta1$ = 0.5

    Coefficient $\beta2$ = 0.9

Experiment Setup (2/8)

Dependencies

Time

- Provides various time-related functions

Pickle

- Implements binary protocols for serializing and de-serializing a Python object structure

Argparse

- Write user-friendly command-line interfaces

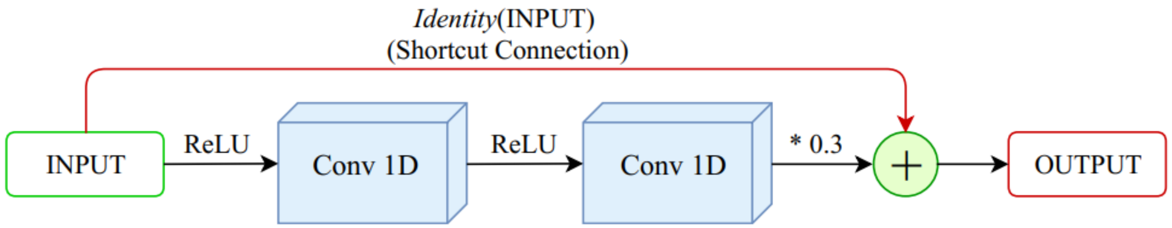
Numpy

- Scientific computing

Tensorflow

- Numerical computation and large-scale machine learning

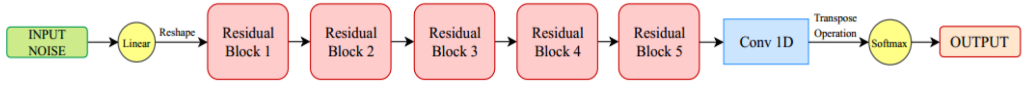
Residual block code snippet



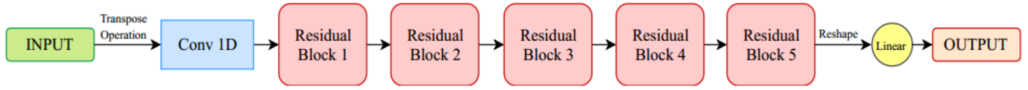

```python
def ResBlock(name, inputs, dim):
    # print("- Creating ResBlock -")
    output = inputs
    output = tf.nn.relu(output)
    output = lib.ops.conv1d.Conv1D(name+'.1', dim, dim, 5, output)
    # print("After conv:", output)
    output = tf.nn.relu(output)
    output = lib.ops.conv1d.Conv1D(name+'.2', dim, dim, 5, output)
    return inputs + (0.3*output)
```

Experiment Setup (4/8)

G & D blocks' code snippet



(a) Generator Architecture, *G*



(b) Discriminator Architecture, *D*

```
def Generator(n_samples, seq_len, layer_dim, output_dim, prev_outputs=None):
    # print("- Creating Generator -")
    output = make_noise(shape=[n_samples, 128])
    # print("Initialized:", output)
    output = lib.ops.linear.Linear('Generator.Input', 128, seq_len * layer_dim, output)
    # print("Lineared:", output)
    output = tf.reshape(output, [-1, seq_len, layer_dim,])
    # print("Reshaped:", output)
    output = ResBlock('Generator.1', output, layer_dim)
    output = ResBlock('Generator.2', output, layer_dim)
    output = ResBlock('Generator.3', output, layer_dim)
    output = ResBlock('Generator.4', output, layer_dim)
    output = ResBlock('Generator.5', output, layer_dim)
    output = lib.ops.conv1d.Conv1D('Generator.Output', layer_dim, output_dim, 1, output)
    output = softmax(output, output_dim)
    return output
```

```
def Discriminator(inputs, seq_len, layer_dim, input_dim):
    output = inputs
    output = lib.ops.conv1d.Conv1D('Discriminator.Input', input_dim, layer_dim, 1, output)
    output = ResBlock('Discriminator.1', output, layer_dim)
    output = ResBlock('Discriminator.2', output, layer_dim)
    output = ResBlock('Discriminator.3', output, layer_dim)
    output = ResBlock('Discriminator.4', output, layer_dim)
    output = ResBlock('Discriminator.5', output, layer_dim)
    output = tf.reshape(output, [-1, seq_len * layer_dim])
    output = lib.ops.linear.Linear('Discriminator.Output', seq_len * layer_dim, 1, output)
    return output
```

Experiment Setup (5/8)

Modeling Generator

```
fake_inputs = models.Generator(args.batch_size, args.seq_length, args.layer_dim, len(charmap))
fake_inputs_discrete = tf.argmax(fake_inputs, fake_inputs.get_shape().ndims-1)
```

Modeling Discriminator

```
disc_real = models.Discriminator(real_inputs, args.seq_length, args.layer_dim, len(charmap))
disc_fake = models.Discriminator(fake_inputs, args.seq_length, args.layer_dim, len(charmap))

disc_cost = tf.reduce_mean(disc_fake) - tf.reduce_mean(disc_real)
gen_cost = -tf.reduce_mean(disc_fake)
```

Experiment Setup (6/8)

Training code snippet

```python
# WGAN lipschitz-penalty
alpha = tf.random_uniform(
    shape=[args.batch_size,1,1],
    minval=0.,
    maxval=1.
)

differences = fake_inputs - real_inputs
interpolates = real_inputs + (alpha*differences)
gradients = tf.gradients(models.Discriminator(interpolates, args.seq_length, args.layer_dim, len(charmap)), [interpolates]
slopes = tf.sqrt(tf.reduce_sum(tf.square(gradients), reduction_indices=[1,2]))
gradient_penalty = tf.reduce_mean((slopes-1.)**2)
disc_cost += args.lamb * gradient_penalty

gen_params = lib.params_with_name('Generator')
disc_params = lib.params_with_name('Discriminator')
```

Training code snippet

```
gen_train_op = tf.train.AdamOptimizer(learning_rate=1e-4, beta1=0.5, beta2=0.9).minimize(gen_cost, var_list=gen_params)
disc_train_op = tf.train.AdamOptimizer(learning_rate=1e-4, beta1=0.5, beta2=0.9).minimize(disc_cost, var_list=disc_params)
```

- ***Adam*** combines the best properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems.
- ***learning rate:-*** The proportion that weights are updated
- ***Beta1:-*** For decaying the running average of the gradient
- ***Beta2:-*** For decaying the running average of the square of the gradient

Adam == Adaptive moment estimation

Generating samples

```
def generate_samples():
    samples = session.run(fake_inputs)
    samples = np.argmax(samples, axis=2)
    decoded_samples = []
    for i in range(len(samples)):
        decoded = []
        for j in range(len(samples[i])):
            decoded.append(inv_charmap[samples[i][j]])
        decoded_samples.append(tuple(decoded))
    return decoded_samples
```

```
# Output to text file after every 100 samples
if iteration % 100 == 0 and iteration > 0:

    samples = []
    for i in range(10):
        samples.extend(generate_samples())

    for i in range(4):
        lm = utils.NgramLanguageModel(i+1, samples, tokenize=False)
        lib.plot.plot('js{}'.format(i+1), lm.js_with(true_char_ngram_lms[i]))

    with open(os.path.join(args.output_dir, 'samples', 'samples_{}.txt').format(iteration), 'w') as f:
        for s in samples:
            s = "".join(s)
            f.write(s + "\n")
```

Training and Testing (1/2)

- Two goals:
  - How well PassGAN predicts passwords when trained and tested on the same dataset
  - Whether PassGAN generalizes across password datasets

RockYou Dataset [7]

- A password list derived from an attack on a former MySpace supplier
- ✓32,503,388 passwords
- ✓29,599,680 passwords ≤ 10 characters
- ✓80% training set & 20% unobserved passwords' testing set

LinkedIn Dataset [8]

- 43,354,871 unique passwords ≤ 10 characters
- ✓40,593,536 were not in the training dataset from RockYou.

[7]. RockYou. 2010. RockYou. http://downloads.skullsecurity.org/passwords/rockyou.txt.bz2
[8]. LinkedIn. [n. d.]. LinkedIn. https://hashes.org/public.php

Password Sampling Procedure for HashCat, JTR, Markov Model, PCFG and FLA

HashCat and JTR

Instantiated using passwords from the training set sorted by frequency in descending order

HashCat Best64

Generated 754,315,842 passwords, out of which 361,728,683 were unique and of length 10 characters or less

HashCat gen2 and JTR SpiderLab

Uniformly sampled a random subset of size 10^9 from their output

FLA

Trained a model containing 2-hidden layers and 1 dense layer of size 512.

- With $p = 10^{-10}$; 747,542,984 passwords of length 10 characters or less are generated

3-gram Markov model

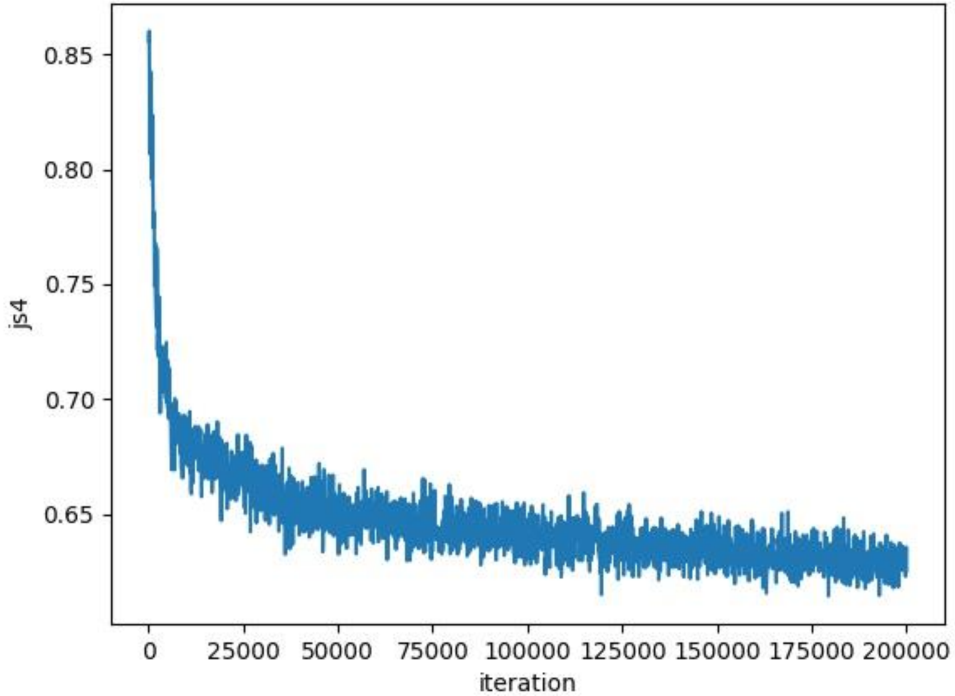
Generated 494,369,794 unique passwords of length 10 or less

PCFG

Generated 10 billion unique passwords of length 10 or less

Evaluation (1/6)

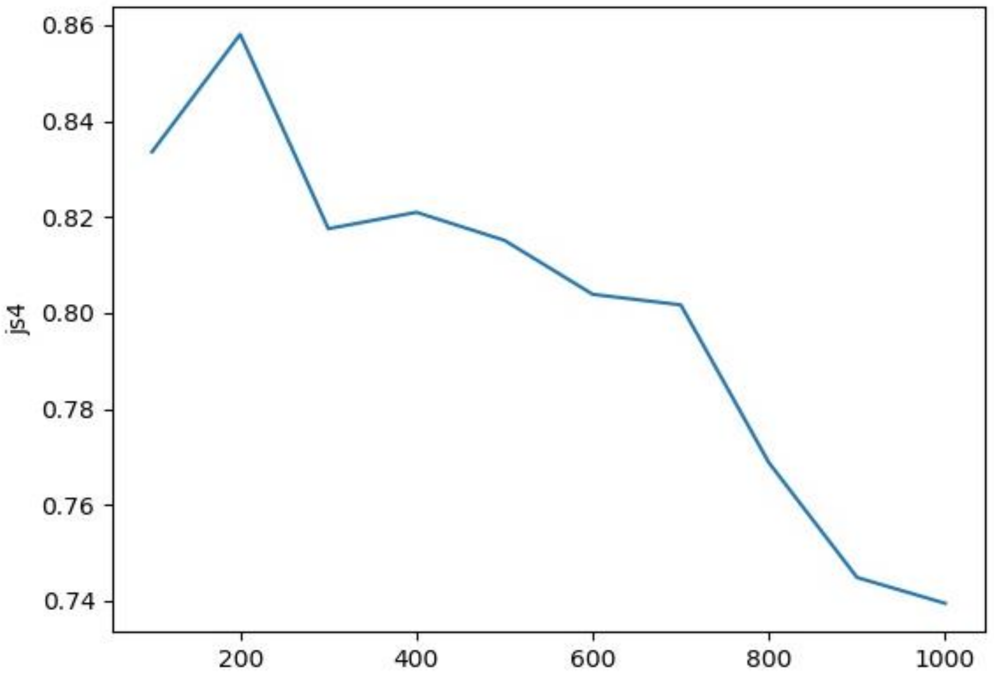
Training Loss



- The Jensen–Shannon divergence is a method of measuring the similarity between two probability distributions bounded by [0,1]
- Minimizing generator yields minimizing the JS divergence when the discriminator is optimal.

Evaluation (2/6)

- Code demonstrated on:
  - Intel Core i5
  - 8GB RAM
  - 512 SSD
  - No GPU card
- Code run on Jupyter Notebook
- Parameters kept as initial except iterations & dataset size
- 10^6 passwords generated
- Just a POC ☺ ☺ , took more than 4 hours

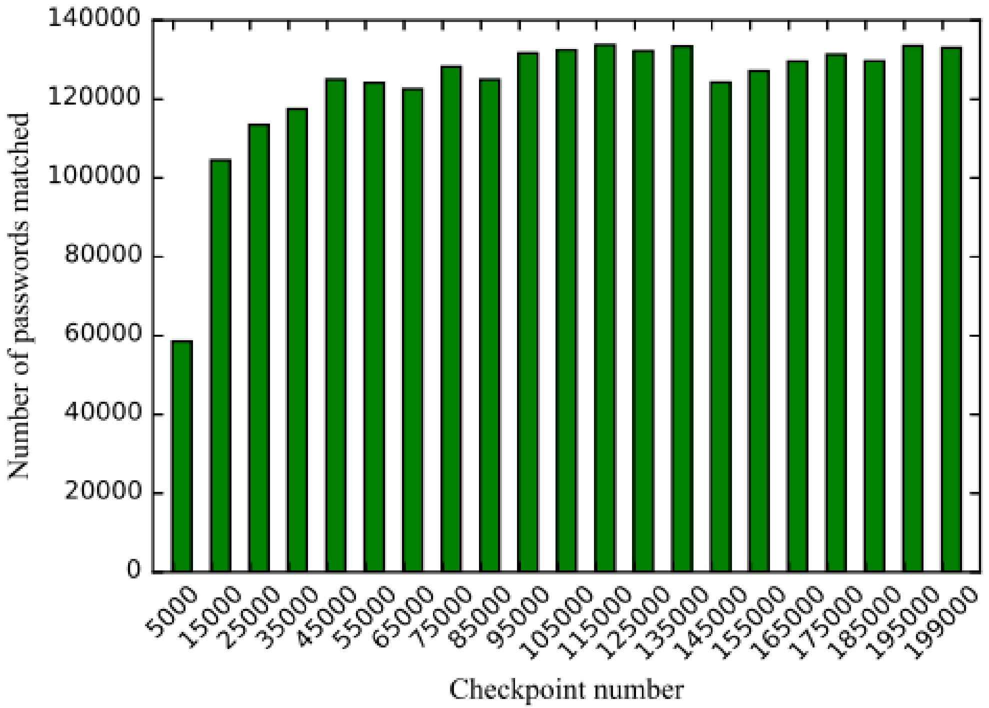



```
joray97
andariy
sh15t3
imalila
tacten6
searilc50
yj053052
ri392012
gilereltv
janleras19
0920mon
haceonal
jeshani
t70mo09
rjsb399
mi10510
j23cme9h
ye2mey2
brwosbas3
acman67o
```

Evaluation (3/6)

Passwords Generated	Unique Passwords	Passwords matched in testing set, and not in training set (1,978,367 unique samples)
10^4	9,738	103 (0.005%)
10^5	94,400	957 (0.048%)
10^6	855,972	7,543 (0.381%)
10^7	7,064,483	40,320 (2.038%)
10^8	52,815,412	133,061 (6.726%)
10^9	356,216,832	298,608 (15.094%)
10^{10}	2,152,819,961	515,079 (26.036%)
$2 \cdot 10^{10}$	3,617,982,306	584,466 (29.543%)
$3 \cdot 10^{10}$	4,877,585,915	625,245 (31.604%)
$4 \cdot 10^{10}$	6,015,716,395	653,978 (33.056%)
$5 \cdot 10^{10}$	7,069,285,569	676,439 (34.192%)

Number of passwords generated by PassGAN that match passwords in the RockYou testing set.



Number of unique passwords generated on various checkpoints matching the RockYou testing set for 10^8 password samples

Evaluation (4/6)

- Is PassGAN able to meet the performance of the other tools despite its lack of any a-priori knowledge on password structures?

Approach	(1) Unique Passwords	(2) Matches	(3) Number of passwords required for PassGAN to outperform (2)	(4) PassGAN Matches
JTR Spyderlab	10^9	461,395 (23.32%)	$1.4 \cdot 10^9$	461,398 (23.32%)
Markov Model 3-gram	$4.9 \cdot 10^8$	532,961 (26.93%)	$2.47 \cdot 10^9$	532,962 (26.93%)
HashCat gen2	10^9	597,899 (30.22%)	$4.8 \cdot 10^9$	625,245 (31.60%)
HashCat Best64	$3.6 \cdot 10^8$	630,068 (31.84%)	$5.06 \cdot 10^9$	630,335 (31.86%)
PCFG	10^9	486,416 (24.59%)	$2.1 \cdot 10^9$	511,453 (25.85%)
FLA $p = 10^{-10}$	$7.4 \cdot 10^8$	652,585 (32.99%)	$6 \cdot 10^9$	653,978 (33.06%)

- Similar trend is observed for LinkedIn testing set

- PassGAN has an advantage when guessing passwords from a dataset different from the one it was trained on.

Evaluation (5/6)

- Idea:- Use the output of multiple tools in order to combine the benefits of rule-based tools and ML-based tools
- Here PassGAN is combined with HashCat Best64

Removed all passwords matched by HashCat Best64 from the RockYou and LinkedIn testing sets. → New test sets, containing 1,348,300 (RockYou) and 33,394,178 (LinkedIn) passwords → Match 51% of passwords from the "new" RockYou dataset, and 73% passwords from the "new" LinkedIn dataset.

- Combining rules with machine learning password guessing is an effective strategy.
- PassGAN can capture portions of the password space not covered by rule-based approaches.

Evaluation (6/6)

➢ Comparison between PassGAN and FLA in terms of probability densities and password distribution

(a) RockYou Training Set

Password	Number of Occurrences in Training Set	Frequency in Training Set
123456	232,844	0.9833%
12345	63,135	0.2666%
123456789	61,531	0.2598%
password	47,507	0.2006%
iloveyou	40,037	0.1691%
princess	26,669	0.1126%
1234567	17,399	0.0735%
rockyou	16,765	0.0708%
12345678	16,536	0.0698%
abc123	13,243	0.0559%
nicole	12,992	0.0549%
daniel	12,337	0.0521%
babygirl	12,130	0.0512%
monkey	11,726	0.0495%
lovely	11,533	0.0487%
jessica	11,262	0.0476%
654321	11,181	0.0472%
michael	11,174	0.0472%

(b) FLA

Password	Rank in Training Set	Frequency in Training Set	Probability assigned by FLA
123456	1	0.9833%	2.81E-3
12345	2	0.2666%	1.06E-3
123457	3,224	0.0016%	2.87E-4
1234566	5,769	0.0010%	1.85E-4
1234565	9,692	0.0006%	1.11E-4
1234567	7	0.0735%	1.00E-4
12345669	848,078	0.0000%	9.84E-5
123458	7,359	0.0008%	9.54E-5
12345679	7,818	0.0007%	9.07E-5
123459	8,155	0.0007%	7.33E-5
lover	457	0.0079%	6.73E-5
love	384	0.0089%	6.09E-5
223456	69,163	0.0001%	5.14E-5
22345	118,098	0.0001%	4.61E-5
1234564	293,340	0.0000%	3.81E-5
123454	23,725	0.0003%	3.56E-5
1234569	5,305	0.0010%	3.54E-5
lovin	39,712	0.0002%	3.21E-5

(c) PassGAN

Password	Rank in Training Set	Frequency in Training Set	Frequency in PassGAN's Output
123456	1	0.9833%	1.0096%
123456789	3	0.25985%	0.222%
12345	2	0.26662%	0.2162%
iloveyou	5	0.16908%	0.1006%
1234567	7	0.07348%	0.0755%
angel	33	0.03558%	0.0638%
12345678	9	0.06983%	0.0508%
iloveu	21	0.04471%	0.0485%
angela	109	0.01921%	0.0338%
daniel	12	0.0521%	0.033%
sweety	90	0.02171%	0.0257%
angels	57	0.02787%	0.0245%
maria	210	0.01342%	0.0159%
loveyou	52	0.0287%	0.0154%
andrew	55	0.02815%	0.0131%
123256	301,429	0.00003%	0.013%
iluv!u	–	–	0.0127%
dangel	38,800	0.00018%	0.0123%

➢ The most likely samples from PassGAN exhibit closer resemblance to the training set and its probabilities than FLA does.

Shortcomings & Performance Enhancements

Approach Shortcomings	➢ Outputs a more significant number of passwords to achieve the same result as rule-based tools.
Performance Enhancements	➢ Training PassGAN on a larger dataset ➢ Changing the generative model behind PassGAN to a conditional GAN might improve password guessing in all scenarios in which the adversary knows a set of keywords commonly used by the user.

Problem Mitigation

Using Honeywords [9]

Honey-words (false passwords) are associated with each user's account.

- An adversary who steals a file of hashed passwords and inverts the hash function cannot tell if he has found the password or a honeyword.
- The "honey-checker" can distinguish the user password from honey-words for the login routine, and will set off an alarm if a honey-word is submitted.

Using own model

Generating own model and check user's password against generated lists

Human like passwords

Treating all human-like passwords as insecure

- This requires classification of human likeliness

[9]. P. B. Shamini, E. Dhivya, S. Jayasree and M. P. Lakshmi, "Detection and avoidance of attacker using honey words in purchase portal," 2017 Third International Conference on Science Technology Engineering & Management (ICONSTEM), Chennai, 2017, pp. 260-263.

Conclusion

- Character-level GANs are well suited for generating password guesses.
- Current rule-based password guessing is very efficient but limited.
- The main downside of rule-based password guessing is that rules can generate only a finite, relatively small set of passwords. In contrast, PassGAN was able to eventually surpass the number of matches achieved using password generation rules.
- The best password guessing strategy is to use multiple tools.
- GANs generalize well to password datasets other than their training dataset.

Q & A

Stay Safe

PassGAN: A Deep Learning Approach for Password Guessing

Yohannes B. Bekele
March 2020