Enabling secure passwords via Deep Learning: Towards a new generation of attacks and defenses

Dario Pasquini
ID number 1421784

Ph.D. Thesis

Thesis Advisor: **Massimo Bernaschi**

PhD School in Computer Science
Department of Computer Science
Sapienza University of Rome; Italy

SAPIENZA
UNIVERSITÀ DI ROMA

A thesis submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science

To S. and M.—who came too late and left too soon.

Abstract

In the present thesis, we aim at alleviating the inherent limitations affecting current solutions in password security. First and foremost, this process requires to devise adversary models that accurately describe real-world guessing attacks. Then, it necessitates the implementation of techniques that are capable of guiding users to choose secure and usable passwords at composition time.

Unfortunately, despite more than three decades of active research dedicated to define and improve these methodologies, existing approaches still present two major drawbacks: (1) current adversary models rely on simplistic adversarial behaviors that only imperfectly describe the guessing strategies adopted by real-world attackers; (2) existing proactive techniques such as password strength meters, by construction, are unable to fully support users during the password composition process.

Here, we show how **Deep Learning** techniques allow us to define novel approaches, that were either unfeasible or unpractical before and that move towards addressing those issues:

(1) We introduce **dynamic adversary models** in password guessing. Similarly to real-world adversaries, dynamic models automatically adjust their guessing strategy for the current attacked-set of passwords by exploiting information collected during the running attack.

(2) We introduce new guessing techniques that make **dictionary attacks consistently more resilient to inadequate configurations**. This novel framework allows dictionary attacks to self-heal and move towards optimal attacks' performance, requiring no supervision.

(3) We introduce **Interpretable Probabilistic Password Strength Meters**. This novel class of meters exhibits a natural and general feedback mechanism capable of describing to the users the latent relation between password strength and password structure. Unlike existing heuristic constructions, this method is free from any human bias, and, more importantly, its feedback has a clear probabilistic interpretation.

Eventually, these general techniques allow us to increase the rigorousness and reliability of password security analysis and proactive methodologies that stem on top of them.

Contents

Chapter 1

Introduction

In spite of their known security issues, passwords remain, by large, the most widely used authentication mechanism [34]. As a matter of fact, textual passwords have dominated human-computer authentication [36] and will probably continue to dominate it in the foreseeable future. Unfortunately, users tend to select their passwords as easy-to-remember sequence of characters, which result in very skewed distributions that can be easily modeled by an attacker. This makes passwords, and authentication systems that implement them, inherently susceptible to guessing attacks.

In this context, a crucial step to protect the security and privacy of users is to help them in choosing passwords that can be hardly guessed. It is apparent that this process requires to (1) devise adversary models that accurately describe real-world guessing attacks; and then, (2) provide proactive techniques that are capable of guiding users to choose secure and usable passwords at composition time. However, despite more than three decades of active research, both current password models and proactive approaches suffer of inherent limitations that prevent them from fully accomplishing their tasks.

The most pervasive limitation of current adversary models is that they apply the same static guessing strategy on each attacked-set of passwords, ignoring trivial information that can be either *a priori* collected or gained from the running attack. However, as widely documented [33, 46, 84], passwords composition habits change from sub-population to sub-population, and, although passwords tend to follow the same general distribution, credentials created under different environments exhibit unique biases. Real-world attackers exploit such biases and perform their guessing attacks incorporating prior knowledge on the targets and dynamically adjusting their guesses during the attack [108]. This allows them to drastically boost attacks' effectiveness and, ultimately, compromise accounts that would not be compromised otherwise. Eventually, the inadequacy of current approaches to model realistic adversarial behaviors prevents them from soundly estimating password strength and fundamentally biases the conclusion of password security analysis.

Additionally, current adversary models mostly rely on probabilistic password models that are only imperfect descriptions of real-world guessing attacks. The main reason is that real attackers would rather rely on more pragmatic methods such as **dictionary attacks**. However, correctly modeling and simulating this class of attacks in password security studies is inherently difficult. In order for dictionary attacks to be representative of the real-world threat, they must be thoughtfully configured and tuned according to a process that requires domain-knowledge and expertise that cannot be easily replicated by researchers and security practitioners. Ultimately, the consequence of inaccurately

calibrating dictionary attacks is a quite limited reliability of password security analyses, due to a strong measurement bias. In this direction, developing techniques capable of automatically producing or simulating expert attack setups remains a critical [76] as well as open problem [108].

For similar reasons, also proactive solutions such as password strength meters suffer of evident shortcomings. These tools aim at evaluating the security of candidate passwords and preventing users from choosing insecure credentials at composition time. In the process, password meters should be able to support and guide users towards secure passwords by implementing explanatory mechanisms. Unfortunately, state-of-the-art password meters base their estimates on blackbox parametric probabilistic models that leave no room for interpretation of the evaluated passwords; they do not provide any **natural form of feedback** to users on what is wrong with their password or how to improve it. In this direction, enabling **interpretability** in password strength meters is one of the core challenges in password security. Unfortunately, while solutions have been proposed [104, 115] and carefully implemented, these lack of generality and soundness. Solely by relying on hardwired, heuristic mechanisms that address only a few common feedback scenarios, they fail, by construction, to give full support to the user during the composition process.

1.1 Thesis contributions

In the present thesis, we demonstrate how deep learning techniques allow us to reduce the most inherent limitations that affect reactive and proactive approaches in password security, moving towards a new generation of methodologies.

Dynamic password models: Basing on deep generative models [58, 103] and the representation learning paradigm [29], we account for **dynamic** attackers in password guessing. Here, we devise a technique that is capable of dynamically modifying and improving itself by relying on the information recovered from the passwords guessed during the attack [94]. We achieved this by exploiting the smoothness and the geometric properties exhibited by latent representation of passwords learned from GANs [58] and congruent models. With these techniques, we are the first to implement and demonstrate the effectiveness of automatic, dynamic attacks in password guessing. In a dynamic attack, the password model autonomously adapts to the attacked distribution of passwords by generalizing the information gained by the passwords recovered within the running attack. Eventually, relying on these techniques, we demonstrate that the strength of a password is not a static property, but it is a function of the other passwords that occur in the same environment. The reported evidence change our understanding of password security and suggest us to rethink current approaches aimed at estimating it.

Dynamic and Adaptive Dictionary attacks: Relying on the same general intuition, we increase the reliability of dictionary attacks by introducing and validating the concept of dynamic dictionaries. Congruently, we use deep learning networks to measure functional compatibility between dictionary words and mangling rules for simulating the proficiency of expert attack configurations. Combining these two complementary approaches, we cast a new generation of dictionary attacks that is consistently more resilient to inaccurate configurations and that reduces measurement bias in modeling password strength.

The proposed attack permits to assign sounder strength estimates to passwords as they better describe real-world adversarial strategies and their threat.

Interpretable Probabilistic Password Strength Meters: Finally, by extending our work on generative models, we introduce probabilistic interpretable mechanisms in password strength meters. We devise and implement the first password strength meter capable of producing a general and rigorous feedback mechanism. For the first time, we demonstrate that the class of probabilistic password meters inherently owns the capability of describing the latent relation between password strength and password structure. Unlike existing approaches, the proposed method is completely free from any human bias, and, more importantly, its feedback has a probabilistic interpretation. By leveraging this approach, we cast a new class of interpretable password meters that have the natural capability of smoothly guiding users to choose stronger passwords and improve their understanding of password security.

1.2 List of publications

The present thesis is based on research papers that have been previously published. This Section briefly lists such contributions, while Section 1.2.1 covers works that have not been included in the thesis.

(Papers are listed following the order on which they are covered in the thesis)

- [94] **Dario Pasquini**, Ankit Gangwal, Giuseppe Ateniese, Massimo Bernaschi, Mauro Conti. *Improving Password Guessing via Representation Learning.* `In 42th IEEE Symposium on Security and Privacy (S&P21)`, May 2021.

In this paper, we demonstrate that representation learning [29] enabled by deep generative models can be used to cast novel attacks that further threaten password-based authentication systems. More importantly, the paper firstly introduces the concept of **dynamic password guessing** and dynamic attackers that will be extended in [93].

- [93] **Dario Pasquini**, Marco Cianfriglia, Giuseppe Ateniese, Massimo Bernaschi. *Reducing Bias in Modeling Real-world Password Strength via Deep Learning and Dynamic Dictionaries.* `In 30th USENIX Security Symposium (USENIX21)`, August 2021

In this work, we increase the rigorousness and reliability of password strength estimates via dictionary attacks. We achieved this by introducing deep learning approaches in mangling rules attacks and advanced techniques aimed at replicating dynamic attackers.

- [91] **Dario Pasquini**, Giuseppe Ateniese, Massimo Bernaschi. *Interpretable Probabilistic Password Strength Meters via Deep Learning.* `In 25th European Symposium on Research in Computer Security (ESORICS20)`, September 2020.

In this work, we introduce a new estimation process for password distributions that enables the creation of the first probabilistic interpretable password strength meter. The proposed meter is based on a complete, undirect description of password probability implemented via deep convolutional neural networks.

1.2.1 Additional publications

Adversarial Machine Learning

A second part of our research, which is not reported in this thesis, has been dedicated to asses the security of deep learning models and distributed training procedures.

- [92] **Dario Pasquini**, Giuseppe Ateniese, Massimo Bernaschi. *Unleashing the Tiger: Inference Attacks on Split Learning* `ACM Computer and Communications Security (CCS21)`, (to appear) November 2021.

In this paper, we investigate the security of *split learning*—an emerging collaborative machine learning framework that enables peak performance by requiring minimal resources consumption. Here, we make explicit the vulnerabilities of the protocol and demonstrate its inherent insecurity.

- [95] **Dario Pasquini**, Marco Mingione, Massimo Bernaschi. *Adversarial out-domain examples for generative models.* `In 2019 IEEE European Symposium on Security and Privacy Workshops, EuroS&P Workshops`, 2019

In this paper, we show the existence of an attack against deep generative models such as Generative Adversarial Networks (GAN). The attack demonstrates that an adversary can force a trained generative model to reproduce arbitrary data instances as output by casting suitable adversarial input. In the paper, we show how this adversarial input can be forged to be indistinguishable from legit inputs.

Numeric methods on GPUs

A third part of our research, which is not reported in this document, has been dedicated to develop numeric methods on GPUs. This line of research primarily resulted in:

- [30] Massimo Bernaschi, Pasqua D'Ambra, **Dario Pasquini**. *AMG based on compatible weighted matching for GPUs.* `Parallel Computing`, 2020.

- [31] Massimo Bernaschi, Pasqua D'Ambra, **Dario Pasquini**. *BootCMatchG: an Adaptive Algebraic MultiGrid Linear Solver for GPUs.* `Software Impacts`, 2020

These papers present an *Algebraic Multigrid* (AMG) method designed and implemented to run on NVIDIA's GPUs. AMG methods are used to reduce the number of iterations required for the numerical solution of very large and sparse linear systems of equations using a multilevel strategy of relaxation and coarse-grid correction. The implemented AMG method relies on a new approach for coarsening sparse symmetric positive definite (spd) matrices, "*named coarsening based on compatible weighted matching*". It exploits maximum weight matching in the adjacency graph of the sparse matrix, driven by the principle of compatible relaxation, providing a suitable aggregation of unknowns which goes beyond the limits of the usual heuristics applied in the current methods. We adopt an approximate solution of the maximum weight matching problem, based on a recently proposed parallel algorithm, referred as the *Suitor* algorithm, and show that it allow us to obtain good quality coarse matrices for our AMG on GPUs. We exploit inherent parallelism of modern GPUs in all the kernels involving sparse matrix computations both for the setup of the preconditioner and for its application in a Krylov solver, outperforming

preconditioners available in Nvidia AmgX library. We report results about a large set of linear systems arising from the discretization of partial differential equations (PDEs).

In [30, 31] we implemented a single GPU version of the software, a distributed version of the latter is in its final phase of development and will be released soon.

1.3 Thesis Organization

The thesis starts in Chapter 2, where we give a brief overview on the fundamental concepts needed for the comprehension of the various topics covered within the thesis. Then, the core contributions are organized in two disjointed parts i.e., Part I and Part II. In Part I , we introduce novel adversary models that are based on deep learning techniques. This includes two works—[94] and [93] which are reported in Chapter 3 and Chapter 4 respectively. Part II, instead, is dedicated to proactive mechanisms enabled via deep learning. This is mainly based on the paper [91] which is covered in Chapter 5. The thesis terminates with the conclusion in Chapter 6, although additional resources are reported in the Appendices.

Chapter 2

Background

In the present Chapter, we cover general concepts that are essential to a better understanding of the various contributes that follow. We start by providing core concepts in password security. Then, in Section 2.2 we present a brief introduction to neural networks and deep learning, mainly focusing on deep generative models. When necessary, additional and specific background is reported within the respective Chapters of Part I and Part II of the thesis.

2.1 Password security

Human-chosen passwords do not distribute uniformly in the highly dimensional space of the possible passwords. Users tend to choose easy-to-remember sequence of characters that aggregate in relatively few dense zones of the passwords space. Real-world passwords, therefore, tend to cluster in very bounded distributions that can be modeled by an attacker, making authentication-systems intrinsically susceptible to **guessing attacks**. In a guessing attack, the attacker aims at recovering plaintext credentials by attempting several candidate passwords (guesses) till success or *budget* exhaustion; this happens by either searching for pre-images of password hashes (**offline attack**) or attempting remote logins (**online attack**). In this process, the attacker relies on a so-called **password model** that defines which, and in which order, guesses should be tried to maximize the effectiveness of the attack.[1]

2.1.1 Guessing Attacks

Offline attack In an offline guessing attack, the attacker has been capable of leaking a set of stored passwords from a compromised system; typically, a web application. This can be achieved through various vectors. Very often, password leaks are enabled by SQL-injections or similar vulnerabilities affecting the application layer. As a matter of fact, passwords leaks occur very frequently [8] and represent a critical security threat due to the users' password reuse habits [64, 44].

Once gained access to the set of stored passwords $X = \{h(x_1), \dots, h(x_n)\}$, the attacker starts the offline attack relying on his/her computational resources. In the most common

[1] In the present thesis, we use the term *password model* to refer to all kinds of approximation of the password generative process; not only the probabilistic ones that are often referred with such name in literature.

case, the leaked passwords are stored under a one-way function h.[2] Targeting a password hash h_x, the attacker tries to recover the original pre-image by applying h on suitable guesses until a collision $h(g) = h_x$ is found. When no additional information is maintained on the single attacked users, this results in a so-called *trawling attack*, where each guess g is hashed and compared to each entry in X.

To prevent the attacker from speeding-up the attack using a suitable pre-computation process (e.g., rainbow tables), passwords are hashed with a *salt*—a random token e.g.,

$$h_x \;=\; h(x + s),$$

where "+" refers to the string concatenation operation and s is a sampled salt. The use of salt also slows down trawling attacks as the attacker has to recompute the hash function for each comparison against X, incrementing the attack cost of a factor $\mathcal{O}(|X|)$.

The adopted h used to store the passwords plays a critical role in reducing the adversarial capabilities in offline attacks. Indeed, the computational cost of computing h is the only limiting factor that decides how many guesses the attackers can perform and thus, how many passwords they can recover. In this direction, iterative application of hash functions (i.e., work factor) or "*bruteforce-aware*" hash functions can be used to artificially increase the computational cost of h or limit its parallelization [97, 98, 66]. Typically, these solutions come with adjustable parameters that allow the passwords owner to define the required computational burned to the attacker. However, this must be carefully chosen as to not limit the service usability and/or enable further vulnerabilities in the system (e.g., denial of service attacks for web applications).

Online attack Online guessing attacks require the attacker to verify the attempted guesses using the standard login procedure deployed from the attacked service. This strongly slows down the attack procedure and puts an hard limit on the number of guesses the attacker can perform. Indeed, it has become a common practice to implement blocking (i.e., deny login after a certain number of failed authentications) and throttling mechanisms (i.e., slow down the login procedure after each failed authentication) in remote authentication systems.

In the online scenario, attackers cannot soundly rely on general trawling guessing strategies used in offline attacks. They have to resort to more tailored attacks such as targeted attacks. Here, the attacker exploits additional information on the specific targeted user such as username or previously leaked passwords [90, 112] to shrink the hypothesis space and maximize the probability of guessing the correct password in the first few guesses.

In this thesis, we mainly focus on studying offline attacks.

2.1.2 Password models

Generally speaking, a password model can be understood as a suitable approximation of the password distribution that enables an educated exploration of the key-space. Existing password models construct over a heterogeneous set of assumptions and rely on either intuitive or rigorous security definitions. From a very practical point of view, we divide those into two macro-classes: namely, parametric and nonparametric password models.

[2]Differently, passwords can be encrypted with a symmetric encryption scheme such as in the cases of password vaults. Here, the guessing attack is on the encryption key that is a low-entropy key as well. However, in this case, honey encryption [41] can be used to slowdown the attacker.

Parametric approaches

Parametric approaches typically build on top of a pure probabilistic reasoning; they assume real-world password distributions sufficiently smooth to be accurately described by suitable parametric probabilistic models. Here, explicitly [85, 89, 40] or implicitly [62, 94] a password mass function is derived from a set of observable data (e.g., previously leaked passwords) and used to attribute a probability to each element of the key-space. During the guessing attack, guesses are produced by traversing the key-space following the decreasing probability ordering imposed by the modeled mass function. While such class of models can potentially enumerate the whole key-space, during the attack, a reasonable cutoff probability is chosen and only the guesses above that threshold are eventually issued. Implementation-wise, the first form of parametric password model is concretized in the work of Narayanan et al. [89], where a Markov filter was used to reduce the search-space of brute-force attack. Further studies extended such initial contribute by introducing suitable regulation techniques [78, 35] and improved enumeration algorithms [53]. Later, Melicher et al. [85] overcome the expressivity limitation of Markov models[3] by modeling password distribution with a recurrent neural network (RNN). This is an autoregressive, character-level model which enables state-of-the-art accuracy in password strength estimation. However, due to intrinsic computational limitations of the enumeration algorithm, such approach results unsuited for real-world password guessing.

Nonparametric models

Nonparametric models rely on more intuitive constructions which tend to be closer to human reasoning. Generally, those assume passwords as realizations of templates and generate novel guesses by abstracting and applying such templates over tokens dictionaries. These approaches maintain collections of tokens that are either directly given as part of the model configuration (e.g., dictionary for dictionary attacks) or extracted from observed passwords in a setup phase. In contrast with parametric models, these can produce only a limited number of guesses which is a function of the chosen initial configuration.
Dictionary-based attacks and their extensions were among the first forms of nonparametric password models. Among dictionary attacks, the extension with mangling-rules [88] widely demonstrated its effectiveness on the trawling attack scenario [6]. These approaches persist nowadays in the form of highly tuned off-the-shelf software: John The Ripper (JTR) [12] and HashCat [7]. Due to their efficiency and easy customization, these tools are the primary weapons of professional security practitioners [108].

Another class of nonparametric models is the one based on Probabilistic Context-Free Grammars (PCFGs). Weir et al. [114] proposed a technique capable of inferring grammars from a set of observed passwords and use those to cast new password guesses. Conceptually, this approach strongly relates to the mangling rules approach, as it exploits a similar assumption and underlying generative process. Nevertheless, PCFG introduces many advantages over the mangling rules. More prominently, this model can learn password-templates directly from the raw data and have a probabilistic interpretation, although PCFGs may not have full support on the key-space.

Given their relevance in the present thesis, a detailed analysis on dictionary attacks follows.

[3] Due to the Markov property assumed for the underlying stochastic process.

Dictionary Attacks and Mangling Rules

Dictionary attacks can be traced back to the inception of password security studies [101, 88]. They stemmed from the observation that users tend to pick their passwords from a bounded and predictable pool of candidates; common natural words and numeric patterns dominate most of this skewed distribution [100]. An attacker, collecting such strings (i.e., creating a dictionary/wordlist), can use them as high-quality guesses during a guessing attack, rapidly covering the key-space's densest zone. These dictionaries are typically constructed by aggregating passwords revealed in previous incidents and plain-word dictionaries.

While dictionary attacks can produce only a limited number of guesses[4], these can be extended through **mangling rules**. Mangling rules attacks describe password distributions by factorizing guesses in two main components: (1) dictionary-words and (2) string transformations (mangling rules). These transformations aim at replicating users composition behavior such as *leeting* or concatenating digits (e.g., "*pa$$w0rd*" or "*password123*") [65]. Mangling transformations are modeled by the attacker and collected in sets (i.e., rules-sets). During the guessing attack, each dictionary word is extended in real-time through mangling rules, creating novel guesses that augment the guessing attack's coverage over the key-space. Hereafter, we use the terms dictionary attack and mangling rules attack interchangeably.

The most widely known implementations of mangling rules are included in the "*password cracking*" software *Hashcat* [7] and *John the Ripper* [12] (JtR). Here, mangling rules are encoded through simple custom programming languages. *Hashcat* and *JtR* share almost overlapping mangling rules languages, although few peculiar instructions are unique to each tool. However, they consistently differ in the way mangling rules are applied during the attack. *Hashcat* follows a **word-major order**, where all the rules of the rule-set are applied to a single dictionary-word before the next dictionary word is considered. In contrast, *JtR* follows a **rule-major order**, where a rule is applied to all the dictionary words before moving to the next rule. The community behind these software packages developed numerous mangling rules sets that are publicly available.

Despite their simplicity, mangling rules attacks represent a substantial threat in offline password guessing. Mangling rules are extremely fast and inherently parallel; they are naturally suited for both parallel hardware (i.e., GPUs) and distributed setups, making them one of the few guessing approaches suitable for large-scale attacks (e.g., botnets). Furthermore, real-world attackers update their guessing strategy dynamically during the attack [108]. Basing on prior knowledge and the initially matched passwords, they tune their guesses generation process to describe their target set of passwords better and eventually recover more of them. To this end, professionals prefer extremely flexible tools that allow for fast and complete customization. While state-of-the-art probabilistic models fail at that, mangling rules make any form of customization feasible as well as natural.

2.1.3 Password Strength Meters (PSMs)

PSMs are password models aimed to an **explicit** estimation of password strength. These are used to proactively induce secure passwords at composition time, when users choose

[4]The required disk space inherently bounds the number of guesses issued from plain dictionary attacks. Guessing attacks can easily go beyond 10^{12} guesses, and storing such a quantity of strings is not practical.

their passwords. The produced estimates are then returned to the users who are notified of the possible weakness of the chosen password and invited to change their password accordingly. This process is often supported by suitable visual feedback mechanisms that aim to guide users towards secure passwords.

A PSM can be formalized as a scalar function rather than a generative model. This associates each password in the key-space[5] to an estimate of its *strength* i.e., its resistance to a guessing attack. In modern approaches, the concept of password strength mostly overlaps with the concept of *guessability* [67]; that is, *how many guesses an attacker has to perform to guess a specific password using a password model.* Ideally, all the password models can be used as a password meter. However, not all the password models are equally suited in practice. Indeed, given the proactive application of PSMs, these are often developed to run client-side on the user's browser.[6] This requires PSMs to be lightweight and capable of estimating guessability at runtime. In this direction, the most convenient class of password models results being that of explicit, parametric probabilistic models. Relying on probabilistic approaches to estimate password strength defines Probabilistic Password Strength Meters which will be detailed next.

Probabilistic Password Strength Meters (PPSMs)

Probabilistic password strength meters are PSMs that base their strength measure on an explicit estimate of password probability. In the process, they resort to probabilistic models to approximate the probability distribution behind a set of known passwords, typically, instances of a password leak. Having an approximation of the mass function, strength estimation is then derived by leveraging adversarial reasoning. Here, password robustness is estimated in consideration of an attacker who knows the underlying password distribution, and that aims at minimizing the guess entropy [83] of her/his guessing attack. To that purpose, the attacker performs an optimal guessing attack, where guesses are issued in decreasing probability order (i.e., high-probability passwords first). More formally, given a probability mass function $P(\mathbf{x})$ defined on the key-space $\mathbb{X}$, the attacker creates an ordering $\mathbb{X}_{P(\mathbf{x})}$ of $\mathbb{X}$ such that:

$$\mathbb{X}_{P(\mathbf{x})} = [x^0, x^1, \ldots, x^n] \quad \text{where} \quad \forall_{i \in [0,n]} : P(x^i) \geq P(x^{i+1}) \quad . \tag{2.1}$$

During the attack, the adversary produces guesses by traversing the list $\mathbb{X}_{P(\mathbf{x})}$. Under this adversarial model, passwords with high probability are considered weak, as they will be quickly guessed. Low-probability passwords, instead, are assessed as secure, as they will be matched by the attacker only after a considerable, possibly not feasible, number of guesses.

A review of previous works on password strength meters will be given in Chapter 5, when we introduce and discuss interpretable probabilistic password strength meters.

2.2 Deep Learning

This section briefly covers specific topics in the deep learning landscape that appear frequently throughout the thesis. Section 2.2.1 glances neural networks and deep neural

[5]Not all the PSMs define each elements of the key-space. For instance, meter based on PCFG are only able to score passwords that can be represented with the leaned grammar.

[6]Generally, we do not want the plaintext user's password to leave the client-side. Therefore, client-server strength meters are not appealing solutions.

networks as well as discusses specific neural architectures. Then, Section 2.2.2 focuses on deep generative models, introducing them and explicating the formalization used later in the thesis.

2.2.1 On Neural Networks and Architectures

A neural network is a differentiable, non-linear[7], function $f(\cdot \mid \theta)$ defined over a family of parametric functions indexed by the set of parameters/weights θ of the network. The family of functions is defined by the so-called **architecture** of the network that is specified as a sequence of logic partitions called layers i.e., non-linear parametric functions on their own. Deep neural networks, in turn, are functions defined by the composition of many layers. They are a powerful function approximator capable of accurately describing relations among high-dimensional spaces. Once a target function is chosen, a differentiable loss function is defined and used to guide the approximation. This process, named *learning*, consists in finding the configuration of parameters θ that minimize the discrepancy between the target function and the neural network by relying on a gradient-descent-based optimization technique. Furthermore, deep neural networks have vastly demonstrated the peculiar and natural capability of generalizing over the input domain. This mainly relates to the qualities of the data representation learned during the training [29].

Convolutional Neural Networks Most of the neural networks employed in the contributions reported in this thesis are Convolutional Neural Network (CNN). These are networks whose architecture mainly assembles convolutional layers. A convolutional layer is a neural layer that leverages parameters sharing to both reduce the number of learnable parameters of the networks[8] and enforce spatial invariance over the input space which can further help generalization int the suitable domains. In this process, Convolution layers approximate functions by learning suitable kernels that are applied over the input channels. Those permit the accurate description of relations among features occurring in spatial proximity. Such properties make CNNs perfectly suited to learn tasks over the image domain, where space invariance and local, spatial relations are natural as well as crucial at the end of learning suitable data representations. Nonetheless, CNN have been widely applied over sequential data such as text, often performing equally or better than Recurrent Neural Networks (RNN) which are naturally suited for sequential data. Throughout the thesis, we heavily apply CNNs over sequential data (i.e., mainly passwords) as those can offer peculiar advantages over recurrent neural networks: they allow faster inference and permit to drop the autoregressive paradigm that appears in recurrent networks and that may not fully describe relations properties among characters in passwords.

Residual architectures Despite Convolutional layers can be arbitrarily combined to assemble suitable CNNs, modern architectures tend to follow a residual structure [61], creating neural nets called **residual neural networks** (or *resnet* in brief). A *resnet* is composed of a block of layers (namely, residual blocks) presenting **skipping connections**; that is, layers that are not adjacent can be connected via an additional connection

[7]Linear neural networks exist, but they are surely less interesting than the non-linear ones.

[8]This property allows the construction of deep architectures with a limited memory fingerprint.

Algorithm 1: Example of residual block:

Data: input tensor: x_{in}

$x = \texttt{batchNormalization}(x_{in})$;
$x = \texttt{ReLU}(x)$;
$x = \texttt{1D-Convolution}(x, f, k)$;
$x = \texttt{batchNormalization}(x)$;
$x = \texttt{ReLU}(x)$;
$x = \texttt{1D-Convolution}(x, f, k)$;
return $x_{in} + x$

that bridges input features skipping intermediate layers. A residual block typically composes of few convolutional layers alternated with the application of point-wise activation functions and features normalization such as batch-normalization [63]. In the block, input features are then directly point-wised added to the output of the last layer of the block, creating an internal residual connection. An example of convolutional block is reported in Algorithm 1. Residual connections have been proven to improve both the gradient flow and the depth/accuracy ratio of networks [61]. Most of the neural networks used throughout the thesis have a residual architecture based on 1D Convolution layers; that is, convolution layers that apply on mono-dimensional channels (i.e., kernels move over a single dimension).

2.2.2 Deep Generative models

Within the thesis, we rely on different deep learning models. However, the most used class is the one of deep generative models. Hereafter, we intend a deep generative model as a probabilistic model trained to perform implicit estimation of an unknown **target data distribution** $p^*(\mathbf{x})$, given a set of observable data (i.e., a train-set) [58, 57]. In the process, a deep neural network is used to parametrize the description of the underlying data distribution.

In contrast to the common *prescribed probabilistic models* [49], implicit probabilistic models do not explicitly estimate the probability density of data. They instead approximate the stochastic procedure that generates data [87]. In other words, we can sample data points from the model as if they were sampled from a random variable following $p(\mathbf{x})$. However, in general, we cannot directly compute the probability of a given state x_i of $\mathbf{x}$. Normalizing flows based models [50, 70] allow exact density estimation by employing suitable neural architectures.

In the used deep generative models are latent variable models. That is, the network is implicitly guided to learn a set of latent variables that unfold the complex interactions among the factors describing data. During the training, a prior distribution is imposed on the learned latent variables so that we can eventually sample realizations of them after the training. Such a prior, referred to as **prior latent distribution** or $\dot{p}(\mathbf{z})$, is an *easy-to-sample*, uninformative and factorized prior. Its factorized form requires that the network assigns a disjointed and independent semantic meaning to each latent variable, and, consequently, learns a disentangled latent data representation for the input domain. In other words, the latent representation is modeled to capture the posterior distribution of the underlying explanatory factors of the observed data [99].

Once trained, a generative network is a deterministic mapping function $G : \mathbf{Z} \rightarrow \mathbf{X}$ between the latent space $\mathbf{Z} : \mathbb{R}^k$ and the data space $\mathbf{X}$ (i.e., where the observed data is

defined), specifically, **a bridge between $\dot{p}(\mathbf{z})$ and the distribution $p(\mathbf{x})$ learned by the model**. More formally, under this construction, the probabilities of data instances have the following form:

$$p(\mathbf{x}) = p(\mathbf{x} \mid z; \theta)\dot{p}(\mathbf{z}), \tag{2.2}$$

where θ is the set of learnable parameters of the generator. Typical choices for $\dot{p}(\mathbf{z})$ are $\mathcal{N}(0, \mathbf{I})$ or $U[0, 1]$ [57].

Sampling points z from the latent space according to $\dot{p}(\mathbf{z})$ and then mapping them in the data-space through the generator, is equivalent to sampling data points from the data space $\mathbf{X}$ according to $p(\mathbf{x})$. During this operation, we can generally also consider an arbitrary $p(\mathbf{z})$ that can be different[9] from $\dot{p}(\mathbf{z})$. In the rest of this thesis, we will refer to the probability density function $p(\mathbf{z})$ of the latent space with the general term of **latent distribution.**

Additionally, the smoothness of the generator forces a geometric organization in the learned latent space. Similar to the feature embedding techniques [59, 74], indeed, the latent representations of semantically bounded data points show strong spatial coherence in the latent space [99].

In the thesis, two deep generative model frameworks are mainly used, namely, Generative Adversarial Networks and generative models based on the auto-encoding paradigm.

Generative Adversarial Networks (GANs) The GANs framework learns a deep generative model by following an adversarial training approach. The training process is guided by a second network D (i.e., the critic/discriminator), which gives a density estimation-by-comparison [87] loss function to the generative network G (i.e., the generator). The adversarial training bypasses the necessity of defining an explicit likelihood-function and allows us to have a good estimation of very sharp distributions [57].
During the training, latent points z are directly sampled from $\dot{p}(\mathbf{z})$ and given as input to G. In turn, the latter maps those in the data-space, where they are fed to the network D. The critic, receiving both ground-truth data instances from the train-set and generating data from G, is trained to allocate density only to real data instances. The generator G, instead, is adversarially trained to force D to arrange probability estimates on the output of $G(z)$. The optimization follows from a coordinate minimization of the losses of the two networks.

Autoencoders (AE) With the term Autoencoder we can refer to any model that conceptually compounds of two networks: an encoder network $Enc : \mathbf{X} \to \mathbf{Z}$ and a decoder network $Dec : \mathbf{Z} \to \mathbf{X}$ trained to learn a form of identity function: $x = Dec(Enc(x))$, or a more useful variation of it. Unlike GANs, no adversarial training is exploited during the training. Typically, a maximum likelihood approach is used, instead, although adversarial training can be used to force topological properties in the codomain of Enc [81].
Once trained, the network Dec can serve as a data generator where meaningful latent points are fed as input to it. However, to allow for efficient sampling from the latent space, an AE needs a form of explicit regularization during the training; that is, the latent space must be forced to be coherent with a chosen prior latent distribution. Widely known AEs implementing this strategy are described in [69, 81, 103].

In the rest of the thesis, we abstract over the concept of generative model and make no distinction between the decoder network Dec and the GAN generator; we refer to

[9] At a cost of representing a distribution different from $p^*(\mathbf{x})$.

either of them as G. In the same way, we employ E to refer to the encoder network used to model the inverse mapping.

Part I

Adversary Models

As discussed in the introduction of the thesis, the low-entropy inherent in human-chosen passwords makes them naturally susceptible to guessing attacks. Within this context, where theoretical security bounds are unrepresentative, it is crucial to accurately model attackers' capabilities and behaviors, as this is the only way we have to discriminate what is secure from what is not.

In this part of the thesis, we demonstrate how deep learning techniques can be used to model powerful attackers and disclose novel threats that further jeopardize the security of passwords. These techniques allow us to grasp a more accurate understanding of real-world adversary and automatize their attack strategies, enabling guessing attacks that more soundly estimate password security.

We start in Chapter 3, by introducing novel guessing techniques based on the representation learning paradigm.

Chapter 4 extends the intuitions presented in Chapter 3 by applying deep learning models on dictionary attacks. With those, we increase the rigorousness and reliability of password strength estimates via dictionary attacks which are the kind of attack that is the most employed by real-world adversaries.

Chapter 3

Guessing attacks based on Representation Learning

As discussed in Section 2.1.3, real-world password distributions are typically composed of several dense zones that can be feasibly estimated by an adversary to perform password-space reduction attacks [118]. Along that line, several probabilistic approaches have been proposed [85, 53, 114]. These techniques—under different assumptions—try to directly estimate the probability distribution behind a set of observed passwords. Such estimation is then used to generate suitable guesses and perform efficient password guessing attacks.

Orthogonal to the current lines of research, we demonstrate that an adversary can further expand the attack opportunities by leveraging representation learning techniques [29]. Representation learning aims at learning useful and explanatory representations [29] from a massive collection of unstructured data. By applying this general approach on a corpus of leaked passwords [21], we demonstrate the advantages that an adversary can gain by learning a suitable representation of the observed password distribution, rather than directly estimating it. In this chapter, we show that this type of representation allows an attacker to establish novel password guessing techniques that further threaten password-based authentication systems.

We model the representation of passwords in the latent space of (1) an instance of *Generative Adversarial Networks* (GANs) [58] generator and (2) an instance of *Wasserstein Auto-Encoders* (WAEs) [103]. This type of representation, thanks to its inherent smoothness [29], enforces a semantic organization in the high-dimensional password space. Such an organization mainly implies that, in the latent space of the generator, respective representations of semantically-related passwords are closer. As a result, geometric relations in the latent space directly translate to semantic relations in the data space. A representative example of this phenomenon is loosely depicted in Figure 3.1, where we show some latent points (with their respective plain-text passwords) localized in a small section of the induced latent space.

We exploit such geometric relations to perform a peculiar form of conditional password generation. Namely, we characterize two main properties: *password strong locality* and *password weak locality*. These locality principles enforce different forms of passwords organization that allow us to design two novel password guessing frameworks, *Conditional Password Guessing (CPG)* and *Dynamic Password Guessing (DPG)*. We emphasize that state-of-the-art approaches are unable to perform such types of advanced attacks or, if somehow altered, become very inefficient. The major contributions of our work are as

Figure 3.1: A small section of the induced latent space around the latent point for the password "rockyou".

follows:

1. We are the first to demonstrate the potential of using fully unsupervised representation learning in the password guessing domain.

2. We introduce a probabilistic and completely unsupervised form of template-based passwords generation. We call this framework CPG. CPG generates arbitrarily biased passwords that can be used: (1) by an adversary to increase the impact of side channels and similar password attacks [27, 82, 110, 28]; or (2) by a legitimate user to recover his/her password. We show the efficiency of CPG against existing solutions via experimental evaluations.

3. We introduce the concept of DPG: DPG is the password guessing approach that dynamically adapts the guessing strategy based on the feedback received from the interaction with the attacked passwords set. We build an Expectation Maximization-inspired DPG implementation based on the password locality enforced by the deep generative model. DPG shows that an attacker can consistently increase the impact of the attack by leveraging the passwords successfully guessed during a running attack.

It is important to highlight that ongoing developments in deep generative frameworks would naturally translate into further improvements in our approach.

Organization: In Section3.1, we present our model improvements and the tools upon which our core work is based. We introduce password locality along with CPG in Sec-

tion 3.2 and DPG in Section 3.3. The evaluation of our proposed techniques is presented in their respective sections.

3.1 Password guessing with Deep Generative Models

Hitaj et al. in their seminal work PassGAN [62] trained a GAN generator as an implicit estimator of password distributions. PassGAN harnesses an off-the-shelf Wasserstein GAN with gradient penalty [60] over a residual-block-based architecture [61]. It assumes a latent space with a standard normal distribution as its prior latent distribution and dimensionality equal to 128. The model is trained on the RockYou [21] password leak, and only passwords with 10 or fewer characters were considered. Despite its underlying potential, the password guessing approach presented in PassGAN suffers from technical limitations and inherent disadvantages in its application.[1] Most limitations can be addressed as shown in Section 3.1.1. However, some limitations are intrinsic to the model itself. A prominent example is the model's inability to assign probabilities to the produced guesses consistently and thus sort them based on popularity. This drawback might make the GAN approach undesirable in a standard trawling scenario. However, in the present Chapter, we show the existence of novel and valuable properties intrinsic to the class of deep generative models. Abstracting the underlying model under the perspective of representation learning, we prove that these properties can be used to devise unique guessing techniques that are infeasible with any existing approaches.

Next, we introduce the necessary improvements to the original PassGAN construction (Section 3.1.1). In Section 3.1.2, we introduce a different and novel deep generative model in the password guessing domain.

3.1.1 Improved GAN model

The password guessing approach presented in PassGAN suffers from an inherent training instability. Under such conditions, the generator and the critic cannot carry out a sufficient number of training iterations. This may lead to an unsuitable approximation of the target data distribution and reduced accuracy in the password guessing task. In the original model, the discrete representation of the strings (i.e., passwords) in the train-set[2] introduces strong instability for two main reasons: (1) The discrete data format is very hard to reproduce for the generator because of the final *softmax* activation function, which can easily cause a low-quality gradient; and (2) the inability of the generator to fully mimic the discrete nature of the train-set makes it straightforward for the critic to distinguish between real and generated data. Hence, the critic can assign the correct *"class"* easily, leaving no room for an enhancement of the generator, especially in the final stages of the training.

To tackle the problems above, we apply a form of stochastic smoothing over the representation of the strings contained in the train-set. This smoothing operation consists of applying an additive noise of small magnitude over the one-hot encoding representation of each character. The smoothing operation is governed by a hyper-parameter γ, which defines the upper-bound of the noise's magnitude. We empirically chose $\gamma = 0.01$

[1] As a matter of fact, PassGAN requires up to ten times more guesses to reach the same number of matched passwords as the probabilistic and non-probabilistic competitors.

[2] Each string is represented as a binary matrix obtained by the concatenation of the one-hot encoded characters.

Number guesses	PassGAN (%)	Our GAN (%)
$1 \cdot 10^8$	6.72	9.51
$1 \cdot 10^9$	15.09	23.33
$1 \cdot 10^{10}$	26.03	40.48
$2 \cdot 10^{10}$	29.54	45.55
$3 \cdot 10^{10}$	31.60	48.40
$4 \cdot 10^{10}$	33.05	50.34
$5 \cdot 10^{10}$	34.19	51.80

Table 3.1: The matched passwords by PassGAN and our improved model over the RockYou test-set

and re-normalize each distribution of characters after the application of the noise. This smoothing operation has a significant impact on the dynamics of the training, allowing us to perform 30 times more training iterations without training collapse [39]. We keep the general GAN framework mostly unchanged because of the excellent performance of the *gradient-penalty-WGAN* [60].

With our improvements in the training process, we can exploit a deeper architecture for both the generator and the critic. We substitute the plain residual blocks with deeper residual bottleneck blocks [61], leaving their number intact. We find the use of batch normalization in the generator to be essential for increasing the number of layers of the networks successfully.

The new architecture and the revised training process allow us to learn a better approximation of the target password distribution, and consequently, outperform the original PassGAN. A comparison between the original and our improved approach is reported in Table 3.1. In this experiment, both models are trained on 80% of RockYou leak and compared in a trawling attack[3] on the remaining 20% of the set. As the 20% test-set does not contain passwords present in the train-set, the performance of a model in this test demonstrates its ability to generate new valid passwords, excluding overfitting artifacts. Hereafter, we use the improved settings described in the present section. We train three different generators, using a 80-20% split of RockYou leak, considering passwords with a maximum length of 10, 16, and 22, respectively.

3.1.2 Autoencoder for password guessing

To highlight the generality of the proposed approaches, we introduce a second and novel deep generative model for password guessing. It is based on Wasserstein Autoencoder (WAE) [103] with moment matching regularization applied to the latent space (called WAE-MMD [103]). To allow for sampling from the latent space, WAE regularizes the latent space to make it coherent with a chosen prior latent distribution.

A WAE learns a latent representation that shares several properties with the one coming from the GAN-based technique. Nevertheless, these models naturally provide a very accurate inverse mapping, i.e., *Enc*, which makes the model superior to the default GAN-based one in certain scenarios.

To add further regulation to the WAE, we train the model as a Context AE (CAE) [96]. During every iteration of the training process of a CAE, the encoder receives a noisy version $\tilde{x}_i$ of the input password x_i. The noisy input is obtained by removing each of the

[3]Under the same configuration proposed in [62].

characters in the password x with a certain probability $p = \frac{\epsilon}{|x_i|}$ where $|x_i|$ is the number of characters in the password, and ϵ is a hyper-parameter fixed to 5 in our setup. Our model receives the mangled input $\tilde{x}_i$, and then it is trained to reproduce the complete password as the output ($x = Dec(Enc(\tilde{x}))$); that is, the model must estimate the missing characters from the context given by the available ones. Furthermore, the CAE training procedure allows us to contextualize the wildcard character that we will use in Section 3.2.2. We refer to our final model as the *Context Wasserstein Autoencoder*, or CWAE.

We set up the CWAE with a deeper version of the architecture used for the GAN generator. We use the same prior latent distribution of our GAN generator, i.e., $\mathcal{N}(0, \mathbf{I})$ with a dimension of 128. The training process is performed over the same train-sets of the GAN.

3.2 Conditional password guessing (CPG) and Passwords strong locality

In this section, we present the password locality concept, and its possible applications for password guessing. In Section 3.2.1, we describe the most natural form of locality that we call *password strong locality*. In Section 3.2.2, we demonstrate the practical application of password locality by introducing a technique that we call "password template inversion" for conditional and partial knowledge passwords generation. Finally, we demonstrate the advantages that our technique offers over existing probabilistic and non-probabilistic password models.

3.2.1 Password strong locality and localized sampling

As we briefly introduced in Section 2.2.2, the latent representation learned by the generator enforces geometric connections among latent points that share semantic relations in the data space. As a result, the latent representation maintains *"similar"* instances closer.

In general, the concept of similarity harnessed in the latent space of a deep generative model solely depends on the modeled data domain (e.g., images, text) and its distribution. However, external properties can be incentivized by the designer via injection of inductive bias during the training. An example is reported in Appendix A.1. In the case of our **passwords latent representations**, the concept of similarity mainly relies on a few key factors such as the structure of the password, the occurrence of common substrings, and the class of characters. Figure 3.2 (obtained by t-SNE [80]) depicts this observation by showing a 2D representation of small portions around three latent points (corresponding to three sample passwords "`jimmy91`", "`abc123abc`", and "`123456`") in the latent space. Looking at the area with password "`jimmy91`" as the center, we can observe how the surrounding passwords share the same general structure (5L2D, i.e., 5 letters followed by 2 digits) and tend to maintain the substring "jimmy" with minor variations. Likewise, the area with the string "`abc123abc`" exhibits a similar phenomenon, where such a string is not present in the selected train-set and does not represent a common password template.

We loosely name **password strong locality** the representation's inherent property of grouping together passwords that share very fine-grained characteristics. The password strong locality property asserts that latent representation of passwords sharing some

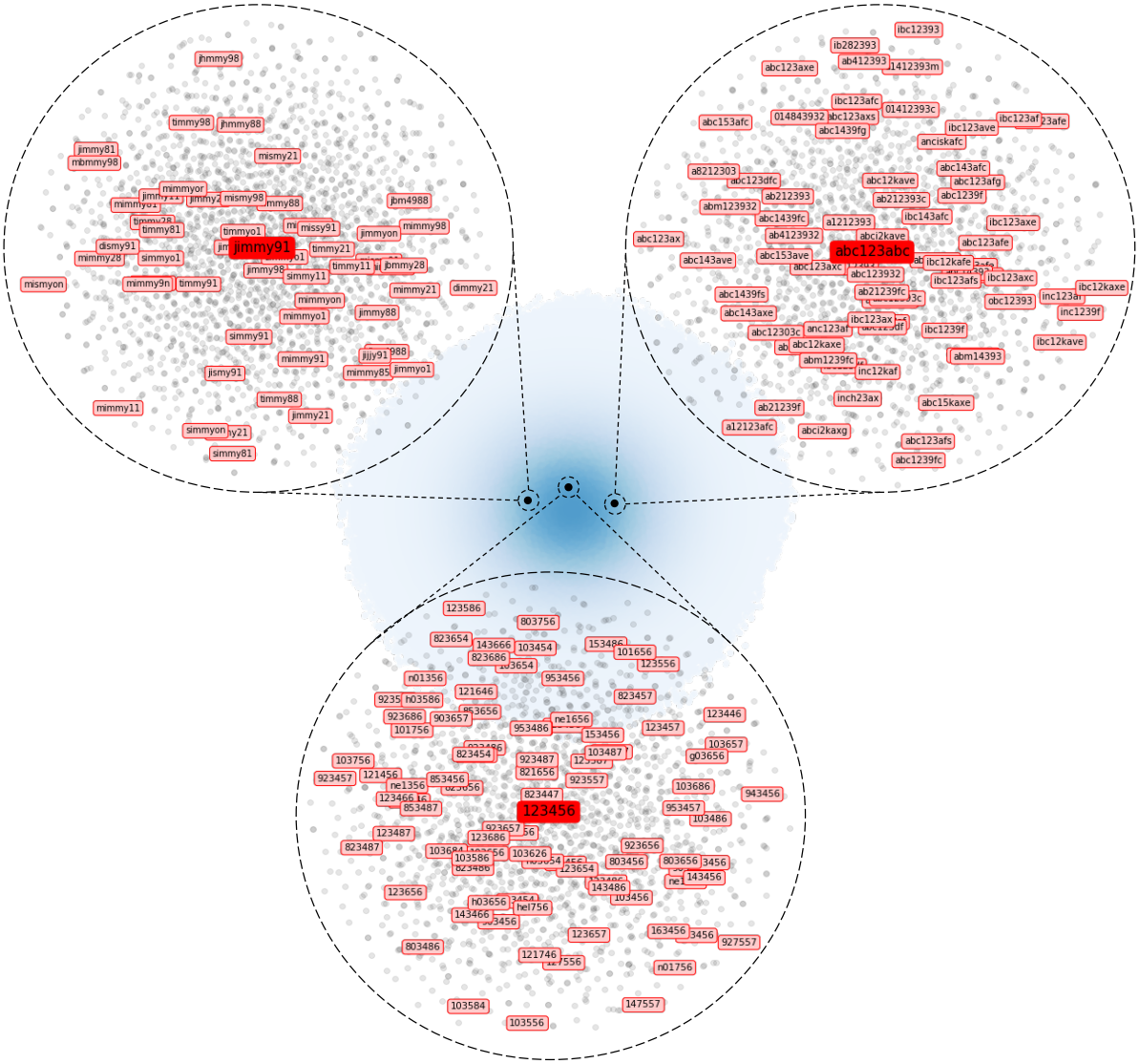


Figure 3.2: 2D representation of small portions around three latent points corresponding to three sample passwords "`jimmy91`", "`abc123abc`", and "`123456`" in the latent space learned from the RockYou train-set. Note: for the sake of better illustration, the image has been cropped.

specific characteristics, such as identical substrings and structure, are organized in close proximity to each other. In Section 3.3.1, we will show that strong locality also implies a weaker but general form of semantic bounding, which we refer to as weak locality.

The strong locality becomes particularly compelling when selecting where to focus on the sampling operation during the password generation process. **Indeed, since different classes of passwords are organized and bounded into different zones of the underlying space, it is possible to generate specific classes of passwords by sampling from specific areas of it.** We leverage this technique to induce arbitrary biases in the generation process.[4] However, we must first define a meaningful and practical way to express such biases, that is, to localize the zones of the latent space we are interested in.

One naive solution resorts to a prototype password x to guide the localization process. In particular, we can generate passwords strictly related to the chosen prototype password x, by fetching latent points around the latent representation z of x (i.e., $x = G(z)$). Thanks to the strong locality, the obtained latent points should be valid latent representations of passwords with an arbitrary strong relation with x. In this context, we refer to the chosen x (or its corresponding latent representation z) with the term **pivot**. The three dark red boxes in Figure 3.2 are the pivot points in the latent space for their corresponding passwords.

To infer the latent representation z from x, we use the encoder network described in Section 3.1.2, as that $z = E(x)$. We highlight that, being this process general and model-independent, other deep generative models such as [51, 69, 81] can be used as well.

Once we obtain the intended pivot z, we can easily generate coherent passwords by restricting the generator's sampling in a confined area of the latent space around z (loosely represented by the small dashed circles in Figure 3.2). To that purpose, we consider a new latent distribution for the generator. The new distribution has the latent representation of the pivot password as its expected value and an arbitrarily small scale. To remain coherent with prior latent distribution and partially avoiding distribution mismatch for the sampled points [116], we chose a Gaussian distribution: $\mathcal{N}(z, \sigma\mathbf{I})$.

According to the concept of password locality, the strength of the semantic relation between a sampled latent point and its pivot should be proportional to the spatial distance between them. Consequently, the chosen value of σ (i.e., standard deviation) offers us a direct way to control the level of semantic bounding existing in the generated passwords. This intuition is better explained by Table 3.2, where passwords obtained with different values of σ for the same pivot password are reported.

Lower values of σ produce highly aligned passwords, whereas larger values of σ allow us to explore areas far from the pivot and produce a different type of *"similar"* passwords. As shown in Table 3.2, all the passwords generated with $\sigma = 0.05$ retained not only the structure of the pivot (i.e., 5L2D), but also observed minor variations coherent with the underlying password distribution. Of note, passwords generated with $\sigma = 0.15$ tend to escape the password template imposed by the pivot and reaching related-but-dissimilar password structures (e.g., "`jimmy91992`" and "`j144988`").

[4] From the model's point of view, this is equivalent to changing the latent distribution, and in particular, reallocating its expected value to a different zone of the latent space.

σ=0.05	σ=0.08	σ=0.10	σ=0.15
jimmy91	jimmy99	mnmm988	jimmy91992
jimmy11	micmy91	tbmmy98	jrm6998
jimmy21	jimsy91	jismyo15	sirsy91
jimmy88	mimmyo1	jizmyon	jrz4988
jimmy81	jbmmy88	j144988	Rimky28
jimmy98	simmy98	jbmm998	missy11
mimmy98	dijmy91	timsy91	jimmy119
jimmy28	jimmy98	jrm4985	sikjy91
simmy91	timsy91	jhmmy88	licky916
mimmy91	jnmm988	jhmm988	gimjyon

Table 3.2: The first-ten passwords obtained with different values of σ starting from the pivot string *"jimmy91"*

3.2.2 Localized passwords generation with password template inversion

As briefly discussed in Section 3.2.1, the password locality property offers a natural way to generate a very specific/confined class of passwords for a chosen pivot, a task accomplished by exploiting an encoder network E. This encoder is trained to approximate the inverse function G^{-1}, and it is the only tool we have to explore the latent space meaningfully. The default behavior of the encoder is to take as an input a string s and precisely localize the corresponding latent representation in the latent space. As shown in Table 3.2, sampling from a distribution centered on the obtained latent point, allows us to generate a set of related passwords. However, this approach alone is not sufficient within the password guessing scenario.

In this section, we show that it is possible to "trick" the encoder network into further localizing general classes of passwords. We can arbitrarily define these classes via a minimal **template**, which expresses the definition of the target password class.

The encoder network can be forced to work around a specific password definition by introducing a **wildcard** character into its alphabet. The wildcard character - represented by the symbol '•' in the present thesis—can be used as a placeholder to indicate an unspecified character. For instance, the template "`jimmy••`" expresses a class of passwords starting with the string "jimmy" followed by two undefined characters. When the encoder inverts this string, the obtained latent point represents the **center** of the cluster of passwords in the latent space with a total length of 8 characters and a prefix "jimmy".

A	B	C	D	E	F	G	H	I	L	M
jimmy••	**jimmy••••**	**••jimmy**	**••mm•91**	**•••••91**	**12•••91**	**A•••••**	**•••A•••**	**Ra••••91**	**(•••1•••)**	**•••#••!!!**
jimmy11	jimmybean	majimmy	summy91	1111991	1231991	Andres	RONALDO	Raider91	(2001999)	123#1!!!!
jimmy13	jimmybear	mujimmy	sammy91	9111991	1211991	ANDRES	MALANIA	Rainer91	(1701939)	tom#!!!!!
jimmy01	jimmy1001	mojimmy	tommy91	a111991	1221991	Andrea	MANANA1	Rain1991	(toe1234)	bom#!!!!!
jimmy12	jimmyjean	myjimmy	tammy91	jan1991	1201991	A10123	SALAN11	Raidel91	(13@1932)	Bom#1!!!!
jimmy10	jimmylove	12jimmy	mommy91	cao1991	1271991	Angela	RATALIS	Ranger91	(gar1k())	Bam#99!!
jimmy20	jimmy2004	jojimmy	jimmy91	ban1991	1234591	A12123	123A123	Rana1991	(1031123)	190#1!!!!
jimmy21	jimmy1234	gojimmy	gimmy91	5121991	1219991	Andrey	BRIANA1	Raid1991	(1231234)	abc#2!!!!
jimmy16	jimmybabe	jjjimmy	iammy91	man1991	1205091	ANDREY	MALA123	Raynay91	(sot123))	123#11!!!'
jimmy19	jimmygirl	aajimmy	mimmy91	1811991	1280791	ABC123	AAIANA1	Rayder91	(Go)12(7)	Bom##!!!!
jimmyes	jimmy1000	m0jimmy	sommy91	jao1991	12g1991	ABERES	BALAND1	RaIN1991	(11_199%)	123#16!!!

Table 3.3: An example of exploiting strong locality property over a generator trained on RockYou train-set for some password templates. Passwords are generated by sampling 10000 strings with $\alpha = 0.8$ and reported in decreasing frequency order.

Therefore, sampling around this latent point allows us to generate good instantiations (according to $p(\mathbf{x})$) of the input template. Column A of Table 3.3 shows an example for the template "`jimmy••`". In practice, we implement this behavior by mapping a wildcard character to an empty one-hot encoded vector when the matrix corresponding to the input string is given to the encoder. The wildcard characters can be placed in any position to define an arbitrarily complex password template; some examples are reported in Table 3.3.

Relying on this technique, the template inversion guides us towards the most plausible zone of the latent space. When we sample from that zone, the wildcards are replaced with high-probability characters according to the distribution $p(\mathbf{x})$, i.e., the probability distribution modeled by the generator. This phenomenon can be observed in the generated samples (Column A of Table 3.3): wildcards in most of the generated passwords have been replaced with digits to conceivably reproduce the frequent password pattern '*lower_ case_ string+digits*' [107]. On the contrary, passwords from the template "`•••••91`" are reported in Column E of Table 3.3. In this example, we ask the generator to find 7-character long passwords where the last two characters are digits. Here, the generated passwords tend to lie towards two most likely password classes for this case, i.e., '*lower_ case_ string+digits*' complementary to the previous case and '*all_ digits.*' As the localized zone of the latent space is a function of all the observed characters, the same template with more observable digits (e.g., Column F of Table 3.3) ends up generating *all_ digits* passwords with higher probability.

3.2.3 Conditional Password Guessing (CPG)

One of the most significant limitations of available probabilistic guessers is their intrinsic rigidity. The inductive bias imposed on such models allows them to be extremely suitable for general trawling attacks, yet it causes them to fail at adapting to different guessing scenarios. For instance, they fail to handle a natural as well as a general form of conditional password generation, such as the template-based one that we proposed in Section 3.2.2. Despite the limitations of existing approaches, generating guesses under arbitrary biases is a useful and helpful procedure. This applies to both security practitioners and common users. Some examples are below:

- An attacker can be interested in generating an arbitrary number of guesses having a particular structure or common substring. For instance, an attacker might want to generate passwords containing the name of the attacked web application as substring.[5]

- A conditional password generation capable of working with partial knowledge can be used by an attacker to improve the impact of side-channel attacks targeting user input [27, 82, 110, 28]. These attacks often recover only an incomplete password (e.g., some characters) due to their accuracy. An attacker can leverage conditional password generation mechanisms to input missing characters and recover the target password.

- Similarly, a legitimate user can be interested in recovering her/his forgotten password while remembering a partial template, for example, "***Jimmy**1**8#".

[5]It has been widely observed that many users tend to incorporate such names in their passwords.

In this direction, conditional password generation is particularly difficult for *autoregressive* password guessers, such as the RNN-based ones (e.g., FLA [85]). Indeed, these approaches, in the general case, are unable to assign a probability to missing characters of a template efficiently; the forward-directionality, intrinsic in their generation process, eliminates the possibility of an efficient appreciation of wildcards occurring before a given substring (e.g., the case in Columns C and E of Table 3.3). In these cases, the probability of an exponential number of passwords could be computed before using the characters in the template to prune the visit tree. This is the case of the template reported in Column E, where the required computational cost for these approaches is not far from computing all the passwords into the chosen probability threshold and filter the ones coherent with the template. More generally, these approaches cannot be efficiently applied when a large number of wildcards is considered. Sampling from the posterior distribution over the missing variables (i.e., wildcards), indeed, is intractable for not minimal alphabets; for instance, for an alphabet of size $|\Sigma|$, it requires $O(|\Sigma|)$ runs of network inference per step of Gibbs sampling or iterated conditional modes [37]. Yet, they can handle the generation for a special case of templates (e.g., Column A and Column B), where the prefix of the template is fully known, and no observable character appears among the *wildcards*.

To generate over arbitrary templates, a possible trivial approach for autoregressive models would be to enumerate passwords according to the chosen cut-off probability and then filter the ones compatible with the chosen bias. However, this solution has two main drawbacks. First, this operation is costly, as well as storage-demanding. More significantly, such an approach can easily become intractable for small cut-off probability values, as the enumeration could require an exponential-scale cost due to the unpruned visit of the space. The second and more substantial limitation of this approach resides in the difficulty of generating relative low-probability guesses. In other words, **if the chosen bias results in candidate passwords having low probabilities (according to the estimated password distribution), those will be unlikely generated during the enumeration process, at least, for a reasonable cut-off probability.** In turn, this translates into the impossibility of enumeration-based approaches to generate the number of valid guesses required to a sound password guessing attack.

By contrast, conditional password generation can seamlessly be implemented within our representation-learning-based approach and its locality property. The password organization imposed by this locality principle maintains similar passwords bounded in a precise zone of the latent space. Localizing such zones using the template inversion technique and sampling from them allow us to enumerate biased passwords with minimal effort. We can conditionally produce suitable guesses for each meaningful bias, even if this yields low probability passwords. Algorithm 2 briefly formalizes this approach. Chosen a template t, we use the encoder network E to obtain the latent representation z^t of t. Then, we sample latent points from a distribution centered in z_t and with scale σ. During the process, we filter the guesses coherent with t (*if* statement at line 6). The effectiveness of this conditional guesses generation process will be demonstrated in the next section.

3.2.4 Evaluation

In this section, we evaluate our proposed CPG framework against state-of-the-art password guessers.

Biased test-sets creation

To create a suitable scenario to evaluate our conditional generation technique CPG, we cast a set of biased password test-sets. In our setup, a bias t_i is a password template; a string $t_i \in \{\Sigma \cup \{\bullet\}\}^*$ where Σ is the password alphabet (210 unicode characters in our case) and '$\bullet$' is the wildcard character. Every password template t_i is randomly extracted from a password sampled from a validation set X_v. We chose the *LinkedIn* [14] password leak as the validation-set. From this set, we keep passwords with length 16 or less, obtaining $6 \cdot 10^7$ unique passwords, which is ~ 5 times the RockYou train-set used to train our model.

More precisely, sampled a ground-truth password x from X_v, we derive t_i by substituting (with a certain probability p) each character in x with a wildcard (e.g., from x="jimmy1991" to t="•i•my•••1"). In our setup, we select $p = 0.5$. In this process, we select only those of the produced templates that contain at least 4 observable characters and at least 5 wildcards. The latter constraint aims at rendering not trivial a brute-force solution ($\sim 3 \cdot 10^{11}$).

After obtaining a large enough collection of valid templates, we create a set of biased password test-sets. This is achieved by collecting all the passwords matching the templates in X_v with an exhaustive search. More precisely, for each template, we collect all the instances x of X_v, such that x satisfies the template t_i; that is, the set $X_v^{t_i} = \{x | x \in X_v \wedge x \vdash t_i\}$. Based on the cardinality of the various $X_v^{t_i}$, we divide those into four classes:

1. T_{common}, if $|X_v^{t_i}| \in [1000, 15000]$
2. T_{uncommon}, if $|X_v^{t_i}| \in [50, 150]$
3. T_{rare}, if $|X_v^{t_i}| \in [10, 15]$
4. $T_{\text{super-rare}}$, if $|X_v^{t_i}| \in [1, 5]$

Eventually, each of the 4 classes of templates composes of 30 different template sets (i.e., $X_v^{t_i}$). Samples of these templates and respective matching passwords are reported in Table A.3 in Appendix A.4.

In the next section, we will use the created biased password sets to evaluate the proposed CPG framework with a set of probabilistic and non-probabilistic state-of-the-art password guessers. We evaluate the ability of each guesser to match the passwords contained in every biased set $X_v^{t_i}$.

Algorithm 2: Conditional Password Guessing (CPG)

Data: Template: t, Int: n, Real: σ

$X = \{\}$;
$z^t = E(t)$;
for $i := 1$ *to* n **do**
  $z_i \sim \mathcal{N}(z^t, \sigma\mathbf{I})$;
  $x_i = G(z_i)$;
  if $x_i \vdash t$ **then**
    $X = X \cup \{x_i\}$;
return X

Results

We perform our guessing attack using the CWAE. This model showed slightly better performance than the GAN approach in this guessing scenario.[6] We report results for the model trained on passwords with a maximum length of 16, as no consistently different results have been obtained with models trained on password lengths 10 and 22.

In our setup, we follow the CPG described in Section 3.2.3. More precisely, for each biased password set $X_v^{t_i}$, we invert the template t_i using the encoder network. Then we sample password around the obtained latent vector using standard-deviation $\sigma = 0.8$ (see Algorithm 2). We generate $n = 10^7$ valid passwords for each template, and then we compute the cardinality of the intersection of the generated guesses with $X_v^{t_i}$ to calculate the number of the guessed passwords.

We compare our CPG with five state-of-the-art guessers; namely, OMEN [53] and FLA [85] for the fully-probabilistic, PCFG [114] for token-based probabilistic, and HashCat [7] for non-probabilistic class. Additionally, we compare against a *min-auto* configuration [108].
As these guessers are not able to perform a natural form of conditional password generation, we exploit the naive approach discussed in Section 3.2.3; that is, we generate a large number of passwords in default mode and then filter the guesses coherently with the requested bias. In particular, we produced 10^{10} passwords for each approach. Details on the specific setup of these tools follow:

- **OMEN:** We trained the Markov chain using the same train-set used for our deep generative model (i.e., 80% RockYou). After that, we generated 10^{10} sorted guesses.
- **PCFG:** Like in the OMEN case, we used the train-set employed for the training of our deep generative model to infer the grammar.
- **HashCat:** We performed a mangling rules-based attack leveraging the train-set used for the training of our deep generative model as a dictionary (considering only unique passwords sorted by frequency), and we use PasswordsPro [11] as the set of rules. We chose the latter based on a suitable number of rules (i.e., 3120) that allowed us to produce a suitable number of guesses.
- **FLA:** We trained the largest model described in [85], i.e., an RNN composed of three LSTM layers of 1000 cells each and two fully connected layers. The training is carried out on the same train-set used for our model.
- **CMU-PGS:** In CMU Password Guessability Service (PGS) [23], the passwords are guessed according to the *min-auto* configuration [108], where guesses of multiple tools (i.e., FLA, Hashcat, John The Ripper, PCFG, Markov Model) are combined. We query the guess-numbers via the web interface and consider passwords requiring fewer than 10^{10} guesses. Recommended tools setup and *"1class1"* have been used.

When we test each of these guessers in the conditional generation, we transform each template in a regular expression (i.e., replacing the wildcards with the point operator) and extract all the guesses matching the template in the 10^{10} generated passwords. Then, we compute the cardinality of the intersection of the correct guesses with each $X_v^{t_i}$ to explicit the number of the guessed passwords.

[6] This is due to the higher quality of the encoder network included with the auto-encoder.

Templates class	**OMEN**	**HashCat** (PasswordPro)	**PCFG**	**FLA**	**CMU-PGS** (min-auto)	**Our CPG** (CWAE)
Common [1000-1500]	0.4383 (± 0.1835)	0.5563 (± 0.1274)	0.7546 (± 0.092)	0.7936 (± 0.0757)	**0.8617** (± 0.0517)	0.8136 (± 0.0641)
Uncommon [50-150]	0.2744 (± 0.1322)	0.3656 (± 0.1897)	0.5794 (± 0.1987)	0.6365 (± 0.1137)	0.7208 (± 0.1015)	**0.8606** (± 0.0686)
Rare [10-15]	0.1182 (± 0.1272)	0.2007 (± 0.1655)	0.4013 (± 0.2514)	0.3983 (± 0.1827)	0.5102 (± 0.2005)	**0.8482** (± 0.1444)
Super-Rare [1-5]	0.0555 (± 0.1448)	0.0900 (± 0.1700)	0.1527 (± 0.2298)	0.1500 (± 0.2961)	0.2277 (± 0.2763)	**0.7722** (± 0.2910)

Table 3.4: Average matched passwords (and relative standard deviation) over the biased passwords test-set divided into 4 classes.

The mean percentage of guessed passwords for each templates class is reported in Table 3.4. Coherently with the discussion done in Section 3.2.3, our CPG framework allows us to produce a large number of biased guesses, and it matched a large portion of passwords accordingly.

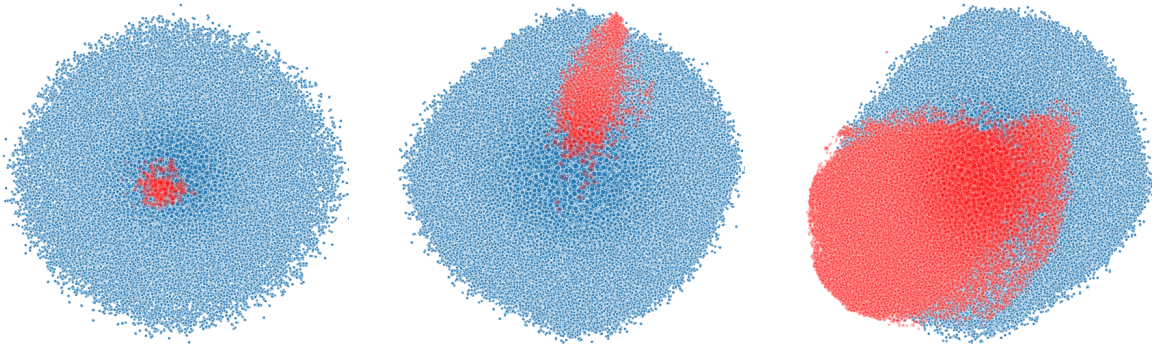
As anticipated, CPG maintains a high match ratio (i.e., $> 70\%$) for each template class independently of the corresponding passwords' low probabilities. In contrast, other guessers are not able to produce such a specific class of passwords. Therefore, they provide shallow coverage of the rare templates. This is also true for the *min-auto* attack, where heterogeneous guesses from multiple tools are combined. For instance, the *min-auto* approach would require three orders of magnitude more guesses to match the same number of passwords as ours in the edge-case of the *Super-Rare* templates. Interestingly, given the strong bias imposed during the generation, CPG matches most passwords of other *single* guessers also under the common templates case. The second best guesser turns out to be FLA that matches a comparable number of passwords as ours in the case of common templates and matches an acceptable number of passwords in the uncommon and rare classes (i.e., $\geq 40\%$). Note that we limited our CPG to generate 10^7 guesses per template; however, more biased passwords can be sampled in a linear cost.

3.3 Dynamic Password Guessing (DPG) and Passwords weak locality

In this section, we present our major contribution, i.e., Dynamic Password Guessing. In Section 3.3.1, we outline the concept of password weak locality. Section 3.3.2 introduces DPG from theoretical (Section 3.3.2) as well as practical (Section 3.3.2) viewpoints.

3.3.1 Password weak locality

The embedding properties of the latent representation map passwords with similar characteristics close to each other in the latent space. We called this property strong locality, and we exploited it to generate variants of a chosen pivot password or template (discussed in Section 3.2.1). In that case, the adjective "strong" highlights the strict semantic relation among the generated set of passwords. However, the same dynamics enable a broader form of semantic bounding among passwords. This latter property partially captures the general features of the entire password distribution. Such features could be very abstract properties of the distribution, such as the average passwords length and



(a) *myspace* (b) *hotmail* (c) *phpbb*

Figure 3.3: Password Weak Locality: 2D visualization of the latent points for three different passwords sets for a generator trained on the RockYou train-set. The red points represent the latent points corresponding to the passwords in the respective password set whereas the blue points loosely represent the dense part of the latent space. Please refer to the color version for better illustration.

character distribution ascribable to password policies. We refer to this observed property as **password weak locality** to contrast it with the strong locality.

As a representative example, Figure 3.3 depicts the 2D representation of passwords from *myspace* [15], *hotmail* [9], and *phpbb* [20] on the latent space learned by a generator.[7] We can observe that the passwords coming from the same dataset tend to be concentrated in the latent space and do not spread abruptly all over the spectrum. The dimensionality of the fraction of latent space covered by an entire password set (the red parts in Figure 3.3 (a), (b), and (c) clearly depends on the heterogeneity of its passwords. Passwords from smaller sets (e.g., *myspace*) are concentrated in restricted and dense zone of the latent space, whereas passwords from larger sets (e.g., as *phpbb*) tend to cover a more significant section while they are still tightly knitted.

In the following sections, we will present evidence of this locality property, and we will show how to exploit it to improve password guessing.

3.3.2 DPG for covariate shift reduction

First, we present the theoretical motivation behind DPG in Section 3.3.2 followed by its instantiation in Section 3.3.2.

Theoretical motivation

Probabilistic password guessing tools implicitly or explicitly attempt to capture the data distribution behind a set of observed passwords, i.e., the train-set. This modeled distribution is then used to generate new and coherent guesses during a password guessing attack. A train-set is usually composed of passwords that were previously leaked. By assumption, every password-set leak is characterized by a specific password distribution $p^*(\mathbf{x})$. When we train the probabilistic model, we implicitly assume $p^*(\mathbf{x})$ to be general enough

[7] It is important to emphasize that these graphical depictions are obtained by a dimension reduction algorithm. Hence, they do not depict latent space accurately. So, they merely serve as a representative illustration. We will verify our assumption empirically later in the Chapter.

to well-represent the entire class of password distributions. This generality is essentially due to the fact that the real-word password guessing attacks are indeed performed over sets of passwords that potentially come from completely different password distributions. As a matter of fact, we typically do not have any information about the attack-set distribution. This can indeed be completely different from the one used for model training. As a representative example, different password policies or users' predominant languages can cause the test-set's distribution to differ from the train-set's distribution drastically. This discrepancy in the distribution of the train-set and test-set is a well-known issue in the domain of machine learning, and it is referred to as *covariate shift* [102].

As stated above, typically, we do not know anything about the distribution of the attacked-set. However, once we crack the first password, we can start to observe and model the attacked distribution. Every new successful guess provides valuable information that we can leverage to improve the quality of the attack, i.e., to reduce the *covariate shift*. This iterative procedure recalls a Bayesian-like approach since there is continuous feedback between observations and the probability distribution.

For fully data-driven approaches, a naive solution to incorporate the acquired information from successful guesses is to fine-tune the model to change the learned password distribution. However, *prescribed probabilistic models* such as FLA directly estimate the password distribution using a parametric function:

$$p(\mathbf{x}) = p(\mathbf{x}; \theta), \tag{3.1}$$

where θ is the set of weights of a neural network. In this case, the only possibility of modifying the distribution $p(\mathbf{x})$ in a meaningful way is to act on θ by harnessing the learning process. However, this is not an easy/attractive solution mainly because the new guessed passwords are potentially inadequate representatives[8] and will not force the model to generalize over the new information. Additionally, the computational cost of fine-tuning the network is considerable, and the final results cannot be guaranteed due to the sensitivity of the learning process.

Similar to FLA, our generative model also exploits a neural network as an estimator. However, its modeled distribution is a joint probability distribution, shown in Eq. 3.2:

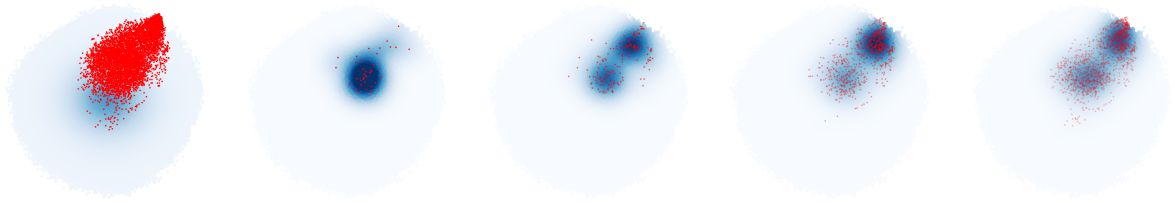
$$p(\mathbf{x}) = p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x} \mid z; \theta)p(\mathbf{z}), \tag{3.2}$$

where $p(\mathbf{z})$ is referred to as the latent distribution.

As introduced in Section 2.2.2, when $p(\mathbf{z}) = \dot{p}(\mathbf{z})$ (i.e., prior latent distribution), $p(\mathbf{x} \mid z;\ \theta)p(\mathbf{z})$ acts as a good approximation of the target data distribution (i.e., the distribution followed by the train-set). Nevertheless, $p(\mathbf{z})$ can be arbitrarily chosen and used to indirectly change the probability distribution modeled by the generator. The RHS of the Eq. 3.2 clearly shows that θ is not the only free parameter affecting the distribution of the final passwords. Indeed, $p(\mathbf{z})$ is completely independent of the generator, and so it can be modified arbitrarily without acting on the parameters of the neural network.

This possibility, along with the passwords locality of the latent space, allows us to correctly and efficiently generalize over the new guessed passwords, leading the pre-trained network to model a password distribution closer to the guessed ones. It is noteworthy that this capability of generalizing over the new points is achieved via the weak locality and not from the neural network itself. **The intuition here is that when we change $p(\mathbf{z})$ to assign more density to a specific guessed password x, we are also increasing**

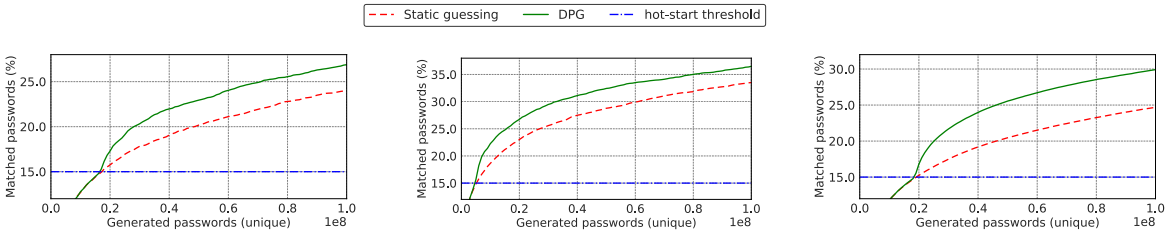
[8] A very few guessed passwords against a dataset of millions of unknown passwords.



(a) Target (b) 10^4 guesses (c) 10^5 guesses (d) 10^6 guesses (e) 10^7 guesses

Figure 3.4: 2D visualization of: (a) the entire *hotmail* dataset (red-part) mapped on the latent space learned from the RockYou train-set and (b-e) the latent space in four progressive attack steps for DPG on the *hotmail* test-set. The red markers portray the guessed passwords at each step (i.e., the Z_i), whereas the color intensity of the blue regions depicts the probability assigned from the used latent distribution (i.e., mixture of Gaussians) to the latent space.

the probability of its neighboring passwords that, due to the weak locality property, share similar characteristics. This, in turn, makes it possible to highlight the general features of the guessed passwords (e.g., structure, length, character set, etc.).

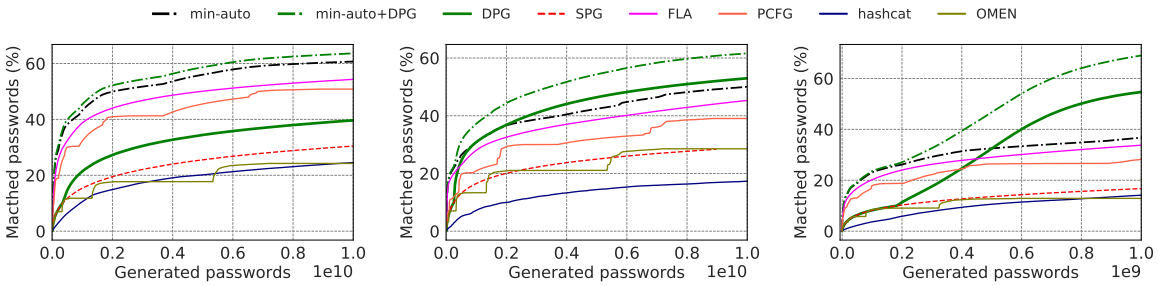


(a) *myspace* (b) *hotmail* (c) *phpbb*

Figure 3.5: The performance gain obtained by DPG (with $\alpha = 0.15$) with respect to static attack for three different test-sets

Thus, by controlling the latent distribution, we can increase the probabilities of the zones potentially covered by the passwords coming from the target distribution. We call this technique **Dynamic Password Guessing (DPG)**. In the case of homogeneous distribution (e.g., *myspace*), we can narrow down the solution space around the dense zones, and avoid exploring the entire latent-space. On the other hand, for passwords sets sampled from distributions far from the one modeled by the generator, we can focus on zones of the latent space, which, otherwise, would have been poorly explored. In both cases, we can reduce the *covariate shift* and improve the performance of the password guessing attack.

In a broad sense, DPG can potentially adapt to very peculiar password distributions; distributions induced from the contexts where no suitable train-sets can be collected. E.g., passwords created under an unmatched composition policy or rare/unobserved users' habits. As long as the generator has a non-zero probability of generating such rare passwords, the feedback given from the correct guesses can consistently be used to reweigh the latent distribution and mimic the unknown target password distribution. We will validate this claim in the next section.



(a) *LinkedIn* (b) *Youku* (c) *Zomato*

Figure 3.6: Performance of various password models on three password leaks. For DPG, we used $\sigma = 0.35$ and $\alpha = 10\%$.

Practical implementation

In this section, we cover DPG from a practical viewpoint. Algorithm 3 briefly describes DPG.

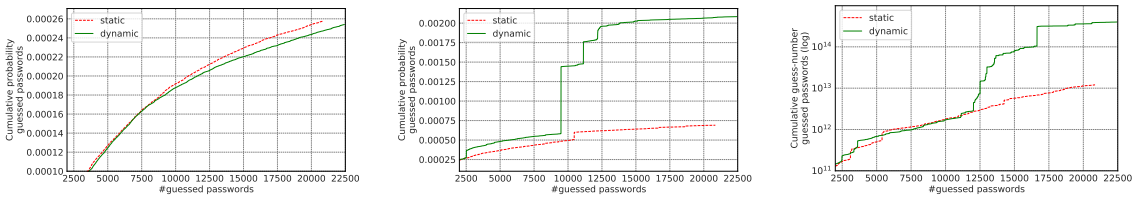
In Algorithm 3, $\mathcal{O}$ represents the target set of passwords, Z is the collection of all the passwords guessed by the generator, and α is defined as the **hot-start** parameter of the attack, an element that we describe later in this section. The variable p_{latent} in the pseudo-code, represents the latent distribution from which we sample latent points. The procedure *makeLatentDistribution* returns the latent distribution induced from the group of guessed passwords Z_i at step i. Leveraging the maximum-likelihood framework, we choose such a distribution to maximize the probability of the set of observed passwords $X_i = \{G(z) \mid z \in Z_i\}$. This is accomplished by considering a latent distribution $p(\mathbf{z} \mid Z_i)$ conditioned to the set of passwords guessed at each step i. The final password distribution represented by the generator during DPG is reported in Eq. 3.3.

$$p(\mathbf{x}) = p(\mathbf{x} \mid z; \theta)p(\mathbf{z} \mid Z_i). \tag{3.3}$$

As a natural extension of the proximity password generation harnessed in Section 3.2.2, we choose to represent $p(\mathbf{z} \mid Z_i)$ as a finite mixture of isotropic Gaussians. In particular, the mixture is composed of n Gaussians, where: (1) n is the number of the latent points in Z_i; and (2) for each $z_j \in Z_i$, a Gaussian is defined as $\mathcal{N}(z_j, \sigma\mathbf{I})$ with center as z_j and a fixed standard deviation σ.

Algorithm 3: Dynamic Password Guessing (DPG)

Data: Set: $\mathcal{O}$, Int: α
$i = 0$;
$p_{\text{latent}} = \dot{p}(\mathbf{x})$;
$Z = \{\}$;
forall $z \sim p_{latent}$ **do**
  $x = G(z)$;
  $x_i = G(z_i)$;
  if $x \in \mathcal{O}$ **then**
    $i++$;
    $Z_i = Z = Z \cup \{z\}$;
    if $i \geq \alpha$ **then**
      $p_{\text{latent}} = \text{makeLatentDistribution}(Z_i)$;



(a) P. train-set (b) P. attackes-set (c) G. train-set

Figure 3.7: Cumulative statistic for a guessing attack over *phpbb*. The figures report the password guessed in the first 10^9 guesses for both static and dynamic.

When the probability of a password, i.e., $x_j = G(z_j)$, is known, we weight the importance of the j^{th} distribution as $P(x_j)$; otherwise a uniform distribution among the Gaussians is assumed. In the reported experiments, we always used uniform weighting. Equation 3.4 defines the probability density function of the latent space.

$$p(\mathbf{z} \mid Z_i) = \sum_{j=0}^{n} P(G(z_j)) \cdot \mathcal{N}(\mathbf{z} \mid z_j, \sigma\mathbf{I}). \tag{3.4}$$

Every new guessed password x introduces a new Gaussian centered at z to the mixture. Consequently, every new guessed password contributes to changes in the latent distribution $p(\mathbf{z} \mid Z_i)$ by moving the density of the distribution in the zone of the latent space where it lies. Figure 3.4 visualizes this phenomenon.

In the context of DPG, the GAN generator performs slightly better than CWEA. For this reason, all the experiments reported in this section are obtained with our GAN generator trained on the RockYou train-set. Figure 3.5 depicts the performance comparison between a static attack (e.g., PassGAN) and DPG over the three passwords sets. Adaptively changing the latent distribution allows us to boost the number of guessed passwords per unit of time. Importantly, this improvement comes without any additional information or assumption over the attacked passwords set. In addition, the computational overhead due to the new sampling technique is negligible. The steep improvement in the performance obtained with DPG supports our view that reducing the *covariate shift* is a sound strategy.

The sudden growth in the guessed passwords in DPG (shown in Figure 3.5) is due to the hot-start or α parameter; in DPG, we use the prior latent distribution until a predetermined number (α) of passwords has been guessed. After that, we start to use the conditional latent distribution $p(\mathbf{z} \mid Z_i)$. The reason is that if DPG starts with the very first guessed password, then the latent distribution can be stuck in a small area of the latent space. However, launching DPG after guessing a sufficient number of passwords (i.e., after finding a set of uncorrelated latent points in the latent space) gives us the possibility to match a heterogeneous set of passwords, which correctly localize the dense zones of the latent space where the attacked passwords are likely to lie.

The final hyper-parameter of our attack is the standard deviation (σ) assigned to every Gaussian in the mixture. Under the Kernel Density Estimation (KDE) perspective, σ represents the *bandwidth* of our Gaussian kernels. In the guessing scenario, instead, this value defines how far we want to sample from the clusters of observed passwords. A larger value of σ allows us to be less biased and explore a wider zone around the guessed passwords; whereas a smaller value enables a more focused inspection of the latter. Appendix A.3 better explicates the effect of σ and α on DPG.

LinkedIn	Guess	o2linkedln	w2linkedln	ydlinkedln	linked6in6	j*linkedln	linked!in.	wslinked1n	linkedgcin	linked6in2	lslinkedln
	FLA **G**	$8.2 \cdot 10^{15}$	$6.3 \cdot 10^{15}$	$3.6 \cdot 10^{15}$	$3.6 \cdot 10^{15}$	$3.0 \cdot 10^{15}$	$2.8 \cdot 10^{15}$	$2.6 \cdot 10^{15}$	$2.5 \cdot 10^{15}$	$1.4 \cdot 10^{15}$	$1.4 \cdot 10^{15}$
	DPG **G**	$3.4 \cdot 10^{9}$	$3.1 \cdot 10^{9}$	$3.6 \cdot 10^{9}$	$4.3 \cdot 10^{9}$	$4.3 \cdot 10^{9}$	$4.8 \cdot 10^{9}$	$4.4 \cdot 10^{9}$	$2.1 \cdot 10^{9}$	$5.6 \cdot 10^{9}$	$4.5 \cdot 10^{9}$
Youku	Guess	guoxuange2	xuhaidong7	caoxia521.	woailc521.	woyijiu521	woaicyhx0	xuhaidong1	woaifiy520	yishwng521	woshiqujie
	FLA **G**	$2.5 \cdot 10^{15}$	$1.7 \cdot 10^{15}$	$1.3 \cdot 10^{15}$	$9.6 \cdot 10^{14}$	$7.3 \cdot 10^{14}$	$6.5 \cdot 10^{14}$	$6.4 \cdot 10^{14}$	$6.4 \cdot 10^{14}$	$5.3 \cdot 10^{14}$	$5.1 \cdot 10^{14}$
	DPG **G**	$3.2 \cdot 10^{9}$	$3.9 \cdot 10^{9}$	$3.5 \cdot 10^{9}$	$3.7 \cdot 10^{9}$	$3.5 \cdot 10^{9}$	$3.3 \cdot 10^{9}$	$3.9 \cdot 10^{9}$	$3.8 \cdot 10^{9}$	$3.7 \cdot 10^{9}$	$3.0 \cdot 10^{9}$
Zomato	Guess	z0mato2016	z0mato2015	zomato9a00	2defd0	zomat_997	3aee0f	zomato_496	zomato_443	zomato.921	zomato_591
	FLA **G**	$1.9 \cdot 10^{14}$	$1.5 \cdot 10^{14}$	$1.2 \cdot 10^{14}$	$7.3 \cdot 10^{13}$	$4.0 \cdot 10^{13}$	$3.8 \cdot 10^{13}$	$3.5 \cdot 10^{13}$	$3.4 \cdot 10^{13}$	$3.2 \cdot 10^{13}$	$3.1 \cdot 10^{13}$
	DPG **G**	$4.5 \cdot 10^{8}$	$7.7 \cdot 10^{8}$	$7.8 \cdot 10^{8}$	$5.1 \cdot 10^{8}$	$1.0 \cdot 10^{9}$	$8.1 \cdot 10^{8}$	$8.0 \cdot 10^{8}$	$1.1 \cdot 10^{9}$	$1.1 \cdot 10^{9}$	$1.1 \cdot 10^{9}$
phpbb	Guess	phpbb3.14	phpbb0472	phpbb4s2	phpbb7825	phpbbid12	phpbb8424	phpbb3546	phpbb4291	phpbb8686	phpbb9801
	FLA **G**	$2.1 \cdot 10^{14}$	$2.1 \cdot 10^{13}$	$2.0 \cdot 10^{13}$	$1.3 \cdot 10^{13}$	$1.0 \cdot 10^{13}$	$9.9 \cdot 10^{12}$	$8.0 \cdot 10^{12}$	$7.2 \cdot 10^{12}$	$5.5 \cdot 10^{12}$	$5.4 \cdot 10^{12}$
	DPG **G**	$2.4 \cdot 10^{8}$	$6.5 \cdot 10^{8}$	$4.8 \cdot 10^{8}$	$4.2 \cdot 10^{8}$	$1.2 \cdot 10^{8}$	$1.1 \cdot 10^{8}$	$1.3 \cdot 10^{8}$	$1.0 \cdot 10^{8}$	$1.4 \cdot 10^{8}$	$2.0 \cdot 10^{8}$

Table 3.5: Example of peculiar passwords guessed via DPG for four password leaks. The required numbers of guesses (i.e., **G**) are reported for both FLA and our DPG. These passwords have been obtained by ordering all the guessed passwords of the DPG attacks in decreasing order based on the guess-number assigned from FLA . The table reports the first 10 entries of the list for each leak.

In Figure 3.6, we report a direct comparison of the proposed DPG against state-of-the-art password models for three password leaks. For the comparison, we used the same tools and configurations described in Section 3.2.4.[9] In the figure, DPG refers to the dynamic guessing attack, whereas SPG to the static one. *min-auto* is obtained by combining the guesses of FLA, Hashcat, OMEN, and PCFG. *min-auto+DPG* is then obtained by adding DPG to the *min-auto* ensemble. Figure 3.6 (a) reports the results for the *LinkedIn* leak. Here, the dynamic adaptation allows us to guess up to 10% more passwords then the static approach. However, it cannot directly match the performance of FLA and PCFG in this general case. Nevertheless, our models behave better than mangling rules and the Markov model. Given the different nature of the dynamic guessing strategy, combining DPG with *min-auto* permits us to guess more passwords. We will better motivate this phenomenon in the next section.

Consistently better results are observed as soon as we consider leaks that exhibit peculiar biases in their password distributions. Figure 3.6 (b) reports the results for the leak *Youku* [13, 5] - a Chinese video hosting service. In this case, the inherent distribution shifts induced by a different class of users causes a substantial covariate shift phenomenon. Here, the dynamic adaptation allows us to guess more passwords than the other tools; DPG improves guess after guess, each time evolving and eventually surpassing the *min-auto* configuration obtained by combining all other models.

Even more interesting results can be observed when we consider leaks that introduce heavier biases. Figure 3.6 (c) reports the results for the *Zomato* [26, 25] leak. This leak is an extreme case since $\sim 40\%$ of its content includes random tokens of six alphanumeric characters. That creates a sharply segmented bimodal distribution that can be detected and efficiently captured by DPG. In this instance, the dynamic adaptation of the latent space allows us to guess up to ~ 5 times more passwords than the static attack (i.e., SPG), allowing our model to match more than 50% of the set in less than 10^9 iterations. On the other hand, static approaches, including *min-auto*, cannot match the performance of DPG in this extreme case. Of note, adding DPG to the ensemble of *min-auto* (i.e., *min-auto+DPG*) allows us to guess $\sim 70\%$ of the set.

The last two examples highlight the ability of DPG to adapt to the target password distribution. However, the result of the *LinkedIn* leak tells us that the dynamic attack

[9]For the *min-auto*, we do not use CMU-PGS [23] directly given their limits on the number of queries allowed and the cardinality of the tested sets.

cannot directly match the performance of state-of-the-art solutions in case there is no evident covariate shift. In the next section, we will show that the DPG algorithm is indeed useful also in such cases, as it soundly permits to guess peculiar passwords of the attacked distribution that would be otherwise ignored.

The impact of the dynamic adaptation

In this section, we clarify the effect of the dynamic latent adaptation over the password distribution originally modeled from the deep generative model. To this end, we compare the probability of the guessed passwords according to different password distributions, namely, (1) the distribution of the train-set and (2) the distribution of the attacked-set of passwords. To soundly represent and generalize such probability distributions, we rely on FLA [85] as an explicit password mass estimator. We train two instances of FLA on the two passwords sets and use the trained models to infer probabilities over the password guessed during the dynamic and static attacks.

Figure 3.7 summarizes our measurements for the *phpbb* password leak (i.e., the attacked distribution). Here, the cumulative probability of the guessed password is reported for both dynamic and static attacks. In particular, Figure 3.7 (a) describes the probabilities assigned from the probability distribution of the train-set (i.e., the FLA instance trained on *RockYou*), whereas Figure 3.7 (b) reports the same data points, but computed according to the probability distribution of the attacked-set (i.e., the FLA instance trained on *phpbb*).

When we perform DPG, we expect the password distribution represented from the deep generative model to gradually diverge from the one learned at training time. Figure 3.7 (a) graphically describes this phenomenon; here, we note how the latent adaptation is causing the model to guess passwords that have a lower probability according to the train-set distribution. More interestingly, whereas the discrepancy between the modeled and the train distribution grows, the discrepancy sharply reduces for the attacked distribution. Figure 3.7 (b) explicates the convergence process towards the latter. Furthermore, this figure gives us a piece of more valuable information. It shows that the DPG guesses passwords that have high-probability according to the attacked distribution, i.e., passwords associated with a higher number of users in the attacked service. Sudden jumps in the latter cumulative probability curve, indeed, can be attributed to the event of guessing such high-probability passwords. To note, once we guess a first high-probability password, we start sampling new guesses around it, guessing more high-probability passwords consequently and making those jumps even steeper.

Relying on the same example, more practical results can be appreciated when we consider the adversarial interpretation. Figure 3.7 (c) reports the cumulative guess-number graph for the static and dynamic attacks measured using the FLA instance trained on *RockYou* (i.e., the train-set of our model). The estimated cumulative guess-number of the dynamic attack is two magnitudes larger than that of the static attack. Considering FLA's accuracy [56], this result confirms that DPG can induce the generation of passwords that have low belief according to the train-set distribution. Moreover, this example shows how DPG can induce the earlier generation of passwords that would require multiple magnitude more guesses to be produced for equivalent state-of-the-art password guessers, such as FLA. In the reported example, we generate 10^9 guesses, matching several passwords that would require up to 10^{14} iterations from FLA (and others; see Appendix A.4). Table 3.5 reports some of those.

We replicated the same analysis on different password leaks, observing the same general behavior. We reported high-guess-number passwords for those other sets as additional examples in Table 3.5. The listed guesses in the table give a clear intuition over the nature of such peculiar passwords. These are induced from unique biases of the attacked distribution. More evident examples are the passwords based on the name of web services that dominate the table. These are indeed the prime examples of peculiar passwords, as they univocally bound to the specific password distribution. More heterogeneous guesses can be observed in the row dedicated to the *Youku* leak. Here, DPG captured passwords composed of peculiar dictionary entries that are not well represented in the train-set of the model (i.e., *RockYou*).
Additionally, the guess-numbers reported in Table 3.5 indicate that these are passwords that are considered secure by state-of-the-art tools, but that can be easily guessed through DPG. Indeed, our experiments show that **DPG allows us to guess passwords that are unique to the attacked password set. Such passwords, given their arbitrary distance from the general password distribution, can be soundly guessed only by leveraging additional sources of information over the attacked password space. DPG distills this necessary knowledge directly through an unsupervised interaction with the attacked set. This allows the guessing attack to automatically focus on unique modalities of the target password distribution that would otherwise be under-represented or ignored.**

3.4 Conclusion

In conclusion, in this chapter, we demonstrated how deep learning can induce a complete paradigm shift in the task of password guessing. We demonstrated that locality principles imposed by the latent representation of deep generative models open new practical and theoretical possibilities in the field. Based on these properties, we discussed two new password guessing frameworks, i.e., CPG and DPG. The CPG framework enables the conditional generation of arbitrarily biased passwords. We empirically demonstrated its inherent advantages with respect to well-established state-of-the-art approaches. In addition, the DPG framework demonstrates that the knowledge from freshly guessed passwords can be successfully generalized and used to mimic the target password distribution. More importantly, this guessing technique allows the generation of passwords that are peculiar for the attacked password distribution, and that would require an impractical effort to be guessed by other guessers.

Chapter 4

Reducing Bias in Modeling Real-world Guessing Attacks

More than three decades of active research provided us with powerful password models [89, 114, 85, 94]. However, very little progress has been made to systematically model real-world attackers [108, 76]. Indeed, professional password crackers rarely harness fully-automated approaches developed in academia. They rely on more pragmatic guessing techniques that present stronger inductive biases. In offline attacks, professionals use high-throughput and flexible techniques such as **dictionary attacks with mangling rules** [2]. Moreover, they rely on highly tuned setups that result from profound expertise that is refined over years of practical experience [108, 76].

Reproducing or modeling these proprietary attack strategies is very difficult, and the end results rarely mimic actual real-world threats [108]. This failure often results in an overestimation of password security that sways studies' conclusions and further jeopardize password-based systems.

In this chapter, we describe a new generation of dictionary attacks that more closely resembles real-world attackers' abilities and guessing strategies. In the process, we devise two complementary techniques that aim to systematically mimic different attackers' behaviors:

(1) By rethinking the underlying framework, we devise the **Adaptive Mangling Rules attack**. This artificially simulates the optimal configurations harnessed by expert adversaries by explicitly handling **the conditional nature of mangling rules**. Here, during the attack, each word from the dictionary is associated with a dedicated and possible unique rules-set that is decided at runtime via a **deep neural network**. Using this technique, we confirmed that standard attacks, based on *off-the-shelf* dictionaries and rules-sets, are sub-optimal and can be easily compressed up to an order of magnitude in the number of guesses.Furthermore, we are the first to explicitly model the strong relationship that bounds mangling rules and dictionary words, demonstrating its connection with optimal configurations.

(2) Our second contribution introduces **dynamic guessing strategies** within dictionary attacks [94]. Real-world adversaries perform their guessing attacks incorporating prior knowledge on the targets and dynamically adjusting their guesses during the attack. In doing so, professionals seek to optimize their configurations and maximize the number of compromised passwords. Unfortunately, automatic guessing techniques fail to model this adversarial behavior. Instead, we demonstrate that dynamic guessing strategies can be enabled in dictionary attacks and substantially improve the guessing attack's

effectiveness while requiring no prior optimization. More prominently, our technique makes dictionary attacks consistently more resilient to misconfigurations by promoting the completeness of the wordlist at runtime.

Finally, we combine these methodologies and introduce the Adaptive Dynamic Mangling rules attack (*AdaMs*). We show that it automatically causes the guessing strategy to progress toward an optimal one, regardless of the initial attack setup. The *AdaMs* attack consistently reduces the overestimation induced by inexpert configurations in dictionary attacks, enabling more robust and sound password strength estimates.

4.1 Related work and preliminaries

In this Section, we start by covering previous relevant works in the dictionary attacks domain. Then, we define the threat model in Section 4.1.2.

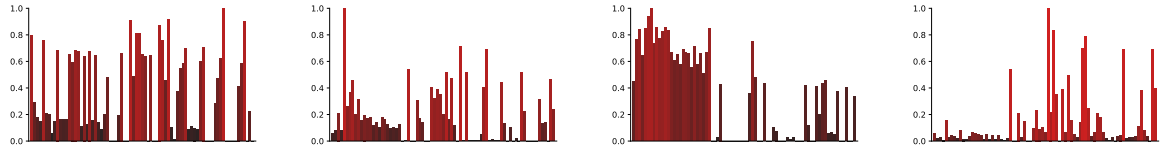
4.1.1 Related Works

Although dictionary attacks are ubiquitous in password security research [45, 54, 55, 85, 73], little effort has been spent studying them. This Section covers the most relevant contributions.

Ur et al. [108] firstly made explicit the large performance gap between optimized and stock configurations for mangling rules attacks. In their work, Ur et al. recruited professional figures in password recovery and compared their performance against off-the-shelf parametric/nonparametric approaches in different guessing scenarios. Here, professional attackers have been shown capable of vastly outperform any password model. This thanks to custom dictionaries, proprietary mangling rules, and the ability to create tailored rules for the attacked set of passwords (referred to as *freestyle* rules). Finally, the authors show that the performance gap between professional and non-professional attacks can be reduced by combining guesses of multiple password models.

More recently, Liu et al. [76] produced a set of tools that can be used to optimize the configuration of dictionaries attacks. These solutions extend previous approaches [4, 10], making them faster. Their core contribution is an algorithm capable of inverting almost all mangling rules; that is, given a rule r and password to evaluate p, the inversion-rule function produces as output a *regex* that matches all the preimages of $r(p)$ i.e., all the dictionary entries that transformed by r would produce p. At the cost of an initial pre-computation phase, following this approach, it is possible to count dictionary-words/mangling-rules hits on an attacked set without enumerating all the possible guesses. Liu et al. used the method to optimize the ordering of mangling rules in a rules-set by sorting them in decreasing hits-count order.[1] In doing so, the authors observed that default rules-sets follow an optimal ordering only rarely.
Basing on the same general approach, they speedup the automatic generation of mangling rules [4] and augment dictionaries by adding missing words in consideration of known attacked sets [10]. Similarly, they derive an approximate guess-number calculator for rule-major order attacks.

[1] Primarily, for rule-major order setups (e.g., JtR).



(a) Only digits. (b) Only capital letters. (c) Strings of length 5. (d) Strings of length 10.

Figure 4.1: Distribution of hits per rule for 4 different input dictionaries for the same attacked-set i.e., *animoto*. Within a plot, each bar depicts the normalized number of hits for one of the 77 mangling rules in *best64*. We performed the attack with *Hashcat*.

4.1.2 Threat Model

In our study, we primarily model the case of trawling, offline attacks. Here, an adversary aims at recovering a set of passwords $\mathbf{X}$ (also referred to as *attacked-set*) coming from an arbitrary password distribution $P(\mathbf{x})$ by performing a guessing attack. To better describe both the current trend in password storing techniques [98, 66, 97] and real-world attackers' goals [32], we assume a rational attacker who is bound to produce a limited number of guesses. More precisely, this attacker aims at maximizing the number of guessed passwords in $\mathbf{X}$ given a predefined *budget* i.e., a maximal number of guesses the attacker is willing to perform on $\mathbf{X}$. Hereafter, we model this strategy under the form of *β-success-rate* [38, 33]:

$$s_\beta(X) = \sum_{i=1}^{\beta} P(x_i). \tag{4.1}$$

Experimental setup In our construction, we do not impose any limitation on the nature of $P(\mathbf{x})$ nor the attacker's *a priori* knowledge. However, in our experiments, we consider a weak attacker who does not retain any initial knowledge of the target distribution i.e., who cannot provide an optimal attack configuration for $\mathbf{X}$ before the attack. This last assumption makes a better description of the use-case of automatic guessing approaches currently used in password security studies.

In the attacks reported in the Chapter, we always sort the words in the dictionary according to their frequency. The password leaks that we use through the Chapter are listed in Appendix B.1.

4.2 The Adaptive Mangling Rules attack

In this Section, we introduce the first core block of our password model: the Adaptive Mangling Rules. We start in Section 4.2.1, where we make explicit the conditional nature of mangling rules while discussing its connection with optimal attack configurations. In Section 4.2.2, we model the functional relationship connecting mangling rules and dictionary words via a deep neural network. Finally, leveraging the introduced tools, we establish the Adaptive Mangling Rules attack in Section 4.2.3.

Motivation: Dictionary attacks are highly sensitive to their configuration; while parametric approaches tend to be more robust to train-sets and hyper-parameters choices, the performance of dictionary attacks crucially depends on the selected dictionary and

rules-set [108, 76]. As evidenced by Ur et al. [108], real-world attackers rely on extremely optimized configurations. Here, dictionaries and mangling rules are jointly created over time through practical experience [2], harnessing a domain knowledge and expertise that is mostly unknown to the academic community [76]. Very often, password security studies rely on publicly available dictionaries and rules-sets that are not as effective as advanced configurations adopted by professionals. Unavoidably, this leads to a constant overestimation of password strength that skews the conclusions of studies and reactive analysis.

Hereafter, we show that the domain-knowledge of professional attackers can be suitably approximated with a **Deep Neural Network**. Given that, we devise a new dictionary attack that autonomously promotes functional interaction between the dictionary and the rules-set, implicitly simulating the precision of real-world attackers' configurations.
We start by presenting the intuition behind our technique. Formalization and methodology are reported later.

4.2.1 The conditional nature of mangling rules

As introduced in Section 2.1.2, dictionary attacks describe password distributions by factorizing guesses into two main components—a dictionary word w and a transformation rule r. Here, the word w acts as a **semantic base**, whereas r is a **syntactic transformation** that aims at providing a suitable guess through the manipulation of w. Generally speaking, such factorized representation can be thought of as an approximation of the typical users' composition behavior: starting from a plain word or phrase, users manipulate it by performing operations such as *leeting*, appending characters or concatenation.

At configuration time, such transformations are abstracted and collected in arbitrary large rules-sets under the form of mangling rules. Then, during the attack, guesses are reproduced by exhaustively applying the collected rules on all the words in the dictionary. In this generation process, rules are applied unconditionally on all the words, assuming that the abstracted syntactic transformations equally interact with all the elements in the dictionary. However, arguably, users do not follow the same simplistic model in their password composition process. Users first select words and then mangling transformations conditioned by those words. That is, mangling transformations are subjective and depend on the base words on which those are applied. For instance, users may prefer to append digits at the end of a name (e.g., "*jimmy*" to "*jimmy91*"), repeat short words rather than long ones (e.g., "*why*" to "*whywhywhy*") or capitalize certain strings over others (e.g., "*cookie*" to "*COOKIE*").

Pragmatically, we can think of each mangling rule as a function that is valid on an arbitrary small subset of the dictionary space, strictly defined by the users' composition habits. Thus, applying a mangling rule on words outside this domain unavoidably brings it to produce guesses that have only a negligible probability of inducing hits during the guessing attack (i.e., that do not replicate users' behavior). This concept is captured in Figure 5.3, where four panels depict the *hits* distribution of the rules-set "*best64*" for four different dictionaries. Each dictionary represents a specific subset of the dictionary space that has been obtained by filtering out suitable strings from the *RockYou* leak; namely, these are passwords composed of: digits (Figure 4.1(a)), capital letters (Figure 4.1(b)), passwords of length 5 (Figure 4.1(c)), and passwords of length 10 (Figure 4.1(d)). The four histograms show how mangling rules selectively and heterogeneously interact with the underlying dictionaries. Rules that produce many hits for a specific dictionary inevitably

perform very poorly with the others.

Eventually, the conditional nature of mangling rules has a critical impact in defining the effectiveness of a dictionary attack. To reach optimal performance, an attacker has to resort on a combination of dictionary and rules-set that mutually maximizes the conditional activation of mangling rules. In this direction, we can see highly optimized configurations used by experts as pairs of dictionaries and rules-sets that organically support each other in the guesses generation process.[2] On the other hand, configurations based on arbitrary chosen rule-sets and dictionaries may not be fully compatible and, as we show later in the Chapter, generate a large number of low-quality guesses. Unavoidably, this phenomenon makes adversary models based on mangling rules inaccurate, and induce an overestimation of password strength [108].

Next, we show how modeling the conditional nature of mangling rules allows us to cast dictionary attacks that are inherently more resilient to poor configurations.

4.2.2 A Model of Rule/Word Compatibility

We introduce the notion of **compatibility** that refers to the functional relation among dictionary words and mangling rules discussed in the previous Section. The compatibility can be thought of as a continuous value defined between a mangling rule r and a dictionary-word w that, intuitively, measures the *utility* of applying the rule r on w. More formally, we model *compatibility* as a function:

$$\pi : R \times \mathbb{W} \to [0, 1],$$

where R and $\mathbb{W}$ are the rule-space (i.e., the set of all the suitable transformations $r : \mathbb{W} \to \mathbb{W}$) and the dictionary-space (i.e., the set of all possible dictionary words), respectively. Values of $\pi(w, r)$ close to 1 indicate that the transformation induced by r is well-defined on w and would lead to a valuable guess. Values close to 0, instead, indicate that users would not apply r over w, i.e., guesses will likely fall outside the dense zone of the password distribution.

This formalization of compatibility function also leads to a straightforward probabilistic interpretation that better supports the learning process through a neural network. Indeed, we can think of π as a probability function over the event:

$$r(w) \in \mathbf{X},$$

where $\mathbf{X}$ is an abstraction of the attacked set of passwords. More precisely, we have that:

$$\forall_{w \in \mathbb{W},\ r \in R}\big(\pi(r, w) = P(r(w) \in \mathbf{X})\big).$$

In other words, $P(r(w) \in \mathbf{X})$ is the probability of guessing an element of $\mathbf{X}$ by trying the guess $g = r(w)$ produced by the application of r over w.
Furthermore, such a probability can be seen as an unnormalized version of the password distribution, creating a direct link to probabilistic password models [85, 89]. Indeed, the

[2] This has also been indirectly observed by Ur et al. in their ablation study on pro's guessing strategy, where the greatest improvement was achieved with a proprietary dictionary in tandem with a proprietary rules-set.

compatibility function can be interpreted as an unnormalized version of the password generating function $P(\mathbf{x})$. That is:

$$\forall_{w \in \mathbb{W},\ r \in R} \langle \frac{\pi(r, w)}{Z} = P(r(w)) \rangle$$

for an intractable partition function Z. This follows from the observation that:

$$\begin{gathered} \forall g_i, g_j \in \mathbf{X} : P(g_i) \geq P(g_j) \Leftrightarrow P(r_i(x_i) \in \mathbf{X}) \geq P(r_j(x_j) \in \mathbf{X}) \\ \text{with :}\ \ g_i = r_i(x_i) \text{ and } g_j = r_j(x_j), \end{gathered} \tag{4.2}$$

where $\mathbf{X}$ is the key-space. However, here, the password distribution is defined over the factorized domain $R \times \mathbb{W}$ rather than directly over the key-space.
This factorized form offers us practical advantages over the classic formulation. More in detail, by choosing and fixing a specific rule-space R (i.e., a rules-set), we can reshape the compatibility function as:

$$\pi_R : \mathbb{W} \to [0, 1]^{|R|}. \tag{4.3}$$

This version of the compatibility function takes as input a dictionary-word and outputs a compatibility value for each rule in the chosen rule-set with **a single inference**. This form is concretely more computational convenient and will be used to model the neural approximation of the compatibility function.

Next, we show how the compatibility function can be inferred from raw data using a deep neural network.

Learning the compatibility function

As stated before, the probabilistic interpretation of the compatibility function makes it possible to learn π using a neural network. Indeed, the probability $P(r(w) \in \mathbf{X})$, in any form, can be described through a binary classification: for each pair word/rule $(w,\ r)$, we have to predict one of two possible outcomes: $g \in \mathbf{X}$ or $g \notin \mathbf{X}$, where $g = r(w)$. In solving this classification task, we can train a neural network in a logistic regression and obtain a good approximation of the probability $P(r(w) \in \mathbf{X})$.

In the same way, the reshaped formulation of π (i.e., Eq. 4.3) describes a **multi-label classification**. In a multi-label classification, each input participates simultaneously to multiple binary classifications i.e., an input is associated with multiple classes at the same time. More formally, having a fixed number of possible classes n, each data point is mapped to a binary vector in $\{0, 1\}^n$. In our case, $n = |R|$ and each bit in the binary vector corresponds to the outcome of the event $r_j(w) \in X$ for a rule $r_j \in R$.

To train a model, then, we have to resort to a supervised learning approach. We have to create a suitable training-set composed of pairs *(input,label)* that the neural network can model during the training. Under our construction, we can easily produce such suitable labels by performing a mangling rules attack. In particular, fixed a rules-set R, we collect pairs (w_i, y_i), where w_i is the input to our model (i.e., a dictionary-word) and y_i is the label vector associated with w_i. As explicated before, the label y_i asserts the membership of the list of guesses $[r_1(w_i), r_2(w_i), \ldots, r_{|R|}(w_i)]$ over a hypothetical target set of passwords $\mathbf{X}$ i.e., :

$$y_i = [r_1(w_i) \in \mathbf{X},\ r_2(w_i) \in \mathbf{X},\ \ldots,\ r_{|R|}(w_i) \in \mathbf{X}] \tag{4.4}$$

To collect labels, then, we have to concertize $\mathbf{X}$ by choosing a representative set of passwords. Intuitively, such a set should be sufficiently large and diverse since it describes

Name	Cardinality	Brief Description
PasswordPro	3120	Manually produced.
generated	14728	Automatically generated.
generated2	65117	Automatically generated.

Table 4.1: Used *Hashcat*'s mangling rules sets.

the entire key-space. Hereafter, we refer to this set as X_A. This is the set of passwords we attack during the process of collecting labels.
In the same way, we have to choose another set of strings W that represents and generalizes the dictionary-space. This is used as input to the neural network during the training process, and as the input dictionary during the simulated guessing attack. Details on the adopted set are given at the end of the section.

Finally, given X_A and W, and chosen a rules-space R, we construct the set of labels by simulating a guessing attack; that is, for each entry w_i in the dictionary W, we collect the label vector y_i (E.q. 4.4). In doing so, we used a modified version of *Hashcat* described in Appendix B.6. Alternatively, the technique proposed in [76] can be used to speedup the labels collection.
Unlike the actual guessing attack, in the process, we do not remove passwords from X_A when those are guessed correctly; that is, the same password can be guessed multiple times by different combinations of rules and words. This is necessary to correctly model the functional compatibility. In the same way, we do not consider the identity mangling rule (i.e., *':'*) in the construction of the training set. When it occurs, we remove it from the rules set. To the same end, we do not consider hits caused by *conditional identity transformations* i.e., $r(w) = w$.

Training set configuration The creation of a training set entails the proper selection of the sets X_A and W as well as the rules-set R. Arguably, the most critical choice is the set X_A, as this is the ground-truth on which we base the approximation of the compatibility function. In our study, we select X_A to be the password leak discovered by *4iQ* in the *Dark Web* [1]. We completely anonymized all entries by removing users' information, and obtained a set of $\sim 4 \cdot 10^8$ of *unique* passwords. We use this set as X_A within our models. Similarly, we want W to be a good description of the dictionary-space. However, in this case, we exploit the generalization capability of the neural network that can automatically infer a general description of the input space from a relatively small training set. In our experiments, we use the *LinkedIn* leak as W.
Finally, we train three neural networks that learn the compatibility function for three different rules-sets; namely *PasswordPro*, *generated* and *generated2*. Those sets are provided with the *Hashcat* software and widely studied in previous works [76, 94, 85]. Table 4.1 lists them along with some additional information.

Eventually, the labels we collect in the guessing process are extremely sparse. In our experiments, more than 95% of the guesses are a miss, causing our training-set to be extremely unbalanced towards the negative class.

Model definition and training We construct our model over a residual structure [61] primarily composed of mono-dimensional convolution layers. Here, input strings are first embedded at character-level via a linear transformation; then, a series of residual

blocks are sequentially applied to extract a global representation for dictionary words. Finally, such representations are mapped into the label-space by means of a single, linear layer that performs the classification task. This architecture is trained in a multi-label classification; each output of the final dense layer is squashed in the interval $[0, 1]$ via the logit function, and binary cross entropy is applied to each probability separately. The network's loss is then obtained by summing up all the cross-entropies of the $|R|$ classes.

As mentioned in the previous Section, our training-set is extremely unbalanced toward the negative class; that is, the vast majority of the ground-truth labels assigned to a training instance are negative. Additionally, a similar disproportion appears in the distribution per rule. Typically, we have many rules that count only a few positive examples, whereas others have orders of magnitude more hits. In our framework, we alleviate the negative effects of those disproportions by inductive bias. In particular, we achieve it by considering a **focal regulation** in our loss function [75].

Originally developed for object detection tasks in which there is a strong imbalance between foreground and background classes, we adopt focal regulation to account for sparse and underrepresented labels when learning the compatibility function. This *focal loss* is mainly characterized by a modulating factor γ that dynamically reduces the importance of well-classified instances in the computation of the loss function, allowing the model to focus on hard examples (e.g., underrepresented rules). More formally, the form of regularized binary cross entropy that we adopt is defined as:

$$\mathrm{FL}(p_j, y_j) = \begin{cases} -(1-\alpha)(1-p_j)^{\gamma}\log(p_j) & \text{if } y_j = 1 \\ \alpha p_j^{\gamma}\log(1-p_j) & \text{if } y_j = 0 \end{cases},$$

where p_j is the probability assigned by the model to the j'th class, and y_j is the ground-truth label (i.e., 1/hit and 0/miss). The parameter α in the equation allows us to declare an *a priori* importance factor to the negative class. We use that to down-weighting the correct predictions of the negative class in the loss function that would be dominant otherwise. In our setup, we dynamically select α based on the distribution of the hits observed in the training set. In particular, we choose $\alpha = \frac{\bar{p}}{(1-\bar{p})}$, where $\bar{p}$ is the ratio of positive labels (i.e., hits/guesses) in the dataset. Differently, we fix $\gamma = 2$ as we found this value to be optimal in our experiments.
Summing up, our loss function is defined as:

$$\mathcal{L}_f = \mathbb{E}_{x,y} \sum_{j=1}^{|R|} \mathrm{FL}(sigmoid(f(x)_j), y_j)$$

where f are the *logits* of the neural network. We train the model using *Adam* stochastic gradient descent [68] until an early-stopping-criteria based on the AUC of a validation set is reached.

Maintaining the same general architecture, we train different networks with different sizes. In our experiments, we noticed that large networks provide a better approximation of the compatibility function, although small networks can be used to reduce the computational cost with a limited loss in utility. We report the results only for our biggest networks.

We implemented our framework on *TensorFlow*; the models have been trained on a *NVIDIA DGX-2* machine. A complete description of the architectures employed is given in Appendix B.2.

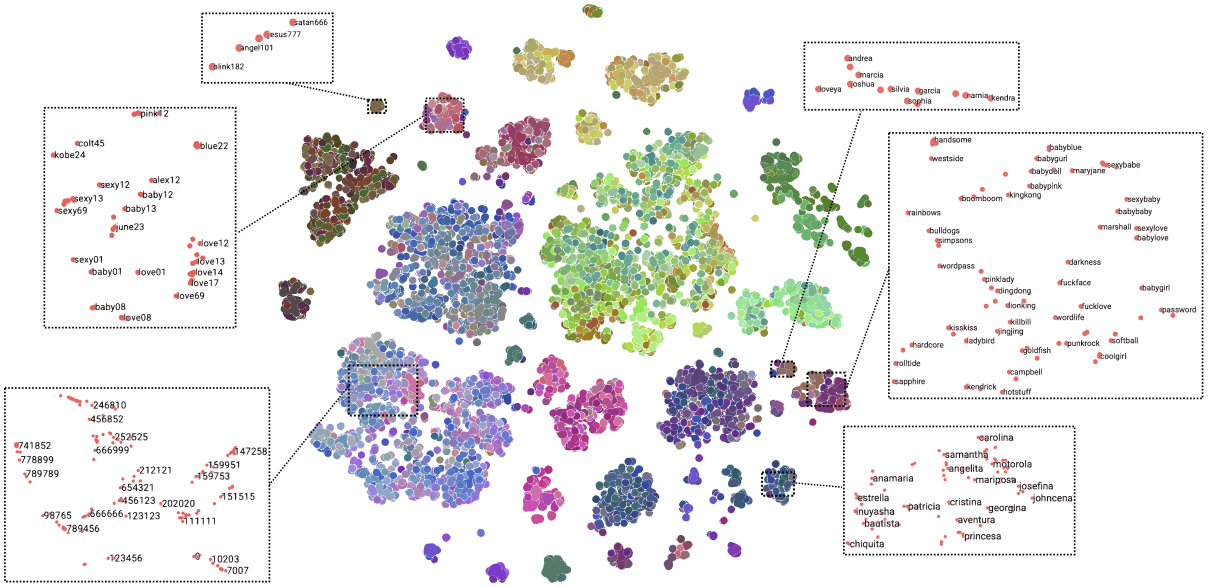
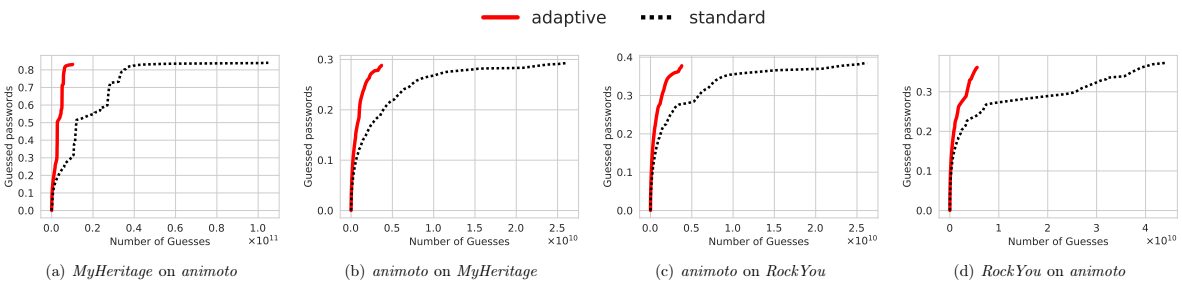


Figure 4.2: Two-dimensional visualization of the dictionary-space representation learned from a model trained on the rule-set *best64* and *LinkedIn*. The visualization is obtained by projection into a two-dimensional space (the output of the last residual block via the *T-SNE* algorithm). Reported colors represent the rules activated from the corresponding dictionary-word. Those are obtained by mapping the label-space to a three-dimensional space via *T-SNE* algorithm. Successively, the latter is mapped into the *RGB* domain, achieving visible colors.

Ultimately, we obtain three different neural networks: one for each rule-set reported in Table 4.1. Summing up, each neural network is an approximation of the compatibility function π_R for the respective rules-set R that is capable of assigning a compatibility score to each rule in $|R|$ with a single network inference i.e., Eq. 4.3. The suitability of these neural approximations will be proven later in the Chapter.

Visualizing the compatibility function The dictionary-space representation learned from the neural net during the training provides a valid intuition over the *compatibility* concept formulated earlier. Indeed, just visualizing it, we can make explicit many of the core assumptions we used to build Section 4.2.1.

Figure 4.2 reports a two-dimensional depiction of such dictionary-space representation obtained using a small set of strings sampled from the *RockYou* password leak. In the figure, every point represents a dictionary-word. Each point's color indicates which rules are activated from the corresponding string in consideration of X_A. These colors are arranged so that similar colors map to sets of active rules with a large intersection.
As shown in the figure, the representation learned from the model organizes dictionary-words that activate the same/similar set of rules close to each other in the space, making explicit sets of clusters that partition the dictionary-space. Intuitively, each of these clusters describes a semantic partition of the dictionary-space, dedicated to activating just a specific subset of mangling rules. Going back to the intuition of Section 4.2.1, such clusters can be seen as the activation domains of the corresponding mangling rules.
Furthermore, the semantic segmentation phenomenon can also be explained by observing samples from the various clusters. These are reported enclosed by dashed rectangles in Figure 4.2.1. In general, in the representation learned by the model, larger clusters group



(a) *MyHeritage* on *animoto* (b) *animoto* on *MyHeritage* (c) *animoto* on *RockYou* (d) *RockYou* on *animoto*

Figure 4.3: Comparison between adaptive and classic mangling rules on four combination password leaks (dictionary/attacked-set) using the rules-set *PasswordPro*. β=0.5 is used for the adaptive case.

macro classes of dictionary-words, such as purely numeric or purely alphabetic strings.[3]. Smaller clusters, instead, tend to bound more specific patterns. For example, clusters on the picture's bottom right indicate how the representation binds words representing first names. It is important to note that no semantic features are given to the model. The model learns this semantic relation solely from dictionary-words' functional meaning; that is, two strings are "*similar*" as they activate the same rules.

Additional approaches To improve the performance of our method, we further investigated domain-specific constructions for multi-label classification. In particular, we tested label embedding techniques. Those are approaches that aim at modeling, implicitly, the correlation among labels. However, although unconditional dependence is evident in the modeled domain, we found no concrete advantage in directly considering it during the training. In the same direction, we investigated more sophisticated embedding techniques, where labels and dictionary-words were jointly mapped to the same latent space [119], yet achieving similar performance.

Additionally, we tested implementations based on *transformer* networks [109], obtaining no substantial improvement. We attribute such a result to the lack of dominant long-term relationships among characters composing dictionary-words. In such a domain, we believe convolutional filters to be fully capable of capturing characters' interactions. Furthermore, convolutional layers are significantly more efficient than the multi-head attention mechanism used by *transformer* networks.

4.2.3 Adaptive Mangling Rules

As motivated in Section 4.2.2, each word in the dictionary interacts just with a limited number of mangling transformations that are conditionally defined by users' composition habits. While modern rules-sets can contain more than ten thousand entries, each dictionary-word w will interact only with a small subset of **compatible rules**, say R_w. As stated before, optimized configurations compose over pairs of dictionaries and rule-sets that have been created to mutually support each other. This is achieved by implicitly maximizing the average cardinality of the compatible set of rules R_w for each dictionary-word w in the dictionary.

[3]As easily predictable, the mangling rules that activate these macro classes of dictionary-words tend to not overlap.

In doing so, advanced attackers rely on domain knowledge and intuition to create optimized configurations. But, thanks to the explicit form of the compatibility function, it is possible to simulate their expertise. The intuition is that, given a dictionary-word w, we can infer the **compatible rules-set** R_w (i.e., the set of rules that interact well with w) according to the compatibility scores assigned by the neural approximation of π. More formally, given π for the rules-set R and a dictionary-word w, we can determine the compatible rules-set for w by *thresholding* the compatibility values assigned by the neural network to the rules in R:

$$R_w \approx R_w^\beta = \{r \mid r \in R \wedge \pi(w, r) > (1 - \beta)\}, \tag{4.5}$$

where $\beta \in (0, 1]$ is a threshold parameter whose effect will be discussed later.

At this point, we simulate high-quality configuration attacks by ensuring dictionary-words does not interact with rules outside its compatible rules-set R_w^β. Algorithm 4 implements this strategy by following a word-major order in the generation of guesses. Every dictionary-word is limited to interact with the subset of compatible rules R_w^β that is decided by the neural net. **Intuitively, this is equivalent to assigning and applying a dedicated (and possibly unique) rules-set to each word in the dictionary.** Note that, the selection of the compatible rules-set is performed at runtime, during the attack, and does not require any pre-computation. We call this novel guessing strategy **Adaptive Mangling Rules**, since the rule-set is continuously adapted during the attack to better assist the selected dictionary.

The efficacy of *adaptive* mangling rules over the standard attack is shown in Figure 4.3, where multiple examples are reported. The adaptive mangling rules reduce the number of produced guesses while maintaining the hits count mostly unchanged. In our experiments, the adaptive approach induces compatible rules-sets that, on average, are an order of magnitude smaller than the complete rules-set. Typically, for β=0.5, only $\sim$ 10%/15% of the rules are conditionally applied to the dictionary-words. Considering the percentage of guessed passwords for adaptive and non-adaptive attacks, this means that approximately 90% of guesses are wasted during classic, unoptimized mangling rules attacks. Figure 4.4 further reports the distribution of selected rules during the adaptive attack of Figure 4.3(a). It emphasizes how mangling rules heterogeneously interact with the underlying dictionary. Although very few rules interact well with all the words (e.g., selection frequency is $> 70\%$), most of the mangling rules participate only in rare events.

Further empirical validation for the adaptive mangling rules will be given later in Section 4.4.

The Attack Budget Unlike standard dictionary attacks, whose effectiveness solely depends on the initial configuration, adaptive mangling rules can be controlled by an

Algorithm 4: Adaptive mangling rules attack.

```
Data: dictonary D, rules-set R, budget β, neural net π_R
forall w ∈ D do
    R_w^β = {r | π_R(w)_r > (1 − β)};
    forall r ∈ R_w^β do
        g = r(w);
        issue g;
```

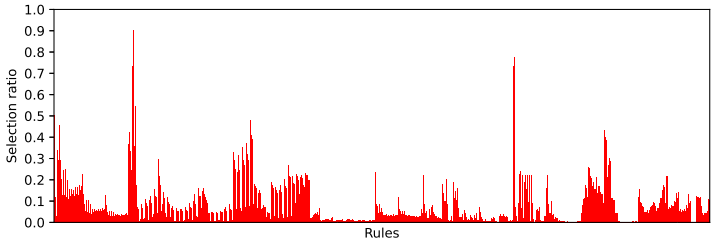
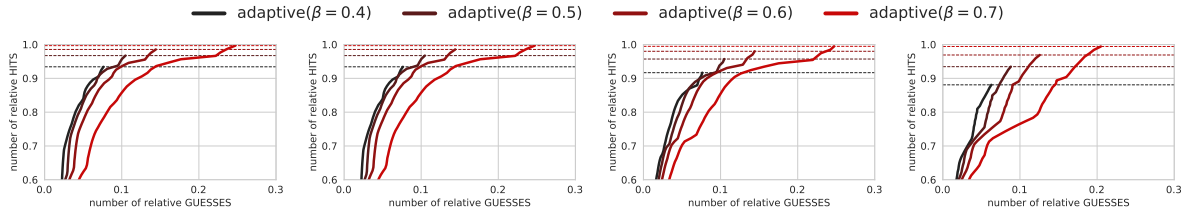


Figure 4.4: Selection frequencies of adaptive mangling rules for the 3120 rules of *PasswordPro*.



(a) *MyHeritage* on *animoto* (b) *animoto* on *MyHeritage* (c) *animoto* on *RockYou* (d) *RockYou* on *animoto*

Figure 4.5: Effect of the parameter β on the guessing performance for four different combinations of password sets and *PasswordPro* rules. Plots are normalized according to the results of the standard mangling rules attack (i.e., $\beta = 1$). For instance, (x=0.1, y=0.95) means that we guessed 95% of the password guessed with the standard mangling rules attack by performing 10% of the guesses required from the latter.

additional scalar parameter that we refer to as the **attack budget** β. This parameter defines the threshold of compatibility that a rule must exceed to be included in the rules-set R_w^β for a word w. Indirectly, this value determines the average size of compatible rules-sets, and consequently, the total number of guesses performed during the attack. More precisely, low values of β force compatible rule-sets to include only rules with high-compatibility. Those will produce only a limited number of guesses, inducing very precise attacks at the cost of missing possible hits (i.e., high precision, low recall). Higher values of β translate in a more permissive selection, where also rules with low-compatibility are included in the compatible set. Those will increase the number of produced guesses, inducing more exhaustive, yet more imprecise, attacks (i.e., higher recall, lower precision). When β reaches 1, the adaptive mangling rules attack becomes a standard mangling rules attack, since all the rules are unconditionally included in the compatible rules-set. The effect of the budget parameter is better captured by the examples reported in Figure 4.5. Here, the performance of multiple values of β is visualized and compared with the total hits and guesses performed by a standard mangling rules attack.

The budget parameter β can be used to model different types of adversaries. For instance, rational attackers [32] change their configuration in consideration of the practical cost of performing the attack. This parameter permit to easily describe those attackers and evaluate password security accordingly. For instance, using a low budget (e.g., β=0.4), we can model a greedy attacker who selects an attack configuration that maximizes guessing precision at the expense of the number of compromised accounts (a rational behavior in case of an expensive hash function).

Seeking a more pragmatic interpretation, the budget parameter is implicitly equiv-

Table 4.2: Number of compatible scores computed per second (c/s) for different networks. Values computed on a single NVIDIA V100 GPU.

generated2 *(large)*	**generated** *(large)*	**PasswordPro** *(large)*
130.550.403 c/s	89.049.382 c/s	31.836.734 c/s

alent to *early-stopping*[4] (i.e., Eq. 4.1), where single guesses are sorted in optimal order i.e., guesses are exhaustively generated before the attack, and indirectly sorted by decreasing probability/compatibility.

The optimal value of β depends on the rules-set. In our tests, we found these optimal values to be 0.6, 0.8 and 0.8 for *PassowordPro*, *generated* and *generated2*, respectively. Hereafter, we use these setups, unless otherwise specified.

Computational cost One of the core advantages of dictionary attacks over more sophisticated approaches [85, 114, 89] is their speed. For mangling rules attacks, generating guesses has almost a negligible impact. Despite being consistently more complex in their mechanisms, adaptive mangling rules do not change this feature.

In Algorithm 4, the only additional operation over the standard mangling rules attack is the selection of compatible rules for each dictionary-word via the trained neural net. As discussed in Section 4.2.2, this operation requires just a single network inference to be computed; that is, with a single inference, we obtain a compatibility score for each element in $\{w\} \times R$. Furthermore, inference for multiple consecutive words can be trivially batched and computed in parallel, further reducing the computation's impact.

Table 4.2 reports the number of compatibility values that different neural networks can compute per second. In the table, we used our largest networks without any form of optimization. Nevertheless, the overhead over the plain mangling rules attack is minimal (see Appendix B.5). Additionally, similar to standard dictionary attacks, adaptive mangling rules attacks are inherently parallel and, therefore, distributed and scalable.

4.3 Dynamic Dictionary attacks

This section introduces the second and last component of our password model—a dynamic mechanism that systematically adapts the guessing configuration to the unknown attacked-set. In Section 4.3.1, we introduce the **Dynamic Dictionary Augmentation** technique. Next, in Section 4.3.2, we introduce the concept of a **Dynamic Budgets**.

Motivation: As widely documented [33, 46, 84], password composition habits slightly change from sub-population to sub-population. Although passwords tend to follow the same general distribution, credentials created under different environments exhibit unique biases. Users within the same group usually choose passwords related to each other, influenced mostly by environmental factors or the underlying applicative layer. Major factors, for example, are users' mother tongue [46], community interests [118] and, imposed password composition policies [73]. These have a significant impact on defining the final password distribution, and, consequently, the *guessability* of the passwords [67].

[4]The attack stops before the guesses are terminated.

The same factors that shape a password distribution are generally available to the attackers who can collect and use them to drastically improve the configuration of their guessing attacks.

Unfortunately, current automatic reactive/proactive guessing techniques fail to describe this natural adversarial behavior [115, 67, 84, 108, 76]. Those methods are based on static configurations that apply the same guessing strategy to each attacked-set of password, mostly ignoring trivial information that can be either *a priori* collected or distilled from the running attack. In this Section, we discuss suitable modifications of the mangling-rules framework to describe a more realistic guessing strategy. In particular, avoiding the necessity of any prior knowledge over the attacked-set, we rely on the concept of **dynamic attack** previously introduced in Section 3.3. That is, a dynamic adversary who changes his guessing strategy according to the attack's success rate. Successful guesses are used to select future attempts with the goal of exploiting the non-i.i.d. of passwords originated from the same environment. Similarly, this general guessing approach can be easily linked to the optimal guessing strategy harnessed from human experts in [108], where mangling rules were created at execution time based on the initially guessed passwords.

4.3.1 Dynamic Dictionary Augmentation

In the GAN-based approach of Chapter 3, the dynamic adaptation of the guessing strategy is obtained from password latent space manipulations of deep generative models. A similar effect is reproduced within our mangling rules approach by relying on a consistently simpler, yet powerful, solution based on hit-recycling. That is, every time we guess a new password by applying a mangling rule over a dictionary word, we insert the guessed password in the dictionary at runtime. In practice, **we dynamically augment the dictionary during the attack using the guessed passwords**.[5] In the process, every new hit is directly reconsidered and semantically extended through mangling rules. This recursive method brings about massive chains/trees of hits that can extend for thousands of levels.[6]

Figure 4.6 depicts an extremely small subtree ("*hits-tree*") obtained by attacking the password leak *phpBB*. The tree starts when the word "*steph*" is mangled, incidentally producing the word "*phpphp*". Since the latter lies in a dense zone of the attacked set (i.e., it is a common users' practice to insert the name of the website or related strings in their password), it induces multiple hits and causes the attack to focus in that specific zone of the key-space. The focus of the attack grows exponentially hit after hit and automatically stops only when no more passwords are matched. Eventually, this process makes it possible to guess passwords that would be missed with the static approach. For instance, in Figure 4.6, all the nodes in bold are passwords matched by the *dynamic* attack but missed by the *static* one (i.e., standard dictionary attack) under the same configuration.

Figure 4.7 compares the guessing performance of the dynamic attack against the static version on a few examples for the *PasswordPro* rules-set. The plots show that the dynamic augmentation of the dictionary has a very heterogeneous effect on the guessing attacks. In the case of Figure 4.7(a), the dynamic attack produces a substantial increment in the

[5] Although we have not found any direct reference to the *hits-recycling* technique in the literature, it is likely well known and routinely deployed by professionals.

[6] I.e., a forest, where the root of each tree is a word from the original dictionary.

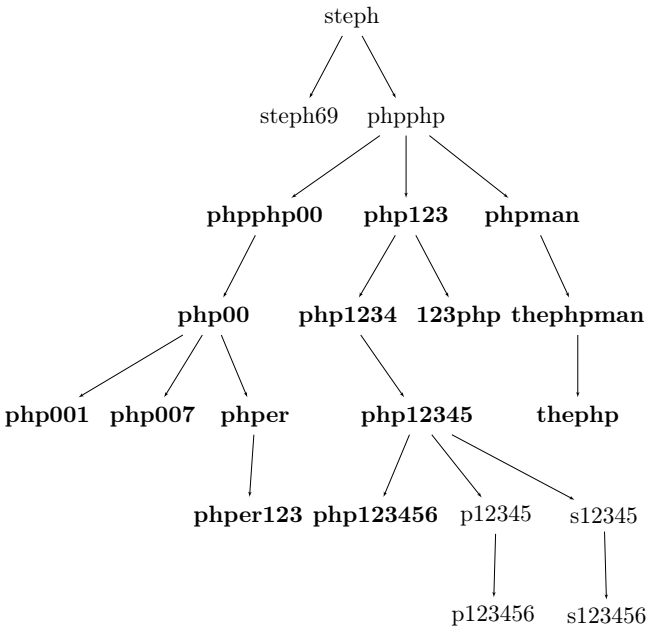
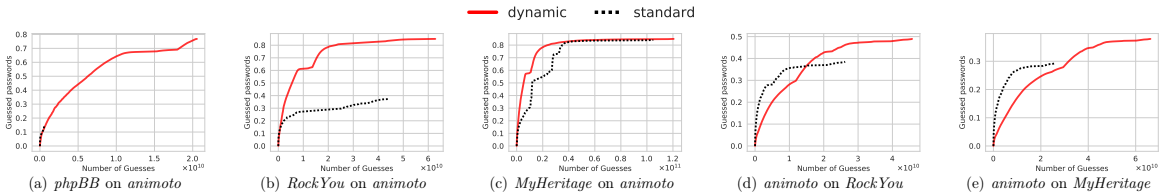


Figure 4.6: Example of small *hits-tree* induced by the dynamic attack performed on the *phpBB* leak. In the tree, every vertex is a guessed password; an edge between two nodes indicates that the child password has been guessed by applying a mangling rule to the parent password.

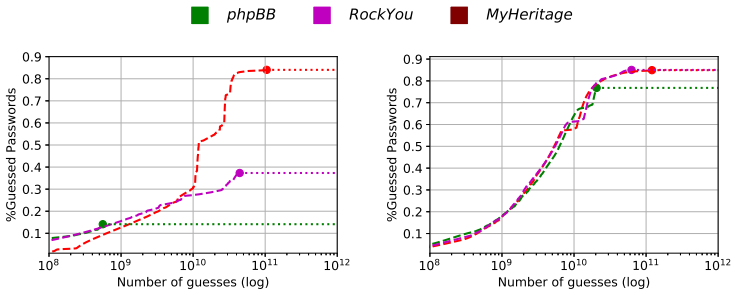


(a) *phpBB* on *animoto* (b) *RockYou* on *animoto* (c) *MyHeritage* on *animoto* (d) *animoto* on *RockYou* (e) *animoto* on *MyHeritage*

Figure 4.7: Performance comparison between dynamic and classic (static) attack for five different setups of dictionary/attacked-set. The rules set *PasswordPro* in **non-adaptive mode** is used in all the reported attacks. The 5 setups have been handpicked to fully represent the possible effects of the dynamic dictionary augmentation.

number of guesses as well as in the number of hits i.e., from $\sim 15\%$ to $\sim 80\%$ recovered passwords. Arguably, such a gap is due to the minimal size of the original dictionary *phpBB*. In the attack of Figure 4.7(b), instead, a similar improvement is achieved by requiring only a small number of guesses. On the other hand, in the attack depicted in Figure 4.7(c), the dynamic augmentation has a limited effect on the final hits number. However, it increases the attack precision in the initial phase. Conversely, attacks in Figures 4.7(d) and 4.7(e) show a decreased precision in the initial phase of the attack, but that is compensated later by the dynamic approach. The same results are reported in Appendix B.4 for the rules-sets *generated* and *generated2*.

Another interesting property of the dynamic augmentation is that it makes the guessing attack consistently less sensitive to the choice of the input dictionary. Indeed, in contrast with the static approach, different choices of the initial dictionary tend to produce very homogeneous results in the dynamic approach. This behavior is captured in Figure 4.8, where results, obtained by varying three input dictionaries, are compared between static and dynamic attack. The standard attacks (Figure 4.8(a)) result in very



(a) standard attack (b) dynamic attack

Figure 4.8: Guessing attacks performed on the *animoto* leak using three different dictionaries. The panel on the left reports the guessing curves for the static setup. The panel on the right reports those for the dynamic setup. The x-axis is logarithmic.

different outcomes; for instance, using *phpBB* we match 15% of the attacked-set, whereas we match more than 80% with *MyHeritage*. These differences in performance are leveled out by the dynamic augmentation of the dictionary (Figure 4.8(b)); all the dynamic attacks recover $\sim$ 80% of the attacked-set. Intuitively, dynamic augmentation remedies deficiencies in the initial configuration of the dictionary, promoting its completeness. These claims will find further support in Section 4.4.

4.3.2 Dynamic budgets

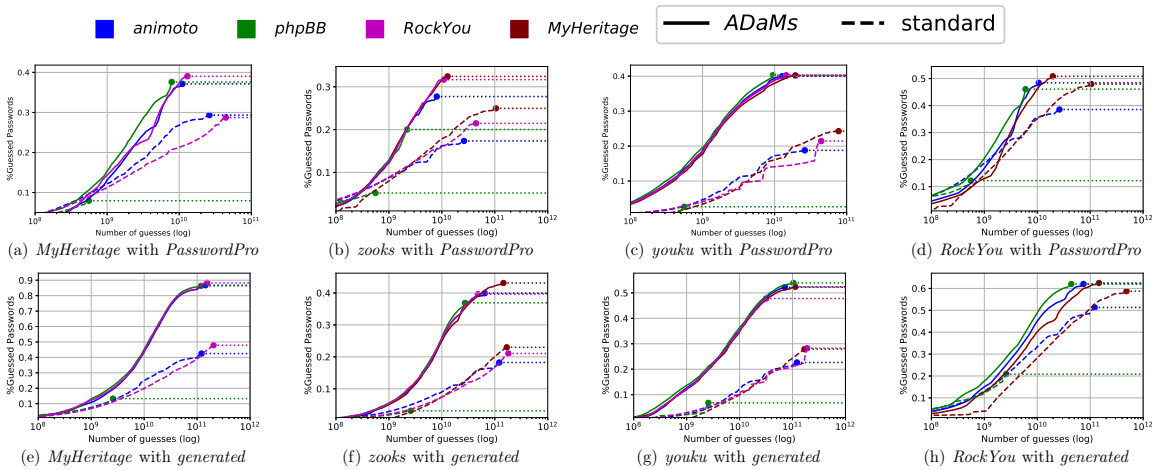
Adaptive mangling rules (Section 4.2.3) demonstrated that it is possible to consistently improve the precision of the guessing attack by promoting compatibility among rules-set and dictionary (i.e., simulating high-quality configurations at runtime). This approach assumes that the compatibility function modeled before the attack is sufficiently general to simulate good configurations for each possible attacked-set. However, as motivated in the introduction of Section 4.3, every attacked set of passwords present peculiar biases and, therefore, different compatibility relations among rules and dictionary-words.
To reduce the effect of this dependence, we introduce an additional dynamic approach supporting the adaptive mangling rules framework. Rather than modifying the neural network at runtime (which is neither a practical nor a reliable solution), we alter the selection process of compatible rules by acting on the budget parameter β.

Algorithm 5 details our solution. Here, rather than having a global parameter β

Algorithm 5: Adaptive rules with Dynamic budget

Data: dictonary D, rules-set R, attacked-set X, budget β

forall $w \in D$ **do**
  $R_w^\beta = \{r | \pi_R(w)_r > (1 - B_i)\}$;
  forall $r \in R_w^\beta$ **do**
    $g = r(w)$;
    if $g \in X$ **then**
      $X = X - \{g\}$;
      $B_r = B_r + \Delta$;
    $B = B \cdot \frac{\sum^{|B|} \beta}{\sum^{|B|} B}$;



(a) *MyHeritage* with *PasswordPro* (b) *zooks* with *PasswordPro* (c) *youku* with *PasswordPro* (d) *RockYou* with *PasswordPro*

(e) *MyHeritage* with *generated* (f) *zooks* with *generated* (g) *youku* with *generated* (h) *RockYou* with *generated*

Figure 4.9: Each plot reports the number of guesses (in log scale) and the percentage of matched passwords for different rule-sets and dictionaries against several attacked-sets. Each row reports a rule-set, whereas each column identifies an attacked-set. We use four dictionary, each identified by a colored line. Continuous lines show *AdaMs* attack whereas dashed lines refer to standard mangling rules attacks.

for all the rules of the rules-set R, we have a budget vector B that assigns a dedicated budget value to each rule in R (i.e., $B \in (0,1]^{|R|}$). Initially, all the budget values in B are initialized to the same value β (i.e., $\forall_{r \in R}\ B_r = \beta$) given as an input parameter. During the attack, the elements of B are individually increased and decreased to better describe the attacked set of passwords. Within this context, increasing the budget B_r of a rule r means reducing the compatibility threshold needed to include r in the compatible rules-set of a dictionary-word w, and, consequently, making r more popular during the attack. On the other hand, by decreasing B_r, we reduce the chances of selection for r; r is selected only in case of high-compatibility words.
In the algorithm, we increase the budget B_r when the rule r produces a hit . The added increment is a small value Δ that scales inversely with the number of guesses produced.

At the end of the internal loop, the vector B is then normalized; i.e., we scale the values in B so that $\sum_r^R B = \sum_i^{|R|} \beta$. Normalizing B has two aims. **(1)** It reduces the budgets for non-hitting rules (the mass we add to the budget of rule r is subtracted from all other budgets.). **(2)** It maintains the total budget of the attack (i.e., $\sum_i^{|R|} \beta$) unchanged so that dynamic and static budget leads to almost the same number of guesses during the attack for a given β. Furthermore, we impose a maximum and a minimum bound on the increments or decrements of B. This is to prevent values of zero (rule always excluded) or equal/higher than one (rule always included).

As for the dynamic dictionary augmentation, the dynamic budget has always a positive, but, heterogeneous, effect on the guessing performance. Mostly, the number of hits increases or remains unaffected. Among the proposed techniques, this is the one with the mildest effect. Yet, this will be particularly useful when combined with dynamic dictionary augmentation in the next Section. Appendix B.3 better explicates the improvement induced from the dynamic budgets.

4.4 Adaptive, Dynamic Mangling rules: *AdaMs*

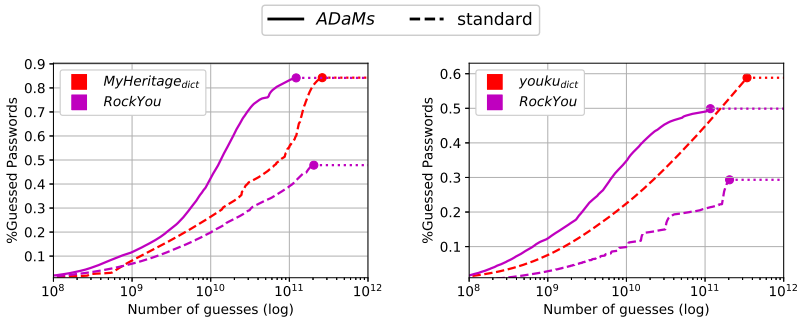
The results of the previous section confirm the effectiveness of the dynamic guessing mechanisms. We increased the number of hits compared to classic dictionary attacks by using the produced guesses to improve the attack on the fly. However, in the process, we also increased the number of guesses, possibly in a way that is hard to control and gauge. Moreover, by changing the dictionary at runtime, we disrupt any form of optimization of the initial configuration, such as any *a priori* ordering in the wordlist [76] and any joint optimization with the rules-set[7]. Unavoidably, this leads to sub-optimal attacks that may overestimate passwords strength.
To mitigate this phenomenon, we combine the dynamic augmentation technique with the Adaptive Mangling Rules framework. The latter seeks an optimal configuration at runtime on the dynamic dictionary, promoting compatibility with the rules-set and limiting the impact of imperfect dictionary-words. This process is further supported by the dynamic budgets that address possible covariate-shifts [102] of the compatibility function induced by the augmented dictionary.

Hereafter, we refer to this final guessing strategy as *AdaMs* (**A**daptive, **D**yn**a**mic **M**angling rule**s**). Details on the implementation of *AdaMs* are given in Appendix B.6, whereas we benchmark it in Appendix B.5.

4.4.1 Evaluation

Figure 4.9 reports an extensive comparison of *AdaMs* against standard mangling-rules attacks. In the figure, we test all pairs of dictionary/rule-set obtained from the combination of the dictionaries: *MyHeritage*, *RockYou*, *animoto*, *phpBB* and the rules-sets: *PasswordPro* and *generated* on four attacked-sets. Results for *generated2* are reported in Appendix B.4 instead. Hereafter, we switch to a logarithm scale given the heterogeneity of the number of guesses produced by the various configurations.
For the reasons given in the previous sections, *AdaMs* outperforms standard mangling rules within the same configurations, while requiring fewer guesses on average. More interestingly, *AdaMs* attacks generally exceed the hits count of all the standard attacks regardless of the selected dictionary. In particular, this is always true for the *generated* rules-set.
Conversely, in cases where the dynamic dictionary augmentation offers only a small gain in the number of hits (e.g., attacking *RockYou*), *AdaMs* equalizes the performance of various dictionaries, typically, towards the optimal configuration for the standard attack. In Figures 4.9(d) and 4.9(h), all the configurations of *AdaMs* reach a number of hits comparable to the best configuration for the standard attack, i.e., using *MyHeritage*, while requiring up to an order of magnitude fewer guesses (e.g., Figure 4.9(d)), further confirming that the best standard attack is far from being optimal. In the reported experiments, the only outlier is *phpBB* when used against *zooks* in Figure 4.9(b). Here, *AdaMs* did not reach/exceed all the standard attacks in the number of hits despite consistently redressing the initial configuration. However, this discrepancy is canceled out when more mangling rules are considered i.e., in Figure 4.9(f).
Eventually, the *AdaMs* attack makes the initial selection of the dictionary systematically less influential. For instance, in our experiments, a set such as *phpBB* reaches the same performance of wordlists that are two orders of magnitude larger (e.g., *RockYou*). The

[7] I.e., new words may not interact well with the mangling rules in use.



(a) $MyHeritage_{\text{target}}$ (b) $youku_{\text{target}}$

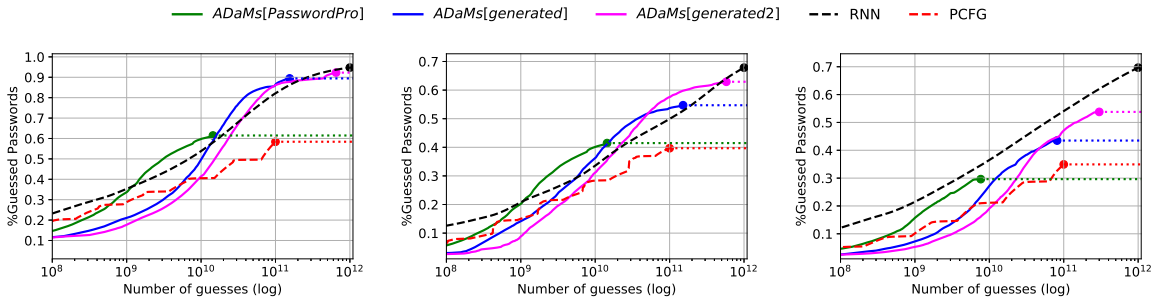
Figure 4.10: Comparison of *AdaMs* against optimal dictionary for two sets of passwords.

crucial factor remains the rules-set's cardinality that ultimately determines the magnitude of the attack, even though it does not appreciably affect the guessing performance.

The effectiveness of *AdaMs* is better captured by the results reported in Figure 4.10. Here, we create a synthetic optimal dictionary for an attacked-set and evaluate the capability of *AdaMs* to converge to the performance of such an optimal configuration. To this end, given a password leak X, we randomly divide it in two disjointed sets of equal size, say X_{dict} and X_{target}. Then, we attack X_{target} by using both X_{dict} (i.e., optimal dictionary) and an external dictionary (i.e., sub-optimal dictionary). Arguably, X_{dict} is the *a priori* optimal dictionary to attack X_{target} since X_{dict} and X_{target} are samples of the very same distribution.
We report the results for two sets: *MyHeritage* and *youku*. The attacks are carried out by using the rules-set *generated* and *RockYou* as the external dictionary.
In the case of *MyHeritage*, the *AdaMs* attack is more precise than the optimal dictionary and produces a comparable number of hits. Similarly, in the case of *youku*, the *AdaMs* attack guesses faster than the optimal dictionary within the first 10^{11} guesses. However, in this case, it does not reach an equivalent number of guessed passwords. We can attribute this to the high discrepancy between the initial dictionary *RockYou* and the attacked-set *youku* that cannot be bridged without prior knowledge.[8] Nevertheless, the dictionary augmentation technique can induce a dictionary that has a comparable utility to one of the best optimal *a priori* setup, while requiring no information on the attacked-set. In the process, the adaptive framework consistently accounts for the noise introduced by the augmentation, allowing *AdaMs* to be even more precise than the optimal dictionary for most of the attack (i.e., within the first 10^{11} guesses). Further comparison with other password models follow.

Figure 4.11 reports a direct comparison against the RNN-based approach of Melicher et al. [85] and PCFG [114]. The RNN-based password model is the state-of-the-art for password strength estimation, although its computational cost in generating guesses makes it impractical for real password guessing. We train the model using *RockYou* and simulate password guessing attacks using [47]. In the process, we use default parameters of the available software [17] and consider passwords with guess-number within 10^{12}.
PCFG is the academic approach that better mirrors the guessing generation process of dictionary attacks. We train the PCFG-based model on *RockYou* using the default setting [19]. In this case, we limit to the first 10^{11}.

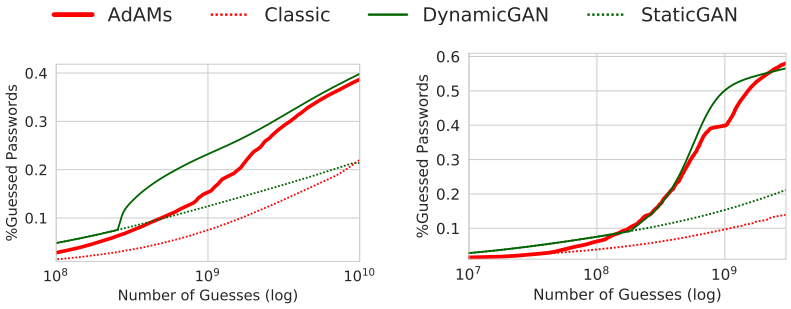
[8]The leak *youku* is mostly composed of Chinese passwords that are underrepresented in *RockYou*.



(a) *MyHeritage* (b) *youku* (c) *zooks*

Figure 4.11: Comparison of the *AdaMs* attacks against he RNN-based approach of Melicher et al. [85] and PCFG [114] for three password leaks.

We compare the models on three leaks: *MyHeritage*, *youku* and *zooks*. For the *AdaMs* attacks, we use *RockYou* as a dictionary, whereas we report results for three rules-sets.
Surprisingly, the *AdaMs*reach performance very close to the one obtained from the RNN-based model. It even outperforms the parametric attack in two of the three attack-sets. Similarly, *AdaMs* tend to perform better than PCFG in the three cases, especially after the initial guesses. Furthermore, Figure 4.12 compares *AdaMs* against the GAN-based dy-



(a) *youku* (b) *zomato*

Figure 4.12: Performance comparison between *AdaMs* and the GAN-based dynamic attack. Classic mangling rules attack and StaticGAN are reported as baseline.

namic attack described in Chapter 3. We base the comparison on the same leaks; namely, the *youku* and *zomato* leak (details given in Table B.1). The GAN-based model is trained on the *RockYou* leak and the attack is performed with the same hyper-parameters used in the original setup: $\sigma = 0.35$ and *hot-start* $\alpha = 10\%$. Despite our simpler approach, the *AdaMs* attack performs very similarly to the GAN-based attack, besides being significantly faster in generating guesses.

4.5 Conclusion

The *AdaMs* attack autonomously pushes the attack strategy towards the optimal one, producing password strength estimates that better model actual adversarial capabilities. As shown in Figure 4.9, the approach also makes the guessing attack more resilient to

deficiencies in the initial configuration, reducing the bias induced by non-expert setups. In this direction, the *AdaMs* attack further proves the intrinsic unsuitability of arbitrarily chosen configurations and the overestimation of password security that those can induce. Compared with other systems [85], our framework provides researchers and security practitioners with a markedly more efficient and flexible solution. **We advocate the proposed technique as a necessary substitute for standard dictionary attacks in the evaluation of password strength as this fundamentally better represents real-world attackers' capabilities and attack strategies.**

Part II

Proactive Mechanisms

In the first part of the thesis, we showed how deep learning techniques can be used to improve/enable guessing attacks. In this Chapter, we demonstrate how these same techniques that threaten passwords can be used to improve their security and soundness. Indeed, deep learning models can be used to obtain compact and precise descriptions of password probabilities. Such estimates, in turn, can be used to cast accurate password meters capable of guiding users towards passwords that are resilient to guessing attacks. Furthermore, these models permit novel kind of inference over the modeled distribution, allowing us to construct unprecedented interpretable mechanisms that support users during the composition process.

In Chapter 5 we introduce a new estimation process for password distributions that enables the creation of the **first interpretable probabilistic password strength meter**.

Chapter 5

Towards Interpretable Password Strength Meters

Accurately measuring password strength is essential to guarantee the security of password-based authentication systems. Even more critical, however, is training users to select secure passwords in the first place. One common approach is to rely on password policies that list a series of requirements for a strong password. This approach is limited or even harmful [43]. Alternatively, Passwords Strength Meters (PSMs) have been shown to be useful and are witnessing increasing adoption in commercial solutions [106, 56].
The first instantiations of PSMs were based on simple heuristic constructions. Password strength was estimated via either handcrafted features such as *LUDS* (which counts lower and uppercase letters, digits, and symbols) or heuristic entropy definitions. Unavoidably, given their heuristic nature, this class of PSMs failed to accurately measure password security [46, 113].
More recently, thanks to an active academic interest, PSMs based on more sound constructions and rigorous security definitions have been proposed. In the last decade, indeed, a considerable research effort gave rise to more precise meters capable of accurately measuring password strength [85, 111, 40].
However, meters have also become proportionally more opaque and inherently hard to interpret due to the increasing complexity of the employed approaches. State-of-the-art solutions base their estimates on blackbox parametric probabilistic models [85, 40] that leave no room for interpretation of the evaluated passwords; they do not provide any feedback to users on what is wrong with their password or how to improve it. We advocate for explainable approaches in password meters, where users receive additional insights and become cognizant of which parts of their passwords could straightforwardly improve. This makes the password selection process less painful since users can keep their passwords of choice mostly unchanged while ensuring they are secure.

In this Chapter, we show that the same rigorous probabilistic framework capable of accurately measuring password strength can also fundamentally describe the relation between password security and password structure. By rethinking the underlying mass estimation process, we create the first *interpretable probabilistic password strength* meter. Here, the password probability measured by our meter can be decomposed and used to estimate further the strength of every single character of the password. This explainable approach allows us to assign a security score to each atomic component of the password and determine its contribution to the overall security strength. This evaluation is, in turn, returned to the user who can tweak a few "weak" characters and consistently im-

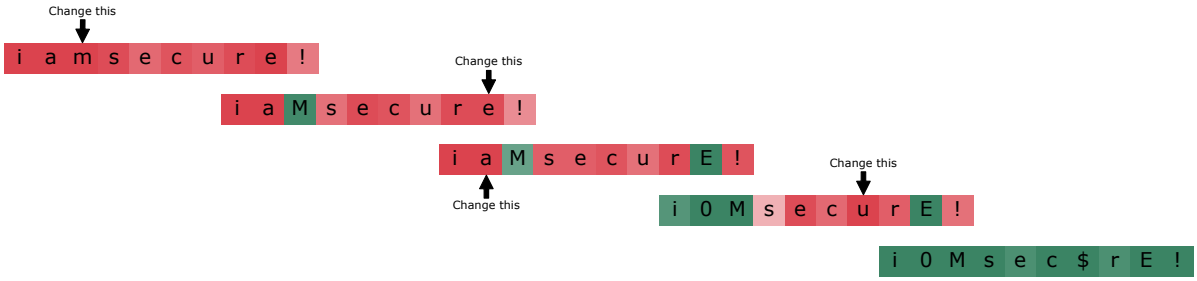


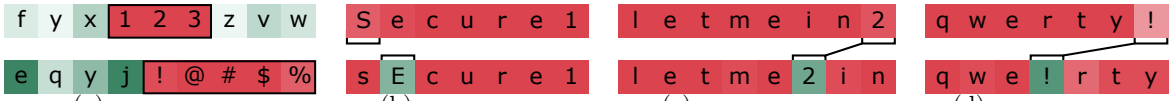
Figure 5.1: Example of the character-level feedback mechanism and password composition process induced by our meter. In the figure, "*iamsecure!*" is the password initially chosen by the user. Colors indicate the estimated character security: red (insecure) → green (secure).

prove the password strength against guessing attacks. Figure 5.1 illustrates the selection process. In devising the proposed mass estimation process, we found it ideally suited for being implemented via a deep learning architecture. In this chapter, we show how that can be cast as an efficient client-side meter employing deep convolutional neural networks. Our work's major contributions are: (i) We formulate a novel password probability estimation framework based on non-autoregressive probabilistic models. (ii) We show that such a framework can be used to build a precise and sound password feedback mechanism. (iii) We implement the proposed meter via an efficient and lightweight deep learning framework ideally suited for client-side operability.

5.1 Related Works

Here, we briefly review early approaches to the definition of PSMs. We focus on the most influential works as well as to the ones most related to ours.

Probabilistic PSMs: Originally thought for guessing attacks [89], Markov model approaches have found natural application in the password strength estimation context. Castelluccia et al. [40] use a stationary, finite-state Markov chain as a direct password mass estimator. Their model computes the joint probability by separately measuring the conditional probability of each pair of n-grams in the observed passwords. Melicher et al. [85] extended the Markov model approach by leveraging a character/token level Recurrent Neural Network (RNN) for modeling the probability of passwords. As discussed in the introduction, pure probabilistic approaches are not capable of any natural form of feedback. In order to partially cope with this shortcoming, a hybrid approach has been investigated in [104]. Here, the model of Melicher et al. [85] is aggregated with a series of 21 heuristic, hand-crafted feedback mechanisms such as detection of *leeting behaviors* or common tokens (e.g., keyboard walks).
Even if harnessing a consistently different form of feedback, our framework merges these solutions into a single and jointly learned model. Additionally, in contrast with [104], our feedback has a concrete probabilistic interpretation as well as complete freedom from any form of human bias. Interestingly enough, our model autonomously learns some of the heuristics hardwired in [104]. For instance, our model learned that capitalizing characters in the middle of the string could consistently improve password strength (see Figure 5.2).



(a) Common tokens. (b) Cap. first/inner. (c) Numeric last/inner (d) Special last/inner.

Figure 5.2: In Panel (a), the model automatically highlights the presence of weak substrings by assigning high probabilities to the characters composing them. Panels (b), (c), and (d) are examples of self-learned weak/strong password composition patterns. In panel (b), the model assigns a high probability to the capitalization of the first letter (a common practice), whereas it assigns low probability when the capitalization is performed on inner characters. Panel (c) and (d) report similar results for numeric and special characters.

Token look-up PSMs: Another relevant class of meters is that based on the token look-up approach. Generally speaking, these are non-parametric solutions that base their strength estimation on collections of sorted lists of tokens like leaked passwords and word dictionaries. Here, a password is modeled as a combination of tokens, and the relative security score is derived from the ranking of the tokens in the known dictionaries. Unlike probabilistic solutions, token-based PSMs are able to return feedback to the user, such as an explanation for the weakness of a password relying on the semantic attributed to the tokens composing the password. A leading member of token look-up meters is *zxcvbn* [115], which assumes a password as a combination of tokens such as *token, reversed, sequence repeat, keyboard, and date*. This meter scores passwords according to a heuristic characterization of the guess-number [83]. Such score is described as the number of combinations of tokens necessary to match the tested password by traversing the sorted tokens lists.
zxcvbn is capable of feedback. For instance, if one of the password components is identified as *"repeat"*, *zxcvbn* will recommend the user to avoid the use of repeated characters in the password. Naturally, this kind of feedback mechanism inherently lacks generality and addresses just a few human-chosen scenarios. As discussed by the authors themselves, *zxcvbn* suffers from various limitations. By assumption, it is unable to model the relationships among different patterns occurring in the same passwords. Additionally, like other token look-up based approaches, it fails to coherently model unobserved patterns and tokens.

Another example of token look-up approach is the one proposed in [72]. *Telepathwords* discourages a user from choosing weak passwords by predicting the next most probable characters during the password typing. In particular, predicted characters are shown to the user in order to dissuade him/her from choosing them as the next characters in the password. These are reported together with an explanation of why those characters were predicted. However, as for *zxcvbn*, such feedback solely relies on hardwired scenarios (for instance, the use of profanity in the password). *Telepathwords* is server-side only.

5.2 Meter foundations

In this section, we introduce the theoretical foundations of the proposed estimation process. First, in Section 5.2.1, we introduce and motivate the probabilistic character-level feedback mechanism. Later, in Section 5.2.2, we describe how that mechanism can be obtained using a non-autoregressive probabilistic models.

5.2.1 Character-level strength estimation via probabilistic model

As introduced in Section 2.1.3, PPSMs employ probabilistic models to approximate the probability mass function of an observed password distribution, say $P(\mathbf{x})$. Estimating $P(\mathbf{x})$, however, could be particularly challenging, and suitable estimation techniques must be adopted to make the process feasible. In this direction, a general solution is to factorize the domain of the mass function (i.e., the key-space); that is, passwords are modeled as a concatenation of smaller factors, typically, decomposed at the character level. Afterward, password distribution is estimated by modeling stochastic interactions among these simpler components. More formally, every password is assumed as a realization $x = [x_1, \ldots, x_\ell]$ of a random vector of the kind $\mathbf{x} = [\mathbf{x}_1, \ldots, \mathbf{x}_\ell]$, where each disjoint random variable $\mathbf{x}_i$ represents the character at position i in the string. Then, $P(\mathbf{x})$ is described through probabilistic models that formalize the relations among those random variables, eventually defining a joint probability distribution. In the process, every random variable is associated with a **local conditional probability distribution** (here, referred to as Q) that describes the behavior of $\mathbf{x}_i$ in consideration of the underlying model i.e., $Q(\mathbf{x}_i) = P(\mathbf{x}_i \mid par(\mathbf{x}_i))$. Eventually, the joint measurement of probability is derived from the aggregation of the marginalized local conditional probability distributions, typically under the form $P(\mathbf{x}) = \prod_{i=1}^{\ell} Q(\mathbf{x}_i = x_i)$.

As introduced in Section 2.1.3, the joint probability can be employed as a good representative for password strength. However, such a global assessment unavoidably hides much fine-grained information that can be extremely valuable to a password meter. In particular, the joint probability offers us an atomic interpretation of the password strength, but it fails at disentangling the relation between password strength and password structure. That is, it does not clarify which factors of an evaluated password are making that password insecure. However, as widely demonstrated by non-probabilistic approaches [115, 104, 72], users benefit from the awareness of which part of the chosen password is easily predictable and which is not. In this direction, we argue that the **local conditional probabilities** that naturally appear in the estimation of the joint one, if correctly shaped, can offer detailed insights into the strength or the weakness of each factor of a password. **Such character-level probability assignments are an explicit interpretation of the relation between the structure of a password and its security.** The main intuition here is that: high values of $Q(x_i)$ tell us that x_i (i.e., the character at position i in the string) has a high impact on increasing the password probability and must be changed to make the password stronger. Instead, characters with low conditional probability are pushing the password to have low probability and must be maintained unchanged. Figure 5.2 reports some visual representations of such probabilistic reasoning. Each segment's background color renders the value of the local conditional probability of the character. Red describes high probability values, whereas green describes low probability assignments. Such a mechanism can naturally discover weak passwords components and explicitly guide the user to explore alternatives. For instance, local conditional probabilities can spot the presence of predictable tokens in the password without the explicit use of dictionaries (Figure 5.2(a)). These measurements are able to automatically describe common password patterns like those manually modeled from other approaches [104], see Figures 5.2(b), 5.2(c) and 5.2(d). More importantly, they can potentially describe latent composition patterns that have never been observed and modeled by human beings. In doing this, neither supervision nor human-reasoning is required.

(a) (b)

Figure 5.3: Estimated local conditional probabilities for two pairs of passwords. The numbers depicted above the strings report the $Q(x_i)$ value for each character (rounding applied).

5.2.2 Our Probabilistic model

In order to build a suitable probabilistic model to cast the feedback mechanism previously described, we resort to a non-autoregressive language model similar to *Word2Vec* (Continuous Bag of Words) [86] and *BERT* [48], but defined at character-level. According to that description, the probability of the character x_i directly depends on any other character in the string, i.e., the full context. In other words, we model each variable $\mathbf{x}_i$ as a stochastic function of all the others. This intuition is better captured from the evaluation of local conditional probability (Figure 5.1).

$$Q(\mathbf{x}_i) = \begin{cases} P(\mathbf{x}_i \mid \mathbf{x}_{i+1}, \dots \mathbf{x}_\ell) & i = 1 \\ P(\mathbf{x}_i \mid \mathbf{x}_1, \dots \mathbf{x}_{i-1}) & i = \ell \\ P(\mathbf{x}_i \mid \mathbf{x}_1, \dots, \mathbf{x}_{i-1}, \mathbf{x}_{i+1}, \dots \mathbf{x}_\ell) & 1 < i < \ell \end{cases} . \tag{5.1}$$

Henceforth, we use the notation $Q(\mathbf{x}_i)$ to refer to the local conditional distribution of the i'th character within the password x. When x is not clear from the context, we write $Q(\mathbf{x}_i \mid x)$ to make it explicit. The notation $Q(\mathbf{x}_i{=}s)$ or $Q(s)$, instead, refers to the marginalization of the distribution according to the symbol s.

Eventually, such undirected formalization allows us to produce a versatile feedback mechanism that ignore the limitations of autoregressive model. Now, every local probability is computed within the context offered by any other symbol in the string. In the example, $y{=}$"*aaaaaaa*" / $z{=}$"*a######*", the local conditional probability of the first character can be backward-influenced from the context offered from the subsequent part of the string. This is clearly observable from the output of an instance of our meter reported in Figure 5.3(a), where the value of $Q(\mathbf{x}_1{=}\text{'}a\text{'})$ drastically varies between the two cases, i.e., y and z. As expected, we have $Q(\mathbf{x}_1{=}\text{'}a\text{'}|y) \gg Q(\mathbf{x}_1{=}\text{'}a\text{'}|z)$ verified in the example. A similar intuitive result is reported in Figure 5.3(b), where the example $x{=}$"*(password)*" is considered. Here, the meter first scores the string $x'{=}$"*(password* ", then it scores the complete password $x{=}$"*(password)*". In this case, we expect that the presence of the last character *')'* would consistently influence the conditional measurement of the first bracket in the string. Such expectation is perfectly captured from the reported output, where appending at the end of the string the symbol *')'* increases the probability of the first bracket of a factor ~ 15.

However, obtaining these improvements does not come for free. Indeed, under this construction, the productory over the local conditional probabilities does not provide the exact joint probability distribution of $\mathbf{x}$. Instead, such product results in a unnormalized version of it: $P(\mathbf{x}) \propto \prod_{i=1}^{\ell} Q(\mathbf{x}_i){=}\tilde{P}(\mathbf{x})$ with $P(x) = \frac{\tilde{P}(\mathbf{x})}{Z}$. In the equation, Z is the partition function. This result follows from the Hammersley–Clifford theorem [71]. Nevertheless, the unnormalized joint distribution preserves the core properties needed to the

meter functionality. Most importantly, we have that:

$$\forall x, x' \ : \ P(x) \geq P(x') \Leftrightarrow \tilde{P}(x) \geq \tilde{P}(x') \quad . \tag{5.2}$$

That is, if we sort a list of passwords according to the true joint $P(\mathbf{x})$ or according to the unnormalized version $\tilde{P}(\mathbf{x})$, we obtain the same identical ordering. Consequently, no deviation from the adversarial interpretation of PPSMs described in Section 2.1.3 is implied. Indeed, we have $\mathbb{X}_{P(\mathbf{x})} = \mathbb{X}_{\tilde{P}(\mathbf{x})}$ for every password distribution, key-space, and suitable sorting function.

Details on the password feedback mechanism and further applications

Joint probability can be understood as a compatibility score assigned to a specific configuration of the probabilistic model; it tells us the likelihood of observing a sequence of characters during the interaction with the password generative process. On a smaller scale, a local conditional probability measures the impact that a single character has in the final security score. Namely, it indicates how much the character contributes to the probability of observing a certain password x. Within this interpretation, low-probabilities characters push the joint probability of x to be closer to zero (secure), whereas high-probability characters (i.e., $Q(x_1) \lesssim 1$) make no significant contribution to lowering the password probability (insecure). Therefore, users can strengthen their candidate passwords by substituting high-probability characters with suitable lower-probability ones (e.g., Figure 5.1).
Unfortunately, users' perception of password security has been shown to be generally erroneous [105], and, without explicit guidelines, it would be difficult for them to select suitable lower-probability substitutes. To address this limitation, one could adopt our approach based on local conditional distributions as an effective mechanism to help users select secure substitute symbols. Indeed, $\forall_i Q(\mathbf{x}_i)$ are able to clarify which symbol is a secure substitute and which is not for each character x_i of x. In particular, a distribution $Q(\mathbf{x}_i)$, defined on the whole alphabet Σ, assigns a probability to every symbol s that the character $\mathbf{x}_i$ can potentially assume. For a symbol $s \in \Sigma$, the probability $Q(\mathbf{x}_i{=}s)$ measures how much the event $\mathbf{x}_i{=}s$ is probable given all the observable characters in x. Under this interpretation, a candidate, secure substitution of x_i is a symbol with very low $Q(\mathbf{x}_i{=}s)$ (as this will lower the joint probability of x). In particular, every symbol s s.t. $Q(\mathbf{x}_i{=}s) < Q(\mathbf{x}_i{=}x_i)$ given x is a secure substitution for x_i. Table 5.1 better depicts this intuition. The Table reports the alphabet sorted by $Q(\mathbf{x}_i)$ for each x_i in the example password $x{=}$"*PaSsW0rD!*". The bold symbols between parenthesis indicate x_i. Within this representation, all the symbols below the respective x_i for each $\mathbf{x}_i$ are suitable substitutions that improve password strength. This intuition is empirically proven in Section 5.4.2. It is important to note that the suggestion mechanism must be randomized to avoid any bias in the final password distribution.[1] To this end, one can provide the user with k random symbols among the pool of secure substitutions, i.e., $\{s \mid Q(\mathbf{x}_i{=}s) < Q(\mathbf{x}_i{=}x_i)\}$.

It is important to highlight that, although we present our approach under the character-level description, our method can be directly applied to n-grams, words and sub-words without any modification.

In summary, in this section, we presented and motivated an estimation process able to unravel the feedback mechanism described in Section 5.2.1. Here, no information about

[1] That is, if weak passwords are always perturbed in the same way, they will be easily guessed.

Rank	•aSsW0rD!	P•SsW0rD!	Pa•sW0rD!	PaS•W0rD!	PaSs•0rD!	PaSsW•rD!	PaSsW0•D!	PaSsW0r•!	PaSsW0rD•
0	**{P}**	A	s	S	w	O	R	d	1
1	S	**{a}**	**{S}**	**{s}**	**{W}**	o	**{r}**	**{D}**	S
2	p	@	c	A	#	**{0}**	N	t	2
3	B	3	n	T	f	I	0	m	s
4	C	4	t	E	k	i	L	l	3
5	M	I	d	H	1	#	D	k	**{!}**
6	1	1	r	O	F	A	n	e	5
7	c	5	x	$	3	@	X	r	9
8	s	0	$	I	0	)	S	f	4
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Table 5.1: First seven entries of the ordering imposed on Σ from the local conditional distribution for each character of the password $x =$ "*PaSsW0rD!*"

the implementation of such methodology has been offered to the reader. Next, in Section 5.3, we describe how such a meter can be shaped via an efficient deep learning framework.

5.3 Meter implementation

In this section, we present a deep-learning-based implementation of the estimation process introduced in Section 5.2.2. Here, we describe the model and its training process. Then, we explain how the trained network can be used as a building block for the proposed password meter.

Model training. From the discussion in Section 5.2.2, our procedure requires the parametrization of an exponentially large number of interactions among random variables. Thus, any tabular approach, such as the one used from Markov Chains or PCFG [114], is *a priori* excluded for any real-world case. To make such a meter feasible, we reformulate the underlying estimation process so that it can be approximated with a neural network. In our approach, we simulate the Markov Random Field described in Section 5.2.2 using a deep convolutional neural network trained to compute $Q(\mathbf{x}_i)$ (Eq. 5.1) for each possible configuration of the structured model. In doing so, we train our network to solve an *masked-language modeling* task defined at character level. Broadly speaking, masked-language modeling is the task of reconstructing missing words from sentences that have been intentionally mangled. **Under the probabilistic perspective, the model is asked to return a probability distribution over all the unobserved elements of x, explicitly measuring the conditional probability of those concerning the observable context**. Therefore, the network has to disentangle and model the semantic relation among all the factors describing the data (e.g., characters in a string) to reconstruct input instances correctly.
Generally, the architecture and the training process used for this task resemble an auto-encoding structure. In the general case, these models are trained to revert self-induced damage carried out on instances of a train-set $\mathbf{X}$. At each training step, an instance $x \in \mathbf{X}$ is artificially mangled with an information-destructive transformation to create a mangled variation $\tilde{x}$. Then, the network, receiving $\tilde{x}$ as input, is optimized to produce an output that most resembles the original x; that is, the network is trained to reconstruct x from $\tilde{x}$.
In our approach, we train a network to infer missing characters in a mangled password. In particular, we iterate over a password leak (i.e., our train-set) by creating mangled passwords and train the network to recover them. The mangling operation is performed

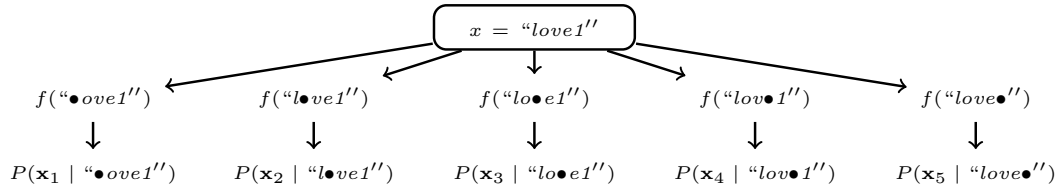


Figure 5.4: Graphical depiction of the complete inference process for the password $x =$ "*love1*". The function f refers to the trained autoencoder and the symbol '•' refers to the deleted character.

by removing a randomly selected character from the string. For example, the train-set entry $x =$ "*iloveyou*" is transformed in $\tilde{x} =$"ilov•you" if the 5'th character is selected for deletion, where the symbol *'•'* represents the *"empty character"*. A compatible proxy-task has been previously used in Chapter 3 to learn a suitable password representation for guessing attacks.

We chose to model our network with a deep *residual* structure arranged to create an autoencoder. The network follows the same general Context Encoder [96] architecture defined in Chapter 3 with some modifications. To create an information bottleneck, the encoder connects with the decoder through a latent space junction obtained through two fully connected layers. We observed that enforcing a latent space, and a prior on that, consistently increases the meter effectiveness. For that reason, we maintained the same regularization proposed in Chapter 3; a maximum mean discrepancy regularization that forces a standard normal distributed latent space. The final loss function of our model is reported in Figure 5.3. In the equation, *Enc* and *Dec* refer to the encoder and decoder network respectively, s is the *softmax* function applied row-wise[2], the distance function d is the cross-entropy, and *mmd* refers to the *maximum mean discrepancy*.

$$\mathbb{E}_{x,\tilde{x}}[d(x,\ s(Dec(Enc(\tilde{x}))))] + \alpha\mathbb{E}_{z\sim N(0,\mathbb{I})}[mmd(z, Enc(\tilde{x}))] \tag{5.3}$$

Henceforth, we refer to the composition of the encoder and the decoder as $f(x) = s(Dec(Enc(x)))$. We train the model on the widely adopted *RockYou* leak [21] considering an 80/20 train-test split. From it, we filter passwords presenting fewer than 5 characters. We train different networks considering different maximum password lengths, namely, 16, 20, and 30. In our experiments, we report results obtained with the model trained on a maximum length equal to 16, as no substantial performance variation has been observed among the different networks. Eventually, we produce three neural nets with different architectures; a large network requiring 36MB of disk space, a medium-size model requiring 18MB, and a smaller version of the second that requires 6.6MB. These models can be potentially further compressed using the same quantization and compression techniques harnessed in [85].[3]

Model inference process. Once the model is trained, we can use it to compute the conditional probability $Q(x_i)$ (Eq. 5.1) for each i and each possible configuration of the probabilistic model. This is done by querying the network f using the same mangling trick performed during the training. The procedure used to compute $Q(x_i)$ for x is summarized in the following steps:

[2]The Decoder outputs ℓ estimations; one for each input character. Therefore, we apply the softmax function separately on each of those to create ℓ probability distributions.

[3]The code, pre-trained models, and other materials related to our work are publicly available at: `https://github.com/pasquini-dario/InterpretablePPSM`.

1. We substitute the i'th character of x with the *empty character* '•', obtaining a mangled password $\tilde{x}$.
2. Then, we feed $\tilde{x}$ to a network that outputs a probability distribution over Σ of the unobserved random variable $\mathbf{x}_i$ i.e., $Q(\mathbf{x}_i)$.
3. Given $Q(\mathbf{x}_i)$, we marginalize out x_i, obtaining the probability: $Q(x_i) = P(\mathbf{x}_i{=}x_i \mid \tilde{x})$.

For instance, if we want to compute the local conditional probability of the character *'e'* in the password $x =$ "*iloveyou*", we first create $\tilde{x} =$"ilov•you" and use it as input for the net, obtaining $Q(\mathbf{x}_5)$, then we marginalize that (i.e., $Q(\mathbf{x}_5{=}\text{'}e\text{'})$) getting the probability $P(\mathbf{x}_5{=}\text{'}e\text{'} \mid \tilde{x})$. From the probabilistic point of view, this process is equivalent to fixing the observable variables in the MRF and querying the model for an estimation of the single unobserved character.

At this point, to cast both the feedback mechanism defined in Section 5.2.1 and the unnormalized joint probability of the string, we have to measure $Q(x_i)$ for each character x_i of the tested password. This is easily achieved by repeating the inference operation described above for each character comprising the input string. A graphical representation of this process is depicted in Figure 5.4. It is important to highlight that the ℓ required inferences are independent, and their evaluation can be performed in parallel (i.e., batch level parallelism), introducing almost negligible overhead over the single inference. Additionally, with the use of a feed-forward network, we avoid the sequential computation that is intrinsic in recurrent networks (e.g., the issue afflicting [85]), and that can be excessive for a reactive client-side implementation. Furthermore, the convolutional structure enables the construction of very deep neural nets with a limited memory footprint.

5.4 Evaluation

In this section, we empirically validate the proposed estimation process as well as its deep learning implementation. First, in Section 5.4.1, we evaluate the capability of the meter of accurately assessing password strength at string-level. Next, in Section 5.4.2, we demonstrate the intrinsic ability of the local conditional probabilities of being sound descriptors of password strength at character-level.

5.4.1 Measuring meter accuracy

In this section, we evaluate the accuracy of the proposed meter at estimating password probabilities. To that purpose, following the adversarial reasoning introduced in Section 2.1.3, we compare the password ordering derived from the meter with the one from the ground-truth password distribution. In doing so, we rely on the guidelines defined in [56] for our evaluation. In particular, given a test-set (i.e., a password leak), we consider a weighted rank correlation coefficient between ground-truth ordering and that derived from the meter. The ground-truth ordering is obtained by sorting the unique entries of the test-set according to the frequency of the password observed in the leak. In the process, we compare our solution with other fully probabilistic meters. A detailed description of the evaluation process follows.

Test-set. For modeling the ground-truth password distribution, we rely on the password leak discovered by 4iQ in the Dark Web[1] on 5th December 2017. It consists of the aggregation of ~ 250 leaks, consisting of 1.4 billion passwords in total. In the cleaning process, we collect passwords with length in the interval $5-16$, obtaining a set of $\sim 4 \cdot 10^8$ unique passwords that we sort in decreasing frequency order. Following the approach of [56], we filter out all the passwords with a frequency lower than 10 from the test-set. Finally, we obtain a test-set composed of 10^7 unique passwords that we refer to as X_{BC}. Given both the large number of entries and the heterogeneity of sources composing it, we consider X_{BC} an accurate description of real-world passwords distribution.

Tested Meters. In the evaluation process, we compare our approach with other probabilistic meters. In particular:

- The Markov model [53] implemented in [16] (the same used in [56]). We investigate different n-grams configurations, namely, 2-grams, 3-grams and 4-grams that we refer to as MM_2, MM_3 and MM_4, respectively. For their training, we employ the same train-set used for our meter.
- The neural approach of Melicher et al. [85]. We use the implementation available at [17] to train the main architecture advocated in [85], i.e., an RNN composed of three LSTM layers of 1000 cells each, and two fully connected layers. The training is carried out on the same train-set used for our meter. We refer to the model as FLA.

Metrics. We follow the guidelines defined by Golla and Dürmuth [56] for evaluating the meters. We use the **weighted** *Spearman* correlation coefficient (ws) to measure the accuracy of the orderings produced by the tested meters, as this has been demonstrated to be the most reliable correlation metric within this context [56]. This metric is defined as

$$\text{ws}(t, m) = \frac{\sum_i^n [w_i (t_i - \bar{t})(m_i - \bar{m})]}{\sqrt{\sum_i^n [w_i (t - \bar{t}_i)^2] \sum_i^n [w_i (m - \bar{m}_i)^2]}} \quad ,$$

where t and m are the sequence of rank assigned to the test-set from the ground-truth distribution and the tested meter, respectively, and where the bar notation (e.g., $\bar{t}$) expresses the weighted mean in consideration of the sequence of weights w. The weights are computed as the normalized inverse of the ground-truth ranks (Eq. 5.4).

$$w = \frac{q}{\sum_i^n q_i} \quad \text{with} \quad q = \frac{1}{t+1} \quad . \tag{5.4}$$

In this metric, the weighting increases the relevance of weak passwords (i.e., the ones with small ranks) in the score computation; that is, the erroneous placing of weak passwords (i.e., asserting a weak password as strong) is highly penalized. Unlike [56], given the large cardinality and diversity of this leak, we directly use the ranking derived from the password frequencies in X_{BC} as ground-truth. Here, passwords with the same frequency value have received the same rank in the computation of the correlation metric.

Results. Table 5.2 reports the measured correlation coefficient for each tested meter. In the table, we also report the required storage as auxiliary metric.
Our meters, even the smallest, achieve higher or comparable score than the most performant Markov Model, i.e., MM_4. On the other hand, our largest model cannot directly

Table 5.2: Rank correlation coefficient computed between X_{BC} and the tested meters.

	MM_2	MM_3	MM_4	**FLA**	**ours** (large)	**ours** (middle)	**ours** (small)
Weighted Spearman ↑	0.154	0.170	0.193	0.217	0.207	0.203	0.199
Required Disk Space ↓	1.1MB	94MB	8.8GB	60MB	36MB	18MB	6.6MB

exceed the accuracy of the state-of-the-art estimator FLA, obtaining only comparable results. However, FLA requires more disk space than ours. Indeed, interestingly, our convolutional implementation permits the creation of remarkably lightweight meters. As a matter of fact, our smallest network shows a comparable result with MM_4 requiring more than a magnitude less disk space.
In conclusion, the results confirm that the probability estimation process defined in Section 5.2.2 is indeed sound and capable of accurately assessing password mass at string-level. The proposed meter shows comparable effectiveness with the state-of-the-art [85], whereas, in the *large* setup, it outperforms standard approaches such as Markov Chains. Nevertheless, we believe that even more accurate estimation can be achieved by investigating deeper architectures and/or by performing hyper-parameters tuning over the model.

5.4.2 Analysis of the relation between local conditional probabilities and password strength

In this section, we test the capability of the proposed meter to correctly model the relation between password structure and password strength. In particular, we investigate the ability of the measured local conditional probabilities of determining the tested passwords' insecure components.
Our evaluation procedure follows three main steps. Starting from a set of weak passwords X:

1. We perform a guessing attack on X in order to estimate the guess-number of each entry of the set.

2. For each password $x \in X$, we substitute n characters of x according to the estimated local conditional probabilities (i.e., we substitute the characters with highest $Q(\mathbf{x}_i)$), producing a perturbed password $\tilde{x}$.

3. We repeat the guessing attack on the set of perturbed passwords and measure the variation in the attributed guess-numbers.

Hereafter, we provide a detailed description of the evaluation procedure.

Passwords sets. The evaluation is carried out considering a set of weak passwords. In particular, we consider the first 10^4 most frequent passwords of the X_{BC} set.

Password perturbations. In the evaluation, we consider three types of password perturbation:
(1) The first acts as a baseline and consists of the substitution of random positioned characters in the passwords with randomly selected symbols. Such a general strategy is used in [104] and [55] to improve the user's password at composition time. The perturbation

Table 5.3: Strength improvement induced by different perturbations. The last two rows of the table report the AGI ratio between the two meter-based approaches and the baseline.

	$n=1$	$n=2$	$n=3$
Baseline (PNP)	0.022	0.351	0.549
Semi-Meter (PNP)	0.036	0.501	0.674
Fully-Meter (PNP)	0.066	0.755	0.884
Baseline **(AGI)**	$3.0 \cdot 10^{10}$	$3.6 \cdot 10^{11}$	$5.6 \cdot 10^{11}$
Semi-Meter **(AGI)**	$4.6 \cdot 10^{10}$	$5.1 \cdot 10^{11}$	$6.8 \cdot 10^{11}$
Fully-Meter **(AGI)**	$8.2 \cdot 10^{10}$	$7.7 \cdot 10^{11}$	$8.9 \cdot 10^{11}$
Semi-Meter / Baseline **(AGI)**	**1.530**	**1.413**	**1.222**
Fully-Meter / Baseline **(AGI)**	**2.768**	**2.110**	**1.588**

is applied by randomly selecting n characters from x and substituting them with symbols sampled from a predefined character pool. In our simulations, the pool consists of the 25 most frequent symbols in X_{BC} (i.e., mainly lowercase letters and digits). Forcing this character-pool aims at preventing the tested perturbation procedures to create artificially complex passwords such as strings containing extremely uncommon *unicode* symbols. We refer to this perturbation procedure as **Baseline**.

(2) The second perturbation partially leverages the local conditional probabilities induced by our meter. Given a password x, we compute the conditional probability $Q(x_i)$ for each character in the string. Then, we select and substitute the character with maximum probability, i.e., $\arg\max_{x_i} Q(x_i)$. The symbol we use in the substitution is randomly selected from the same pool used for the baseline perturbation (i.e., top-25 frequent symbols). When n is greater than one, the procedure is repeated sequentially using the perturbed password obtained from the previous iteration as input for the next step. We refer to this procedure as **Semi-Meter**.

(3) The third perturbation extends the second one by exploiting the local conditional distributions. Here, as in the Semi-Meter-based, we substitute the character in x with the highest probability. However, rather than choosing a substitute symbol in the pool at random, we select that according to the distribution $Q(\mathbf{x}_i)$, where i is the position of the character to be substituted. In particular, we choose the symbol the minimize $Q(\mathbf{x}_i)$, i.e., $\arg\min_{s\in\Sigma'} Q(\mathbf{x}_i{=}s)$, where Σ' is the allowed pool of symbols. We refer to this method as **Fully-Meter**.

Guessing Attack. We evaluate password strength using the *min-auto* strategy advocated in [108]. Here, guessing attacks are simultaneously performed with different guessing tools, and the guess-number of a password is considered the minimum among the attributed guess-numbers. In performing such attacks, we rely on the combination of three widely adopted solutions, namely, HashCat [7], PCFG [114, 19] and the Markov chain approach proposed in [53, 18]. For tools requiring a training phase, i.e., OMEN and PCFG, we use the same train-set used for our model (i.e., 80% of *RockYou*). Similarly, for HashCat, we use the same data set as input dictionary[4] and *generated2* as rules set. During the guesses generation, we maintain the default settings of each implementation. We limit each tool to produce 10^{10} guesses. The total size of the generated guesses is $\sim$ 3TB.

[4]In this case, passwords are unique and sorted in decreasing frequency.

Metrics. In the evaluation, we are interested in measuring the increment of password strength caused by an applied perturbation. We estimate that value by considering the Average Guess-number Increment (henceforth, referred to as AGI); that is, the average delta between the guess-number of the original password and the guess-number of the perturbed password:

$$\text{AGI}(X) = \frac{1}{|X|}\sum_{i=0}^{|X|}[g(\tilde{x}^i) - g(x^i)]$$

where g is the guess-number, and $\tilde{x}^i$ refers to the perturbed version of the i'th password in the test set. During the computation of the guess-numbers, it is possible that we fail to guess a password. In such a case, we attribute an artificial guess-number equals to 10^{12} to the un-guessed passwords. Additionally, we consider the average number of un-guessed passwords as an ancillary metrics; we refer to it with the name of Percentage Non-Guessed Passwords (PNP) and compute it as:

$$\text{PNP(X)} = \frac{1}{|X|}|\{x^i \mid g(x^i) \neq \perp \ \wedge \ g(\tilde{x}^i) = \perp\}|,$$

where $g(x) = \perp$ when x is not guessed during the guessing attack.

Results. We perform the tests over three values of n (i.e., the number of perturbed characters), namely, 1, 2, and 3. Results are summarized in Table 5.3. The AGI caused by the two meter-based solutions is always greater than that produced by random perturbations. On average, that is twice more effective with respect to the Fully-Meter baseline and about 35% greater for the Semi-Meter. The largest relative benefit is observable when $n = 1$, i.e., a single character is modified. Focusing on the Fully-Meter approach, indeed, the guidance of the local conditional probabilities permits a guess-number increment 2.7 times bigger than the one caused by a random substitution in the string. This advantage drops to ~ 1.5 when $n = 3$, since, after two perturbations, passwords tend to be already out of the dense zone of the distribution. Indeed, at $n = 3$ about 88% of the passwords perturbed with the Fully-Meter approach cannot be guessed during the guessing attack (i.e., PNP). This value is only $\sim 55\%$ for the baseline. More interestingly, the results tell us that substituting two ($n = 2$) characters following the guide of the local conditional probabilities causes a guess-number increment greater than the one obtained from three ($n = 3$) random perturbations. As a matter of fact, the AGI for the Fully-Meter perturbation is $\sim 7.6 \cdot 10^{11}$ for $n = 2$ whereas is $\sim 5.7 \cdot 10^{11}$ for the baseline when $n = 3$.
In the end, these results confirm that the local conditional distributions are indeed sound descriptors of password security at the structural level.

Limitations. Since the goal of our evaluation was mainly to validate the soundness of the proposed estimation process, we did not perform user studies and we did not evaluate human-related factors such as password memorability although we recognize their importance.

5.5 Conclusion

In this Chapter, we showed that it is possible to construct interpretable probabilistic password meters by fundamentally rethinking the underlying password mass estimation.

We presented an non-autoregressive probabilistic interpretation of the password generative process that can be used to build precise and sound password feedback mechanisms. Moreover, we demonstrated that such an estimation process could be instantiated via a lightweight deep learning implementation.

Part III

Final Remarks and Future Directions

Chapter 6

Conclusions and future perspectives

In our thesis work, we harnessed deep learning techniques to cast novel directions in the extensively studied, and yet presently active, field of password security. As a result, we improved the core techniques that are pivotal in ensuring password security and, ultimately, users' security and privacy:

Sounder strength estimates via Dynamic Attackers: By abstracting the underlying password model (e.g., generative models or dictionary attacks), the proposed dynamic techniques and supporting mechanisms vastly demonstrated their effectiveness. Even without additional information on the attacked-set, those techniques are able to guess more passwords within the same/smaller number of guesses, making explicit the sub-optimal nature of guessing attacks that model static adversaries. Furthermore, those techniques demonstrated their value in increasing the reliability and robustness of sensitive attacks such as dictionary attacks that are pivotal in accurately describing real-world attackers.

More importantly, dynamic attacks have demonstrated capable of guessing those passwords that are unique to the attacked password set (e.g., Table 3.5). Given their arbitrary distance from the general password distribution, such passwords can be soundly guessed only by leveraging additional sources of information over the attacked password space. The proposed dynamic attacks distill this necessary knowledge directly from an unsupervised interaction with the attacked-set, allowing the guessing attack to automatically focus on unique modalities of the target password distribution that would either be ignored or heavily under-represented otherwise.

The dynamic techniques demonstrate the existence of passwords that are considered secure by state-of-the-art approaches but are inherently weak once the attacker leverages a more realistic guessing strategy. This further result suggests that the security of a password is not an intrinsic property of the string, but it is strongly dependent on other environmental factors that cannot be *a priori* stated. In particular, this insight adds an additional dimension over the definition of password strength: the security of a password is also a function of the passwords that occur in the same environment, **even if those are completely unknown to the adversary before the attack**. In turn, this brings us to question the soundness of existing estimation techniques that model only static attackers as well as the validity of their strength estimates.

In conclusion, given our results, we argue that it is critical to account for advanced as well as realistic attackers, that employ a dynamic approach, in accurately measuring password strength. In this thesis, we introduced novel techniques that permit to shape

such adversaries and enable more robust and sound password strength estimates.

Interpretability in Password Strength Meters: Enabling interpretable mechanisms in password strength meters is a major challenge in password security. Preventing users from choosing weak passwords and soundly guiding them in employing secure ones is critical to ensure users' security. However, existing approaches seek interpretability in heuristic constructions that are inherently limited [104, 115].

In the thesis, we demonstrated that it is possible to learn a general and unsupervised feedback mechanism from raw data by relying on deep learning techniques. The meter we described in Chapter 5 is the first one enabling a rigorous feedback signal that has a clear probabilistic interpretation and it is free from human bias. We validated the proposed approach by showing that it achieves comparable accuracy with autoregressive solutions, while introducing its unique feedback mechanism that generalizes any heuristic construction.

More broadly, we empirically demonstrated that the local conditional distributions of a structured probabilistic model can be used to cast interpretable mechanisms, but under the condition of abandoning the autoregressive paradigm that dominated state-of-the-art meters until now [85, 104]. This general intuition may open new directions in the password strength meters research, enabling the creation of further interpretable approaches.

6.1 Future directions

Given the final remarks, in this Section we sketch future directions and extensions of the works reported in the thesis.

6.1.1 Modeling dynamic attackers with Normalizing Flows

To implement the dynamic password guessing framework introduced in Chapter 3, we used a GAN [58] generator. This implicit probabilistic model [49] cannot assign probability to the generated passwords, preventing us to sort guesses in optimal order and using the generator as a probabilistic password meter (see Section 2.1.3).

These shortcomings of the current implementation can be overcome by relaying on a different class of deep generative models i.e., **normalizing flows** [50, 70]. These models enable density estimation in implicit probabilistic models and come with exact inference, allowing the exact inversion of data points into latent points.

Relying on this class of generative models, we can enhance the proposed solutions and cast additional tools aimed at improving the security of passwords. More prominently, with these, we can produce a **probabilistic password meter that accounts and models dynamic attackers**.

6.1.2 Graph theory meets Password Security

In Chapter 4, we introduced the dynamic dictionary augmentation procedure. As discussed in Section 4.3.1, this simple technique enables the construction of massive knowledge graphs that describe the relationship among passwords guessed during a dynamic attack e.g., Figure 6.1. These graphs may be used as tools to grasp novel insights on

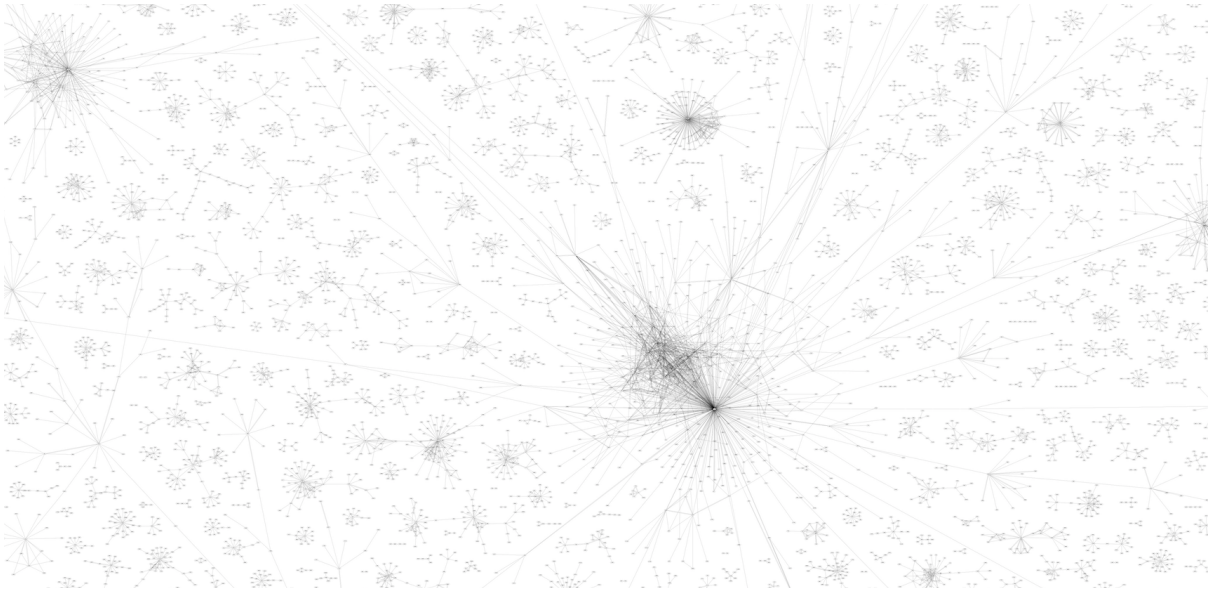


Figure 6.1: Example of graph created with the dynamic dictionary attack.

passwords strength and to explain causality properties among guessed passwords in a dynamic context. Eventually, we may be able to derive proactive mechanisms aimed at reducing the threat of dynamic attackers (i.e., real-world attackers) by studying the properties of these graphs.

6.1.3 Transformers and Interpretable Password Meters

In Chapter 5, we constructed interpretable password meters over a convolutional autoencoder. In that model, kernels are convoluted over the password to model the relation properties among characters. However, given the inductive bias of convolutional layers, those give particular importance to local properties rather than long-range relationship through the password. While local properties are important in human-chosen passwords, one can perform a less biased estimation by constructing the meter over a different neural architecture.

In this direction, **transformers** [109] are naturally suited for this task and offer additional compelling features. The **self-attention mechanism** employed from transformers networks provides an explicit way to model the relationship among characters. Furthermore, the attention weights assigned by the attention mechanisms enable further interpretation on the conditional relation properties of characters. These can explicate the dependence properties between every pair of characters and help to systematically determine the *a priori* influence of a character over the whole password. In addition, a recent breakthrough in the field [42] can drastically improve transformers' computational efficiency and make them suited for client-side operability.

Furthermore, while we implemented our meter at the character level, more powerful and explanatory models may be achieved by a token-based approach, where words or sub-words are used to segment passwords. Within this approach, the user is given a more coarse-grained feedback that may be easier to understand and get insight from.

Possible additional applications We highlight how the validity of our approach has been demonstrated in this thesis work only within the context of password security. However, it is apparent that guessing passwords is just an instance of a more general class of problems that can be defined as *reconstruction of finite sequences of symbols belonging to a predefined set*. We believe that our novel approach may be applied, with suitable adjustments, to other instances of that class of problems (e.g., genomic data).

Part IV

Supplemental material

Appendix A

A.1 Inducing peculiar password latent organizations via inductive bias

Given the absence of precise external bias, the generative models used to learn the latent password representation is free to choose arbitrary spatial arrangements among passwords. In the general case, our generators learn the latent representation that best supports the extremely general generative task imposed during the training. However, this may not be optimal. For instance, the latent spaces learned by our technique tend to keep passwords with similar length very close to each other. The reason is that the length of a password is modeled as one of the core explanatory factors [29] by the latent representation. As a result, passwords with different lengths are distributed far from each other, which is good for DPG but undesirable in other cases. For instance, it may be better to generate passwords that share specific substrings, but that do not have comparable length.

Luckily, this type of specialization is possible within our frameworks. Our deep learning approach is highly versatile, and password organizations that present a peculiar feature can be obtained through the injection of inductive bias during the learning process.

Focusing on the AE (Section 3.1.2), we can indeed induce structure preferences in the latent space organization through regularizations during training. For instance, we can easily reduce the length-based clustering phenomenon described above by acting on the character deletion process used in Section 3.1.2. In the normal case, we learn a latent representation by training the auto-encoder at reconstructing artificially mangled passwords, where each character in the input string is removed with a certain probability. Differently, we can delete a group of k continuous characters given a randomly chosen starting position i. For instance, with $k = 5$, a password *"jimmy1991"* can become *"jimm*****"* with $i = 4$; otherwise *"*****991"* with $i = 0$. Intuitively, the generator collects in the same location passwords that share common substrings, regardless of their length. For instance, given the mangled password *"jimmy*****"*, the generator should be able to recover the passwords *"jimmy"*, *"jimmyjimmy"* and *"jimmy123"*, eventually forcing their latent representations to be close to each other.

As an example, we compare passwords sampled from CWAE trained with different approaches, namely, using the character deletion approach discussed in Section 3.1.2 (here, referred to as *Simple*) and using the group deletion approach discussed above (referred to as *Mask*). Table A.1 reports password sampled around the pivot *"iloveyou1"* for the two CWAEs. Compared to *Simple*, passwords sampled from the *Mask* model tend to have heterogeneous lengths which are arbitrarily different from the one of the pivot.

Simple	Mask
iloveyou13	iloveyou1234
iloveyou12	iloveyou14
iloveYou1	iloveyou12ao
iLoveyou1	iloveyou1222
iloveyou*	iloveyou17a
Iloveyou1	iloveyou12arham
iloheyou1	iloveyou14om
ilOveyou1	iloveyou123o
iloveyou11a	iloveyou1444
iloveyou1a	iloveyou12a4mom1

Table A.1: Passwords sampled around the pivot *"iloveyou1"* for two CWAEs trained with different regularization. The same value of σ is used for both models.

A.2 Learning the inverse mapping for the GAN model

To fully exploit the properties offered by the learned latent representation of passwords, we need a way to explore the latent space efficiently. Therefore, our primary interest is to understand the relation between the observed data (i.e., passwords) and their respective latent representations; in particular, their position within the latent space. A direct way to model this relation is to learn the inverse of the generator function $G^{-1} : \mathbf{X} \rightarrow \mathbf{Z}$. GANs, by default, do not need to learn those functions because that requirement is bypassed by the adversarial training approach. To do so, framework variations [51, 52] or additional training phases [77] are required.

To avoid any source of instability in the original training procedure, we opt to learn the inverse mapping only after the training of the generator is complete. This is accomplished by training a third **encoder** network E that has an identical architecture as the *critic*, except for the size of the output layer. The network is trained to simultaneously map both the real (i.e., data coming from the train-set) and generated (i.e., data coming from G) data to the latent space. Specifically, the loss function of E is mainly defined as the sum of the two cyclic reconstruction errors over the data space. This is presented in the following:

$$\begin{aligned} L_0 &= \mathbb{E}_z[d(G(z), G(E(G_t(z))))], \\ L_1 &= \mathbb{E}_x[d(x, G(E(x)))]. \end{aligned} \tag{A.1}$$

In Eq. (A.1), the function d is the cross-entropy whereas x and z are sampled from the train-set and the prior latent distribution, respectively. The variable t in L_0 refers to the temperature of the final *softmax* layer of the generator. In Eq. (A.1), we do not specify temperature on a generator notation when it is assumed that it does not change during the training. The combination of these two reconstruction errors aims at forcing the encoder to learn a general function capable of inverting both the true and generated data correctly. As discussed in Section 3.1.1, the discrepancy between the representation of the true and generated data (i.e., discrete and continuous data) is potentially harmful to the training process. To deal with this issue, we anneal the temperature t in loss term L_0 during the training. We do that to collapse slowly the continuous representations of the generated data (i.e., the output of the generator) towards the same discrete representation of the real data (i.e., coming from the dataset). Next, an additional loss term, shown in Eq. A.2, is added forcing the encoder to map the data space in a dense zone of the latent space (dense with respect to the prior latent distribution).

$$L_2 = \mathbb{E}_z[d(z, E(G(z)))]. \tag{A.2}$$

Our final loss function for E is reported in Eq. A.3. During the encoder training, we use the same train-set that we used to train the generator, but we consider only the unique passwords in this case.

$$L_E = \alpha L_0 + \beta L_1 + \gamma L_2. \tag{A.3}$$

The information about the hyper-parameters we used is listed in Table A.2.

Hyper-parameter	**Value**
α	0.2
β	0.2
γ	0.6
Batch size	64
Learning rate	0.001
Optimizer	*Adam*
Temperature decay step	250000
Temperature limit	0.1
Temperature scheduler	*polynomial*
Train iteration	$3 \cdot 10^5$

Table A.2: Hyper-parameters used to train our encoder network

A.3 On the impact of hyper-parameters on DPG

In this section, we briefly consider the impact of the two hyper-parameters of DPG over the quality of the attack.

Figure A.1 depicts a comparison among the static attack, a DPG with $\alpha = 15\%$, and a DPG with $\alpha = 0\%$ (i.e., no hot-start). These results confirm that the absence of hot-start indeed affects and eventually degrades the performance of DPG.

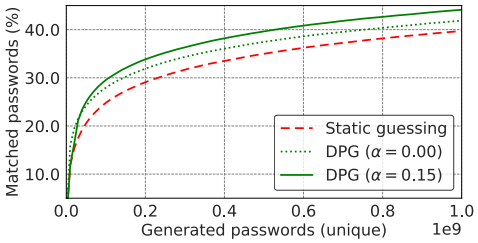


Figure A.1: The impact of α on the performance of DPG for *phpbb* test-set

Figure A.2 depicts the effect of different values of σ on the performance of DPG. Smaller values of α yields better overall results. This outcome suggests that it is not necessary to sample too far from the dense zones imposed by Z_i, and rather a focused exploration of those zones is beneficial. This observation is perfectly coherent with the discussed locality property, giving further support to the speculated ability of the latent space of capturing and translating general features of an entire password distribution in geometric relations.

A.4 Supplementary tables & figures for DPG and CPG

Here, we present supplementary data related to our work. Table A.3 lists the samples of password templates and their respective matching passwords. Table A.4 extends

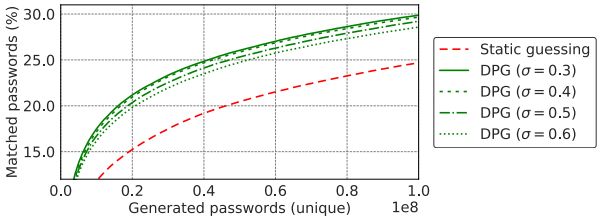


Figure A.2: The impact of σ on the performance of DPG for *phpbb* test-set

Table 3.5 for the attack on the *LinkedIn* set. We report the guess-numbers for John the Ripper, Hashcat, Markov Model, and PCFG. These value have been obtained via the CMU-PGS [23, 108]. Note that PGS sets up its models with a different ground-truth; our train-set is just a subset of the one used from PGS.

In the table, the underscore symbol '_' indicates that the password model failed to match the password. The column '*DPG G.*' reports the guess-number of the dynamic attack. The passwords are sorted using the same criteria used in Table 3.5. We report the top 100 entries.

T_{common}			T_{uncommon}			T_{rare}			$T_{\text{super-rare}}$		
a*e*on*	**ri***19****	***ol*nd*****	**Bi**o**1***	****n1**0*0**	*****dy*78***	**a*6*4*0****	****j99*9****	***n****0!!**	**k*****kbn**	****sb*9*8***	***YR**R*U***
Cameron4$	rizal1982	Colinda23	BigCorp11	Mon171050	sandy@786	a06142001	sbj991980	Qny1960!!	ktyzhekbn	mosby9382	PYR@GR@UP
cameron64	rissi1909	yolanda#1	BigFoot13	Len112080	sandy6789	a26042004	tej991991	ando140!!	kgn5*5kbn	elsb1968!	MYRATROUT
CabeZone1	rimpy1984	Noland405	Bishon111	ben101010	goody1785	ab6643014	Lwj990922	vny@@00!!	ktrnhjkbn	lksbs9080	
madelon13	riana1976	noland339	Bigfoot1#	chn102030	cindy2785	a76645090	nhj990920	lnb7280!!	kbnkbnkbn	ldsbc9886	
Camerone3	rinni1970	rolando13	Bingo2011	Jan172010	maddy2789	a1644104a	naj999999	anaid60!!			
cameronq2	richu1989	roland589	Biddoma12	van102030	buddy8780	a26547054	Slj999999	@ngel20!!			
makedon24	rinks1978	Rolando85	Bigboy117	jan152000	brady1785	a06042007	jjj999999	QnA2010!!			
Kameron76	rinat1978	roland006	Biofoto10	ten142000	maddy@786	a8674600Z	msj991987	Annie20!!			
cameron46	risco1969	RolandD50	Biologo12	jan142000	sandy7780	a76042074	99j99a99k	Annie10!!			
Nakedone1	riken1970	Jolanda48	BioComp10	l4n1n402	Toodys781	am68400en	dej991976	inusa20!!			

Table A.3: Samples of password templates and respective matching passwords.

Guessed P.	DPG G.	JTR G.	Hashcat G.	Markov G.	PCFG G.
o2linkedln	$3.4 \cdot 10^9$	–	–	–	–
w2linkedln	$3.1 \cdot 10^9$	–	–	–	–
ydlinkedln	$3.6 \cdot 10^9$	–	–	–	–
linked6in6	$4.3 \cdot 10^9$	–	–	–	–
j*linkedln	$4.3 \cdot 10^9$	–	–	–	–
linked!in.	$4.8 \cdot 10^9$	–	–	–	$2.1 \cdot 10^{14}$
wslinked1n	$4.4 \cdot 10^9$	–	–	–	–
linkedgcin	$2.1 \cdot 10^9$	–	–	–	–
linked6in2	$5.6 \cdot 10^9$	–	–	–	$1.7 \cdot 10^{14}$
lslinkedln	$4.5 \cdot 10^9$	–	–	–	–
wtlinkedln	$4.5 \cdot 10^9$	–	–	–	–
9auiirji	$5.5 \cdot 10^9$	–	–	–	$7.6 \cdot 10^{13}$
g2linkedln	$3.4 \cdot 10^9$	–	–	–	–
cslinkedln	$4.4 \cdot 10^9$	–	–	–	–
ymlinkedln	$5.2 \cdot 10^9$	–	–	–	–
linked4in6	$4.4 \cdot 10^9$	–	–	–	$2.2 \cdot 10^{14}$
fvlinkedln	$4.7 \cdot 10^9$	–	–	–	–
jslinkedln	$3.7 \cdot 10^9$	–	–	–	–
jzlinkedln	$5.1 \cdot 10^9$	–	–	–	–
sslinkedln	$4.4 \cdot 10^9$	–	–	–	–
grlinkedln	$4.7 \cdot 10^9$	–	–	–	–
linkedm1x1	$2.5 \cdot 10^9$	–	–	–	–
svlinked1n	$5.1 \cdot 10^9$	–	–	–	–
m1linkedln	$3.8 \cdot 10^9$	–	–	–	–
linkedi9in	$2.7 \cdot 10^9$	–	$6.7 \cdot 10^{11}$	–	–
mnlinkedln	$3.7 \cdot 10^9$	–	–	–	–
etlinkedln	$4.9 \cdot 10^9$	–	–	–	–
forc3link	$2.1 \cdot 10^9$	$8.6 \cdot 10^{11}$	$1.0 \cdot 10^9$	–	$4.3 \cdot 10^{11}$
5.linkedin	$4.7 \cdot 10^9$	–	–	–	$8.8 \cdot 10^{11}$
link4rfxa	$4.8 \cdot 10^9$	–	–	–	$4.2 \cdot 10^{11}$
g0linked1n	$2.5 \cdot 10^9$	–	–	–	$1.3 \cdot 10^{14}$
linkedm1m1	$2.9 \cdot 10^9$	–	$6.7 \cdot 10^{11}$	–	–
56linkedln	$4.6 \cdot 10^9$	–	–	–	–
Rbnoi076	$2.0 \cdot 10^9$	–	–	–	$4.2 \cdot 10^{13}$
linkedtgin	$1.9 \cdot 10^9$	–	–	–	–
linked8in4	$5.6 \cdot 10^9$	–	–	–	$2.0 \cdot 10^{14}$
linked!in1	$4.4 \cdot 10^9$	–	–	–	$4.1 \cdot 10^{13}$
imlindedin	$4.9 \cdot 10^9$	–	–	–	–
linkedkbin	$2.9 \cdot 10^9$	–	–	–	–
linked9in6	$4.2 \cdot 10^9$	–	–	–	–
htlinkedln	$4.8 \cdot 10^9$	–	–	–	–
golinkedln	$5.2 \cdot 10^9$	–	–	–	–
ozlinkedln	$5.1 \cdot 10^9$	–	–	–	–
o.linkedin	$4.3 \cdot 10^9$	–	$6.7 \cdot 10^{11}$	–	$6.3 \cdot 10^{11}$
linkedwcz	$2.9 \cdot 10^9$	–	–	–	$2.9 \cdot 10^{13}$
linked_iin	$5.0 \cdot 10^9$	–	–	–	$4.6 \cdot 10^{13}$
linkedrcin	$1.6 \cdot 10^9$	–	–	–	–
42linkedln	$4.5 \cdot 10^9$	–	–	–	–
linkedcmw4	$3.1 \cdot 10^9$	–	–	–	–
mmlinkedln	$3.7 \cdot 10^9$	–	–	–	–
2xrilidi	$5.1 \cdot 10^9$	–	–	–	$1.8 \cdot 10^{13}$
dslinkedln	$4.3 \cdot 10^9$	–	–	–	–
linkedtdin	$2.1 \cdot 10^9$	–	–	–	–
linked1.in	$3.2 \cdot 10^9$	–	–	–	$2.0 \cdot 10^{14}$
linked4in2	$4.4 \cdot 10^9$	–	–	–	$8.6 \cdot 10^{13}$
linked4in4	$4.1 \cdot 10^9$	–	$6.7 \cdot 10^{11}$	–	$1.0 \cdot 10^{14}$
linked.4in	$3.2 \cdot 10^9$	–	–	–	$2.2 \cdot 10^{14}$
pdlinkedln	$4.2 \cdot 10^9$	–	–	–	–
oklinkedln	$5.2 \cdot 10^9$	–	–	–	–
Or2nge47	$1.7 \cdot 10^9$	–	$1.1 \cdot 10^{11}$	–	–
z1linkedin	$4.3 \cdot 10^9$	–	$6.7 \cdot 10^{11}$	–	$8.8 \cdot 10^{10}$
linkednxin	$1.4 \cdot 10^9$	–	–	–	–
53linkedln	$4.9 \cdot 10^9$	–	–	–	–
linkedctq	$2.8 \cdot 10^9$	–	–	–	$2.7 \cdot 10^{13}$
odlinkedln	$3.5 \cdot 10^9$	–	–	–	–
omlinkedln	$3.9 \cdot 10^9$	–	–	–	–
eu293634r	$2.3 \cdot 10^9$	–	–	–	$5.4 \cdot 10^{13}$
hklinked1n	$5.0 \cdot 10^9$	–	–	–	–
linkedfsin	$2.0 \cdot 10^9$	–	$6.7 \cdot 10^{11}$	–	–
lf00garl	$4.3 \cdot 10^9$	–	$8.8 \cdot 10^{11}$	–	$2.6 \cdot 10^{12}$
y9linkedin	$5.3 \cdot 10^9$	–	–	–	$9.2 \cdot 10^{11}$
linked87ln	$2.5 \cdot 10^9$	–	–	–	$1.1 \cdot 10^{14}$
linked544y	$2.8 \cdot 10^9$	–	–	–	$8.9 \cdot 10^{13}$
xbCA0N	$1.9 \cdot 10^9$	–	–	–	–
linktebow	$9.1 \cdot 10^8$	–	–	–	$2.8 \cdot 10^{13}$
y2linkedin	$4.5 \cdot 10^9$	–	–	–	$4.1 \cdot 10^{11}$
linkedmiam	$3.7 \cdot 10^9$	–	–	–	–
73linkedln	$4.1 \cdot 10^9$	–	–	–	–
alasEN00	$4.4 \cdot 10^8$	–	–	–	$9.3 \cdot 10^{11}$
h9linkedin	$4.5 \cdot 10^9$	–	–	–	$7.9 \cdot 10^{11}$
linkedkbl	$1.9 \cdot 10^9$	–	$6.7 \cdot 10^{11}$	–	$2.6 \cdot 10^{13}$
T8wtas00	$5.7 \cdot 10^8$	–	–	–	$5.3 \cdot 10^{13}$
linkedw3s	$4.0 \cdot 10^9$	–	–	–	–
44linkedln	$4.7 \cdot 10^9$	–	–	–	–
unpceddi	$5.1 \cdot 10^9$	–	–	–	$2.4 \cdot 10^{11}$
linkedwge	$1.1 \cdot 10^9$	–	$6.7 \cdot 10^{11}$	–	$2.5 \cdot 10^{13}$
linked39in	$2.3 \cdot 10^9$	–	–	–	$1.1 \cdot 10^{14}$
linked99ln	$2.3 \cdot 10^9$	–	–	–	$8.7 \cdot 10^{13}$
linke14din	$3.1 \cdot 10^9$	–	–	–	$1.8 \cdot 10^{13}$
gxlinkedln	$3.3 \cdot 10^9$	–	–	–	–
linkedkpin	$1.2 \cdot 10^9$	–	–	–	–
gonulelif	$2.5 \cdot 10^9$	–	–	–	$2.5 \cdot 10^{13}$
linkedcsun	$2.1 \cdot 10^9$	–	–	–	–
lclinkedln	$4.4 \cdot 10^9$	–	–	–	–
9.linkedin	$4.7 \cdot 10^9$	–	–	–	$1.1 \cdot 10^{12}$
grswbon3	$2.5 \cdot 10^9$	–	–	–	$2.7 \cdot 10^{12}$
snlinkedln	$4.8 \cdot 10^9$	–	–	–	–

Table A.4: Guess-numbers of the top peculiar password guessed from DPG for *LinkedIn* leak.

Appendix B

B.1 Password leaks

In our study, we made use of different password leaks both for training and testing purposes. They are listed in Table B.1 along with additional information.

Name	Unique Passwords	Brief Description
LinkedIn[14]	$\sim 6 \cdot 10^7$	An employment-oriented online service.
youku [5]	$\sim 4 \cdot 10^7$	Chinese video hosting service.
MyHeritage[24]	$\sim 3 \cdot 10^7$	Online genealogy platform.
zooks[3]	$\sim 2 \cdot 10^7$	Online dating service available in 80 countries.
RockYou[21]	$\sim 10^7$	Gaming platform.
animoto[22]	$\sim 8 \cdot 10^6$	A cloud-based video creation service.
zomato[25]	$\sim 5 \cdot 10^6$	Indian, food delivery application. About 40% of the password are random tokens of six alphanumeric characters.
phpBB	$\sim 10^5$	Software website.

Table B.1: Used Password leaks sorted by size.

B.2 Details on the deep learning framework of the compatibly function

Algorithm 6: Residual Block: `residualBlock(·)`:

Data: input tensor: x_{in}

$x = \texttt{batchNormalization}(x_{in})$;
$x = \texttt{ReLU}(x)$;
$x = \texttt{1D-Convolution}(x, f, k)$;
$x = \texttt{batchNormalization}(x)$;
$x = \texttt{ReLU}(x)$;
$x = \texttt{1D-Convolution}(x, f, k)$;
return $x_{in} + 0.3 \cdot x$

Algorithm 7: Architecture:

Data: input tensor: x_{in}, rules-set R

$x = \texttt{charactersEmbedding}(x_{in}, 128)$;

$x = \texttt{1D-Convolution}(x, f, k)$;

for 0 ***to*** d **do**

  $x = \texttt{residualBlock(x)}$

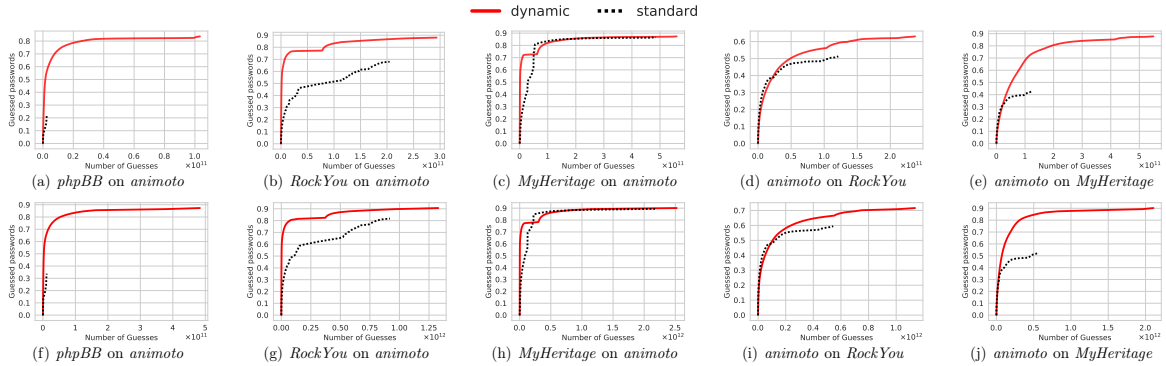
$bneck = \lceil \frac{f}{b} \rceil$;

$x = \texttt{1D-Convolution}(x, bneck, k)$;

$x = \texttt{flattern}(x)$;

$logits = \texttt{dense}(x, |R|)$;

return $logits$



(a) *phpBB* on *animoto* (b) *RockYou* on *animoto* (c) *MyHeritage* on *animoto* (d) *animoto* on *RockYou* (e) *animoto* on *MyHeritage*

(f) *phpBB* on *animoto* (g) *RockYou* on *animoto* (h) *MyHeritage* on *animoto* (i) *animoto* on *RockYou* (j) *animoto* on *MyHeritage*

Figure B.1: Performance comparison between static and dynamic attack for five different setups of dictionary/attacked-set. The results are reported for the rules-sets *generated* (first row) and *generated2* (second row) in non-adaptive mode.

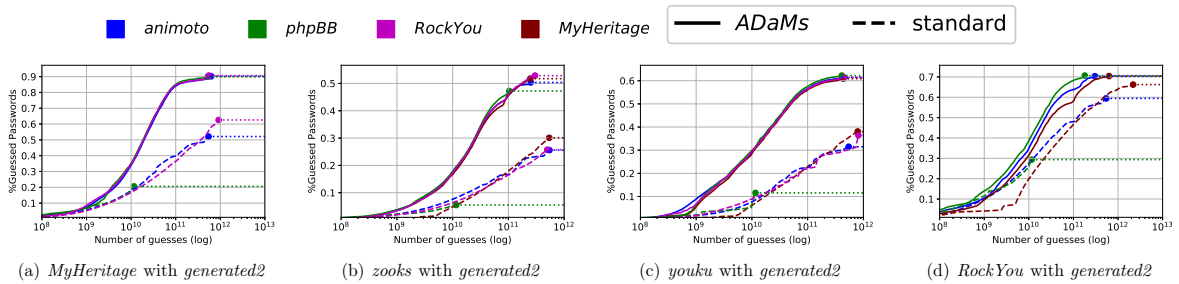
This Appendix details the architecture used to implement the neural approximations of the compatibility functions presented in Section 4.2.2. It can be defined using five parameters, namely:

- **Depth (d):** The number of residual blocks composing the network. Each residual block includes two 1D-convolutional layers, supported by normalization layers and activation i.e., Algorithm 6.
- **Number of filters (f):** The number of filters for each convolutional layer in the network.
- **Kernel size (k):** Size of the kernel used in every convolutional layer in the network.
- **Final Bottleneck (b):** Reduction of the number of filters before the final dense layer.

The final architecture is described in Algorithm 7. Our biggest models are realizations of the parameters: d=15, f=512, k=5. We use b=2 for *PasswordPro* and *generated*, b=3 for *generated2* instead.

B.3 Impact of the Dynamic budget on *AdaMs*

We briefly illustrate the impact of the dynamic budget (i.e., Section 4.3.2) on the performance of *AdaMs*. As previously discussed, the dynamic budget has always a positive or neutral effect. Figure B.3 reports an example for the attacked-set *youku*. In the figure,



(a) *MyHeritage* with *generated2* (b) *zooks* with *generated2* (c) *youku* with *generated2* (d) *RockYou* with *generated2*

Figure B.2: Each plot reports the number of guesses and the percentage of matched passwords for different dictionaries against four attacked-sets. We use four dictionaries, each identified by a color line. Continuous lines show *AdaMs* attack, whereas dashed lines refer to standard mangling rules attacks.

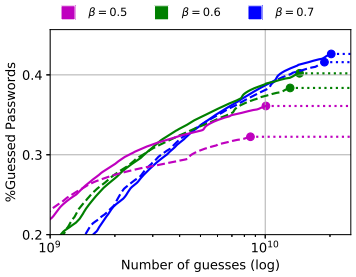


Figure B.3: Effectiveness of the dynamic-budget within *AdaMs* for different value of β. Continuous lines present *AdaMs*, whereas dashed lines are *AdaMs* ablated of the dynamic-budget

continuous lines refer to the complete *AdaMs* attack, whereas dashed lines report the results for *AdaMs* without dynamic budget for the same configuration. We report the results for three values of β.
As shown in the example, the dynamic budget is particularly effective when low β is used. In these cases, the dynamic logic helps better organize the small total budget of the attack, resulting in better global performance. The gain decreases when bigger budgets are adopted.

B.4 Additional results for *AdaMs*and dynamic dictionary

This appendix collects additional results.

Figure B.1 compare dynamic and static mangling rules attack for the rules-sets *generated* and *generated2*.

Figure B.2, instead, compares *AdaMs* and standard mangling rules attack for various dictionaries and attacked-sets using *generated2* as rules-set.

B.5 Benchmarks of the *AdaMs* attack

In this Appendix, we analyze the computational cost of generating guesses with *AdaMs*. Primarily, we test the overhead with respect to standard mangling rules (i.e., *Hashcat*

Table B.2: Number of guesses per second compute single core/GPU on a NVIDIA DGX-2 machine.

AdaMs **generated2**	*AdaMs* **generated**	*AdaMs* **PasswordPro**	**Hashcat** CPU legacy
726182 g/s	709439 g/s	644444 g/s	928647 g/s

CPU legacy).

For the comparison, we produce 10^9 strings and compute the number of guesses generated per second (i.e., g/s). **In the process, we include the time of checking for the guesses in the set of the attacked passwords** (the same methodology is used for each tool and may not be computationally optimal). Note that we do not perform any hash function computation in the process. We repeat the test 5 times using *RockYou* as dictionary and *animoto* as attacked-set, whereas we repeat for the rules-sets: *PasswordPro*, *generated* and *generated2*. Table B.2 averages the time for each tool. The result for the standard mangling rules is reported as average over the three rules-sets.
On average, *AdaMs* are just 25% slower than standard mangling rules. Considering that the Adaptive mangling rules can reduce the number of guesses up to an order of magnitude, this overhead becomes negligible in practice. Moreover, this discrepancy easily fades out when slow hash functions, such as [66, 98, 97], are considered.

B.6 Implementation of *AdaMs*

We rely on the *CPU legacy* version of *Hashcat*[1] to implement *AdaMs* attacks. Our prototype uses the CPU version as it is easier to modify its workflow, although the *Hashcat* GPU engine can trivially support our approach.[2]
In the code, we modify the main loop of *Hashcat*, where it scans over dictionary words and then iterates on all rules. We read a batch of words from the dictionary, we give them as input to the neural network, and then, for each word w in the batch, we apply only the rules whose values of α_R are greater than $(1 - \beta)$. We check all these guesses and, those who match are added on top of the remaining words in the dictionary, i.e., they will be part of the next batch of words. The same batching approach is used for the dynamic budget. Here, budget increments and normalization per rule are performed conjointly after every batch to further reduce computational overhead. In the implementation, we use batch-size equals to 4096 dictionary words.

[1] `https://github.com/hashcat/hashcat-legacy`

[2] The GPU engine is also more suited as it would naturally support the computation of the neural network on GPU, removing the CPU/GPU communication overhead

Bibliography

[1] "1.4 Billion Clear Text Credentials Discovered in a Single Database". https://tinyurl.com/t8jp5h7.

[2] "arstechnica.com: Anatomy of a hack: How crackers ransack passwords like "qeadzcwrsfxv1331".

[3] "arstechnica.com: Dating site Zoosk resets some user accounts following password dump". https://tinyurl.com/y3r2xob5.

[4] "Automatic mangling rules generation". http://passwords12.at.ifi.uio.no/Simon_Marechal-manglingrules_Passwords12.pdf.

[5] "Chinese Video Service Giant Youku Hacked; 100M Accounts Sold on Dark Web". https://tinyurl.com/yb78uxnh.

[6] "Cracking Passwords 101". https://tinyurl.com/y268xahe.

[7] "hashcat". https://tinyurl.com/y636jsz9.

[8] "Have I been pwned? service". https://haveibeenpwned.com.

[9] "Hotmail Password Leak". https://tinyurl.com/yyr2je4m.

[10] "I have the HashCat so I make the rules". https://hashcat.net/events/p14-vegas/I%20have%20the%20%23cat%20i%20make%20the%20rules_YC.pdf.

[11] "InsidePro-PasswordsPro Rules". https://tinyurl.com/vd9jzaz.

[12] "John the Ripper". https://tinyurl.com/j91l.

[13] "Leak Youku". https://tinyurl.com/y9f2xez6.

[14] "LinkedIn Password Leak". https://tinyurl.com/yxf7f5gv.

[15] "MySpace Password Leak". https://tinyurl.com/y433aaah.

[16] "NEMO Markov Model GitHub". https://github.com/RUB-SysSec/NEMO.

[17] "Neural Network Cracking GitHub". https://github.com/cupslab/neural_network_cracking.

[18] "OMEN GitHub". https://github.com/RUB-SysSec/OMEN.

[19] "PCFG GitHub". https://github.com/lakiw/pcfg_cracker.

[20] "phpbb Password Leak". https://tinyurl.com/yxonf7um.

[21] "RockYou Password Leak". https://www.computerworld.com/article/2522045/rockyou-hack-exposes-names–passwords-of-30m-accounts.html.

[22] "techcrunch.com: Animoto hack exposes personal information, location data". https://tinyurl.com/ybhc9uaz.

[23] "The Carnegie Mellon University Password Research Group's Password Guessability Service". https://tinyurl.com/y9362h6z.

[24] "The Verge: MyHeritage breach leaks millions of account details". https://tinyurl.com/y7w6wsrf.

[25] "Zomato hacked: Security breach results in 17 million user data stolen". https://tinyurl.com/y8xec7sr.

[26] "Zomato Password Leak". https://tinyurl.com/ya3sthdp.

[27] Kamran Ali, Alex X Liu, Wei Wang, and Muhammad Shahzad. Keystroke Recognition Using WiFi Signals. In *ACM MobiCom*, pages 90–102, 2015.

[28] Davide Balzarotti, Marco Cova, and Giovanni Vigna. Clearshot: Eavesdropping on Keyboard Input from Video. In *IEEE S&P*, pages 170–183, 2008.

[29] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation Learning: A Review and New Perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013.

[30] Massimo Bernaschi, Pasqua D'Ambra, and Dario Pasquini. AMG based on compatible weighted matching for GPUs. *Parallel Computing*, 92:102599, 2020.

[31] Massimo Bernaschi, Pasqua D'Ambra, and Dario Pasquini. Bootcmatchg: An adaptive algebraic multigrid linear solver for gpus. *Software Impacts*, page 100041, 2020.

[32] J. Blocki, B. Harsha, and S. Zhou. On the economics of offline password cracking. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 853–871, 2018.

[33] J. Bonneau. The science of guessing: Analyzing an anonymized corpus of 70 million passwords. In *2012 IEEE Symposium on Security and Privacy*, pages 538–552, 2012.

[34] J. Bonneau, C. Herley, P. C. v. Oorschot, and F. Stajano. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *2012 IEEE Symposium on Security and Privacy*, pages 553–567, 2012.

[35] Joseph Bonneau. *Guessing human-chosen secrets*. PhD thesis, 06 2012.

[36] Joseph Bonneau, Cormac Herley, Paul C. van Oorschot, and Frank Stajano. Passwords and the evolution of imperfect authentication. *Commun. ACM*, 58(7):78–87, June 2015.

[37] Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pages 10–21, Berlin, Germany, August 2016. Association for Computational Linguistics.

[38] S. Boztas. Entropies, guessing and cryptography, 1999.

[39] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large Scale GAN Training for High Fidelity Natural Image Synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

[40] Claude Castelluccia, Markus Dürmuth, and Daniele Perito. Adaptive Password-Strength Meters from Markov Models. In *NDSS*, 2012.

[41] R. Chatterjee, J. Bonneau, A. Juels, and T. Ristenpart. Cracking-resistant password vaults using natural language encoders. In *2015 IEEE Symposium on Security and Privacy*, pages 481–498, 2015.

[42] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. Rethinking attention with performers, 2020.

[43] Luke St. Clair, Lisa Johansen, William Enck, Matthew Pirretti, Patrick Traynor, Patrick McDaniel, and Trent Jaeger. Password exhaustion: Predicting the end of password usefulness. In *Information Systems Security*, pages 37–55, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[44] Anupam Das, Joseph Bonneau, Matthew Caesar, Nikita Borisov, and XiaoFeng Wang. The tangled web of password reuse. In *NDSS*, volume 14, pages 23–26, 2014.

[45] Xavier de Carné de Carnavalet and Mohammad Mannan. From very weak to very strong: Analyzing password-strength meters. In *NDSS*, 01 2014.

[46] M. Dell' Amico, P. Michiardi, and Y. Roudier. Password strength: An empirical analysis. In *2010 Proceedings IEEE INFOCOM*, pages 1–9, March 2010.

[47] Matteo Dell'Amico and Maurizio Filippone. Monte Carlo Strength Evaluation: Fast and Reliable Password Checking. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, page 158–169, New York, NY, USA, 2015. Association for Computing Machinery.

[48] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[49] Peter J Diggle and Richard J Gratton. Monte Carlo Methods of Inference for Implicit Statistical Models. *Journal of the Royal Statistical Society: Series B (Methodological)*, 46(2):193–212, 1984.

[50] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real NVP. *CoRR*, abs/1605.08803, 2016.

[51] Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. Adversarial Feature Learning. *arXiv preprint arXiv:1605.09782*, 2016.

[52] Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Olivier Mastropietro, Alex Lamb, Martin Arjovsky, and Aaron Courville. Adversarially Learned Inference. *arXiv preprint arXiv:1606.00704*, 2016.

[53] Markus Dürmuth, Fabian Angelstorf, Claude Castelluccia, Daniele Perito, and Abdelberi Chaabane. OMEN: Faster Password Guessing using an Ordered Markov Enumerator. In *ESSoS*, pages 119–132, 2015.

[54] Sascha Fahl, Marian Harbach, Yasemin Acar, and Matthew Smith. On the ecological validity of a password study. In *Proceedings of the Ninth Symposium on Usable Privacy and Security*, SOUPS '13, New York, NY, USA, 2013. Association for Computing Machinery.

[55] Alain Forget, Sonia Chiasson, P. C. van Oorschot, and Robert Biddle. Improving Text Passwords through Persuasion. In *Proceedings of the 4th Symposium on Usable Privacy and Security*, SOUPS '08, page 1–12, New York, NY, USA, 2008. Association for Computing Machinery.

[56] Maximilian Golla and Markus Dürmuth. On the Accuracy of Password Strength Meters. In *ACM CCS*, pages 1567–1582, 2018.

[57] Ian Goodfellow. NIPS 2016 Tutorial: Generative Adversarial Networks. *arXiv preprint arXiv:1701.00160*, 2016.

[58] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. In *NIPS*, pages 2672–2680, 2014.

[59] Palash Goyal and Emilio Ferrara. Graph Embedding Techniques, Applications, and Performance: A Survey. *Elsevier Knowledge-Based Systems*, 151:78–94, 2018.

[60] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved Training of Wasserstein GANs. In *NIPS*, pages 5767–5777, 2017.

[61] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *CVPR*, pages 770–778, 2016.

[62] Briland Hitaj, Paolo Gasti, Giuseppe Ateniese, and Fernando Perez-Cruz. PassGAN: A Deep Learning Approach for Password Guessing. In *ACNS*, pages 217–237, 2019.

[63] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR.

[64] Blake Ives, Kenneth R Walsh, and Helmut Schneider. The domino effect of password reuse. *Communications of the ACM*, 47(4):75–78, 2004.

[65] Markus Jakobsson and Mayank Dhiman. The benefits of understanding passwords. In *Proceedings of the 7th USENIX Conference on Hot Topics in Security*, HotSec'12, page 10, USA, 2012. USENIX Association.

[66] B. Kaliski. "Pkcs# 5: Password-based cryptography specification version 2.0". `https://tools.ietf.org/html/rfc2898`, 2000.

[67] P. G. Kelley, S. Komanduri, M. L. Mazurek, R. Shay, T. Vidas, L. Bauer, N. Christin, L. F. Cranor, and J. Lopez. Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms. In *2012 IEEE Symposium on Security and Privacy*, pages 523–537, 2012.

[68] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[69] Diederik P Kingma and Max Welling. Auto-encoding Variational Bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[70] Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31, pages 10215–10224. Curran Associates, Inc., 2018.

[71] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009.

[72] Saranga Komanduri, Richard Shay, Lorrie Faith Cranor, Cormac Herley, and Stuart Schechter. Telepathwords: Preventing Weak Passwords by Reading Users' Minds. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 591–606, San Diego, CA, August 2014. USENIX Association.

[73] Saranga Komanduri, Richard Shay, Patrick Gage Kelley, Michelle L. Mazurek, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, and Serge Egelman. Of passwords and people: Measuring the effect of password-composition policies. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, New York, NY, USA, 2011. ACM.

[74] Yang Li and Tao Yang. Word Embedding for Understanding Natural Language: A Survey. In *Springer Guide to Big Data Applications*, pages 83–104. 2018.

[75] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollar. Focal loss for dense object detection. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[76] Enze Liu, Amanda Nakanishi, Maximilian Golla, David Cash, and Blase Ur. Reasoning Analytically About Password-Cracking Software. In *IEEE Symposium on Security and Privacy*, SP '19, pages 1272–1289, San Francisco, California, USA, May 2019. IEEE.

[77] Junyu Luo, Yong Xu, Chenwei Tang, and Jiancheng Lv. Learning Inverse Mapping by Autoencoder based Generative Adversarial Nets. In *International Conference on Neural Information Processing*, pages 207–216. Springer, 2017.

[78] Jerry Ma, Weining Yang, Min Luo, and Ninghui Li. A Study Of Probabilistic Password Models. In *IEEE S&P*, pages 689–704, 2014.

[79] Jianzhu Ma, Jian Peng, Sheng Wang, and Jinbo Xu. Estimating the Partition Function of Graphical Models Using Langevin Importance Sampling . In *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics*, volume 31 of *Proceedings of Machine Learning Research*, pages 433–441, Scottsdale, Arizona, USA, 29 Apr–01 May 2013. PMLR.

[80] Laurens van der Maaten and Geoffrey Hinton. Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[81] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. Adversarial Auto-Encoders. *arXiv preprint arXiv:1511.05644*, 2015.

[82] Philip Marquardt, Arunabh Verma, Henry Carter, and Patrick Traynor. (sp)iPhone: Decoding Vibrations From Nearby Keyboards Using Mobile Phone Accelerometers. In *ACM CCS*, pages 551–562, 2011.

[83] J. L. Massey. Guessing and entropy. In *Proceedings of 1994 IEEE International Symposium on Information Theory*, pages 204–, June 1994.

[84] 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 and Communications Security*, CCS '13, page 173–186, New York, NY, USA, 2013. Association for Computing Machinery.

[85] 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 *USENIX Security Symposium*, pages 175–191, 2016. GitHub Repo: `https://tinyurl.com/y9o7jdd8`.

[86] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. Red Hook, NY, USA, 2013. Curran Associates Inc.

[87] Shakir Mohamed and Balaji Lakshminarayanan. Learning in Implicit Generative Models. *arXiv preprint arXiv:1610.03483*, 2016.

[88] Robert Morris and Ken Thompson. Password Security: A Case History. *Communications of the ACM*, 22(11):594–597, 1979.

[89] 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*, CCS '05, page 364–372, New York, NY, USA, 2005. Association for Computing Machinery.

[90] 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.

[91] Dario Pasquini, Giuseppe Ateniese, and Massimo Bernaschi. Interpretable probabilistic password strength meters via deep learning. In *25th European Symposium on Research in Computer Security (ESORICS)*, September 2020.

[92] Dario Pasquini, Giuseppe Ateniese, and Massimo Bernaschi. Unleashing the tiger: Inference attacks on split learning, 2020.

[93] Dario Pasquini, Marco Cianfriglia, Giuseppe Ateniese, and Massimo Bernaschi. Reducing bias in modeling real-world password strength via deep learning and dynamic dictionaries. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, August 2021.

[94] Dario Pasquini, Ankit Gangwal, Giuseppe Ateniese, Massimo Bernaschi, and Mauro Conti. Improving Password Guessing via Representation Learning. In *42th IEEE Symposium on Security and Privacy (Oakland)*, May 2021.

[95] Dario Pasquini, Marco Mingione, and Massimo Bernaschi. Adversarial out-domain examples for generative models. In *2019 IEEE European Symposium on Security and Privacy Workshops, EuroS&P Workshops 2019, Stockholm, Sweden, June 17-19, 2019*, pages 272–280. IEEE, 2019.

[96] Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A Efros. Context Encoders: Feature Learning by Inpainting. In *IEEE CVPR*, pages 2536–2544, 2016.

[97] Colin Percival. Stronger key derivation via sequential memory-hard functions. 01 2009.

[98] Niels Provos and David Mazières. A future-adaptive password scheme. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '99, page 32, USA, 1999. USENIX Association.

[99] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *arXiv preprint arXiv:1511.06434*, 2015.

[100] Stuart Schechter, Cormac Herley, and Michael Mitzenmacher. Popularity is everything: A new approach to protecting passwords from statistical-guessing attacks. In *Proceedings of the 5th USENIX Conference on Hot Topics in Security*, HotSec'10, page 1–8, USA, 2010. USENIX Association.

[101] Donn Seeley. Password cracking: A game of wits. *Commun. ACM*, 32(6):700–703, June 1989.

[102] Masashi Sugiyama, Matthias Krauledat, and Klaus-Robert MÃžller. Covariate Shift Adaptation by Importance Weighted Cross Validation. *Journal of Machine Learning Research*, 8(May):985–1005, 2007.

[103] Ilya Tolstikhin, Olivier Bousquet, Sylvain Gelly, and Bernhard Schoelkopf. Wasserstein Auto-Encoders. *arXiv preprint arXiv:1711.01558*, 2017.

[104] 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 *CHI '17*, 2017.

[105] Blase Ur, Jonathan Bees, Sean M. Segreti, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. Do users' perceptions of password security match reality? In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, page 3748–3760, New York, NY, USA, 2016. Association for Computing Machinery.

[106] 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.

[107] Blase Ur, Fumiko Noma, Jonathan Bees, Sean M Segreti, Richard Shay, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. I Added '!'at the End to Make It Secure: Observing Password Creation in the Lab. In *SOUPS*, pages 123–140, 2015.

[108] 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 *USENIX Security Symposium*, pages 463–481, 2015.

[109] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc., 2017.

[110] Martin Vuagnoux and Sylvain Pasini. Compromising Electromagnetic Emanations of Wired and Wireless Keyboards. In *USENIX Security Symposium*, pages 1–16, 2009.

[111] D. Wang, D. He, H. Cheng, and P. Wang. fuzzyPSM: A New Password Strength Meter Using Fuzzy Probabilistic Context-Free Grammars. In *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 595–606, June 2016.

[112] 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.

[113] Matt Weir, Sudhir Aggarwal, Michael Collins, and Henry Stern. Testing Metrics for Password Creation Policies by Attacking Large Sets of Revealed Passwords. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, page 162–175, New York, NY, USA, 2010. Association for Computing Machinery.

[114] Matt Weir, Sudhir Aggarwal, Breno De Medeiros, and Bill Glodek. Password Cracking using Probabilistic Context-free Grammars. In *IEEE S&P*, pages 391–405, 2009.

[115] Daniel Lowe Wheeler. zxcvbn: Low-Budget Password Strength Estimation. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 157–173, Austin, TX, August 2016. USENIX Association.

[116] Tom White. Sampling Generative Networks. *arXiv preprint arXiv:1609.04468*, 2016.

[117] Junyuan Xie, Linli Xu, and Enhong Chen. Image denoising and inpainting with deep neural networks. In *Advances in Neural Information Processing Systems 25*, pages 341–349. Curran Associates, Inc., 2012.

[118] Roman V Yampolskiy. Analyzing User Password Selection Behavior for Reduction of Password Space. In *ICCST*, pages 109–115, 2006.

[119] Chih-Kuan Yeh, Wei-Chieh Wu, Wei-Jen Ko, and Yu-Chiang Frank Wang. Learning deep latent spaces for multi-label classification. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, page 2838–2844. AAAI Press, 2017.