Neural Cryptanalysis of Classical Ciphers ⋆

Riccardo Focardi and Flaminia L. Luccio

Università Ca' Foscari, Venezia
{focardi,luccio}@unive.it

Abstract. Cryptanalysis identifies weaknesses of ciphers and investigates methods to exploit them in order to compute the plaintext and/or the secret cipher key. Exploitation is nontrivial and, in many cases, weaknesses have been shown to be effective only on reduced versions of the ciphers. In this paper we apply artificial neural networks to automatically "assist" cryptanalysts into exploiting cipher weaknesses. The networks are trained by providing data in a form that points out the weakness together with the encryption key, until the network is able to generalize and predict the key (or evaluate its likelihood) for any possible ciphertext. We illustrate the effectiveness of the approach through simple classical ciphers, by providing the first ciphertext-only attack on substitution ciphers based on neural networks.

Keywords: Artificial neural networks; cryptography; cryptanalysis.

1 Introduction

Cryptography aims at decorrelating as much as possible plaintexts and keys from ciphertexts, so that computing the key from the plaintext and the ciphertext is infeasible, even when the attacker can choose the plaintexts to encrypt or ciphertexts to decrypt (the so-called *chosen-plaintext* and *chosen-ciphertext* scenarios). It is thus very hard to imagine that any fully automated technique will possibly break complex ciphers. On the other side, cryptanalysis looks for ways to break (part of) ciphers, in a white-box fashion: for example, by exploiting weaknesses of internal operations it might become possible to compute part of the key bits reducing the cost of brute-forcing attacks. However, exploitation is nontrivial and, in many cases, weaknesses have been shown to be effective only on reduced versions of the ciphers (see, e.g., [7,8]).

In this paper, we apply artificial neural networks to automatically "assist" cryptanalysts into exploiting cipher weaknesses. Our threat model assumes that the attacker knows or finds new weaknesses of a cipher and trains neural networks to exploit these weaknesses automatically. The networks are trained by providing data in a form that points out the weakness together with the encryption key, until the network is able to generalize and predict the key (or estimate its likelihood) for any possible ciphertext. In other words, neural networks become a tool for the cryptanalyst to automate and

⋆ Work partially supported by CINI Cybersecurity National Laboratory within the project FilieraSicura funded by CISCO Systems Inc. and Leonardo SpA.

optimize her attacks: the cryptanalyst points out the weakness and the neural network learns how to exploit it.

We illustrate the effectiveness of the approach through simple classical ciphers such as, shift (Caesar), Vigenère and substitution. We are aware that these ciphers are considered extremely weak, however they are well suited to illustrate the approach. Cryptanalysis of these ciphers can be done by only inspecting the statistical structure of the ciphertext.[1] Thus, we train the neural networks by providing simple statistical features (such as frequencies of single letters, digrams and trigrams) together with the adopted key until the network generalizes the attack and becomes able to predict a key (or compute its likelihood) from any given ciphertext. Interestingly, we provide the first ciphertext-only attack on substitution ciphers based on neural networks.

Our results are preliminary but we believe that the proposed approach is a promising step towards a more efficient and partially automated cryptanalysis of modern ciphers. In particular, our approach might be adopted to cryptanalyze existing broken ciphers, e.g., in the automotive industry [21,22], or to automate (and hopefully optimize) known attacks in reduced versions of state-of-the-art, secure ciphers.

Contributions. We contribute as follows: (i) we show a very efficient and fast way to cryptanalyze shift ciphers using neural networks. The trained neural network is able to recover the key by providing as input the relative frequencies of the ciphertext letters; (ii) we show how the neural network for shift ciphers can be applied to break Vigenère poly-alphabetic ciphers. The neural network is applied to subtexts until it provides an accurate result and outputs the cipher's key; (iii) we provide a very accurate and efficient strategy to cyptanalyse substitution ciphers. The neural network takes as input the 3-grams (three letters) frequencies and provides a measure of the goodness of the plaintext, with respect to English. This allows for searching the key-space efficiently until the correct plaintext is found. To the best of our knowledge this is the first attack on substitution ciphers based on neural networks.

Related work Recently, lots of work has been done in order to apply different *computational intelligence* techniques to the field of cryptography (see, e.g., [6] for an overview). In this paper we only concentrate on machine learning techniques and more specifically on neural networks. The idea of relating the fields of cryptography and machine learning goes back to 1993 when Rivest in [18], firstly discussed how, at that time, each field was contributing with techniques and insights to the other. After that, much work has been done in this direction, in particular for what concerns the use of machine learning techniques to help cryptographers to develop new good ciphers. In this paper we are interested in solving a different problem, i.e., the application of neural networks to automatically "assist" cryptanalysts into exploiting cipher weaknesses. In this regard, the literature is more limited and we discuss it below.

For what concerns attacks to classical ciphers, [4] proposes the use of neural networks to run *known-plaintext* attacks to Vigenère ciphers which, in fact, can be trivially achieved by computing the difference between ciphertexts and plaintexts. Our attacks, instead,

[1] Notice that, when a pair plaintext-ciphertext is known computing the key of these ciphers is trivial, so we will consider *ciphertext-only* attack scenarios.

are *ciphertext-only*. In [15], the author shows how recurrent neural networks may be used to learn decryption algorithms for the Vigenère, Autokey, and Enigma ciphers. This approach, as the author states, is data inefficient as it requires at least a million training examples, i.e., pairs of plaintext and ciphertext to learn a cipher. From a practical point of view this translates to a few days of calculations on a k40 GPU to get $96 - 97\%$ accuracy. Our networks are trained in a matter of seconds, which is what you would expect for the cryptanalysis of simple ciphers. Another limitation is the key-phrases size, which is limited to 1-6 characters. A recent work by Gomez et al. [13] introduces CipherGAN, a tool based on Generative Adversarial Networks (GANs). The authors experimentally show that CipherGAN can decrypt shift and Vigenère ciphers with an accuracy of 98.7% for shift ciphers and of 75.7% for Vigenère ciphers, which turns out to be far less accurate (and much more computational intensive) than ours. Finally, Liu et al. in [16] propose a novel cost function for unsupervised learning and a stochastic primal-dual gradient (SPDG) algorithm, and they show how they can be applied in real-world scenarios, in particular to break the Caesar cipher, but they do not consider more complex ciphers such as Vigenère and substitution.

Regarding attacks to other ciphers, in [5] the authors propose an application of neural networks to recover in the last round part of the key an hypothetical cryptographic algorithm based on the Feistel architecture, while in [9] the authors present an application of a neural cryptanalysis approach to the Simplified DES (S-DES), and they were able to obtain the correct values for three key bits. These results are quite ad-hoc and their practical impact and how they relate to known cryptanalytic attacks is unclear.

Neural networks have been used for the different problem of cipher classification: in [19] the authors present a neural network that by analyzing some features of the cipher is able to classify it as Playfair, Vigenère or Hill cipher. Another example of application of machine learning techniques to cryptography is presented in [17] where the authors propose to apply these techniques in the context of side channel attacks, which we do not consider here.

Paper Outline. The paper is organized as follows. In Section 2 we briefly recall neural network techniques and we give some background on classical ciphers. In Section 3 we describe our proposed approach starting with the shift cipher, then Vigenère, and finally the substitution cipher. We conclude in Section 4 with some final remarks.

2 Background

In this section we now shortly recall what an artificial neural network is, and how are classical ciphers defined.

Artificial neural networks. An artificial neural network is a network composed of different internal layers, called *hidden layers*, which may vary both in their number and in the number of internal nodes called *neurons*. The first layer, typically depicted on the left is called *input layer*, where the data flow in. It is connected to the second layer and so on up to the *output layer* that produces the model prediction.

Each neuron has a set of input values $z_1, z_2, \ldots$ which are real numbers between 0 and 1, and has also a set of weights $w_1, w_2, \ldots$ associated to each input, and an

overall bias b. Initially, all the weights are set randomly, and then the algorithm changes during its execution. Each neuron has also an *activation function*, that applies non-linear transformations to the inputs and weights. The functions we will be using for our tests will be two very commonly used ones, i.e., the sigmoid and the rectifier functions. Given a set of input values $z_1, z_2, \ldots$, a set of weights $w_1, w_2, \ldots$, and a bias b we define $z = \sum_j w_j z_j + b$. Formally, the *sigmoid function* σ is the non-linear function $\sigma(z) = \frac{1}{1+exp(-z)}$, while the *rectifier function* f is an always positive function $f(z) = \max\{0, z\}$. The interested reader can refer to [14] for more detail.

Classical ciphers. We now shortly recall the classical ciphers that we will analyse in the next sections. We consider a set of plaintexts P, a set of ciphertexts C and a set of keys K. Since in the present work we analyse plaintexts written in English language, we assume that all operations are in $\mathbb{Z}_{26}$.

The shift cipher or Caesar cipher, is a mono-alphabetic cipher that maps a letter of the plaintext each time to the same letter of the ciphertext. Formally:

Definition 1. *[20] Given a plaintext $x \in P$, a ciphertext $y \in C$, and a key $k \in K$, where $P = C = K = \mathbb{Z}_{26}$, a* shift cipher *(or* Caesar cipher*) is defined by the encryption function $E_k(x) = x+k \bmod 26$, and by the decryption function $D_k(y) = y-k \bmod 26$.*

Note that, the number of different keys is limited to 26 (all values between 0 and 25), so a trivial brute-force attack is feasible.

The Vigenère cipher, is a poly-alphabetic cipher that maps a letter of the alphabet into a set of different letters. The cipher works on blocks of m letters with a key of length m. Formally:

Definition 2. *[20] Given a plaintext $x \in P$, a ciphertext $y \in C$, and a key $k \in K$, where $P = C = K = \mathbb{Z}_{26}^m$, and where $\mathbb{Z}_{26}^m$ is $\mathbb{Z}_{26} \times \mathbb{Z}_{26} \times \ldots \times \mathbb{Z}_{26}$, m times, for a key $K = (k_1, \ldots, k_m)$, the* Vigenère cipher *is defined by the encryption function $E_{k_1,\ldots,k_m}(x_1, \ldots, x_m) = (x_1 + k_1, \ldots, x_m + k_m) \bmod 26$, and by the decryption function $D_{k_1,\ldots,k_m}(y_1, \ldots, y_m) = (y_1 - k_1, \ldots, y_m - k_m) \bmod 26$.*

The number of possible keys is 26^m, i.e., all the possible sequences of letters of length m. For m big enough this prevents brute-force attacks.

Finally, a substitution cipher is a mono-alphabetic cipher that maps the letters of the alphabet into a generic permutation. Formally:

Definition 3. *[20] Given a plaintext $x \in P$, a ciphertext $y \in C$, and a key $\rho \in K$, where $P = C = \mathbb{Z}_{26}$, and $K = \{\rho \mid \rho$ is a permutation of $0, 1, \ldots, 25\}$, a* substitution cipher *is defined by the encryption function $E_\rho(x) = \rho(x)$, and by the decryption function $D_\rho(y) = \rho^{-1}(x)$, where ρ^{-1} is the inverse permutation of ρ.*

Note that, in this case the number of all possible keys is 26!, i.e., the number of all permutations of 26 numbers, which is approximately $4 \times 10^{26} > 2^{88}$. Even if this is less than the recommended size for cryptographic keys, i.e., 2^{128}, it is still very hard to run a brute-force attack even using powerful parallel computers.

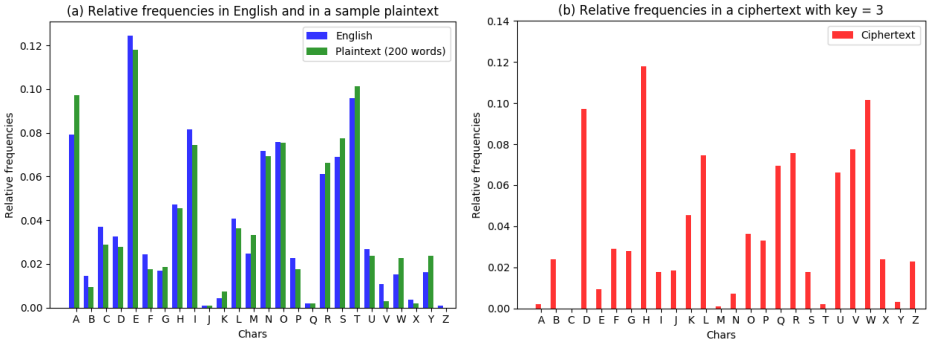

Fig. 1: Caesar cipher circularly shifts the frequency histogram *key* positions to the right.

3 Cryptanalysis of classic ciphers with neural networks

In this section we use neural networks to automate attacks on the classic ciphers of Section 2, by exploiting known statistical weaknesses. We start with Caesar (shift) cipher and we show how to train a neural network to recover the key by simply providing the relative frequencies of ciphertext letters (cf. Section 3.1); then, we show how to apply the Caesar neural network to Vigenère subtexts so to efficiently recover the cipher key (cf. Section 3.2); finally, we train a neural network with 3-grams frequencies in order to compute the likelihood of a substitution key and we use the neural network to efficiently search the key space of substitution ciphers (cf. Section 3.3).

Experimental setup. We consider a corpus of 1 million words extracted from the British National Corpus [1]. Neural networks are implemented using the Keras library [2] with TensorFlow back-end [3]. We run all our tests on a MacBook Pro with 2 GHz Intel Core i7 and 16 GB of RAM, without any GPU or multi-core optimization. We empirically found that simple networks with just one hidden layer were sufficient and could be trained very efficiently with 100% accuracy. This is consistent with the fact that the considered ciphers are simple. More sophisticated networks might be necessary to perform attacks on more complex ciphers.

3.1 Predicting the Caesar cipher key

We start with the toy example of the Caesar (shift) cipher. As we have previously pointed out, this cipher only has 26 keys and can be easily brute-forced by inspecting all the possible 26 decryptions. However, to automate the analysis it would be desirable to detect the correct decryption without human intervention. Because of the very small key-space, in this specific example a check of all possible decryptions against a dictionary would suffice but we investigate a more direct, cryptanalytic approach to directly compute the key, based on the particular encryption function. In fact, the shift cipher is a particularly weak form of mono-alphabetic cipher that preserves the relative order of the frequencies of the alphabet characters.

If we measure the frequencies of the characters in a ciphertext, the obtained histogram is a circular shift of the plaintext frequency histogram. Along the brute-force idea above, it is enough to shift the histogram until it "fits" the one characterizing the plaintext language, e.g., English (that can be easily computed from long enough English texts) to find the decryption key. Figure 1(a) shows the relative frequencies computed from the British National Corpus [1] and from a sample plaintext of 200 words. We notice that, even if there are differences, the shape of the histogram is preserved. Figure 1(b) illustrates the relative frequencies after encryption with key 3. It is clearly visible how the shape of the histogram is (circularly) shifted to the right of three positions.

We now show that brute-forcing the key is not necessary as we can directly train a neural network to recognize the relative shift for the frequency histogram. In particular, we approach the problem as follows:

1. we take a big enough dataset of English plaintexts encrypted under random keys;
2. we compute the frequencies of single letters of the ciphertexts;
3. we train a neural network by providing the frequencies as input and the corresponding key as output;
4. we test the network on an independent dataset.

Thus, intuitively, the cryptanalyst only points out a weakness (the frequency of single letters in this case) and leaves to the neural network the job of correlating this to the cipher key. The fact that the key can be computed directly from the histogram is interesting as it shows the power of neural networks for the cryptanalysis of the cipher: we just provide the histograms and keys of the training cases and this makes it possible to "recognize" the key (on different ciphers) without trying the actual shifts.

Experimental results. We use a rather standard neural network for classification, with an output layer of 26 neurons, one for each possible key. The only hidden layer is also composed of 26 neurons, as we empirically checked that we did not require more layers and thus more complexity. The activation function is sigmoid σ (cf. Section 2). We trained the network using 5097 ciphertexts of 100 words obtained by encrypting different plaintexts under random keys. We tested the model against the same number of ciphertexts obtained by different plaintexts encrypted under random keys with a 100% accuracy, confirming that the model did not overfit the training data. Training takes about 30 seconds and cracking a ciphertext is almost instantaneous. The model is extremely accurate up to about 30 words.

Figure 2 shows the worst-case classification value for the selected keys that we got out of 5097 cracking different ciphertexts of lengths 30, 50, 100 and 200, finding respectively key 1 (Figure 2a), 19 (Figure 2b), 17 (Figure 2c) and 16 (Figure 2d). The attacker picks the key with highest score. For length 30 the best candidate is not so far from the second best, while for bigger texts the prediction is neat. This is due to the fact that relative frequencies of letters in short sentences deviate more from the average ones in English.

3.2 Breaking Vigenère cipher through the Caesar classifier

We now consider the Vigenère cipher that is one of the first poly-alphabetic ciphers ever proposed. The idea is exactly to contrast the problem of preserving the letter frequencies

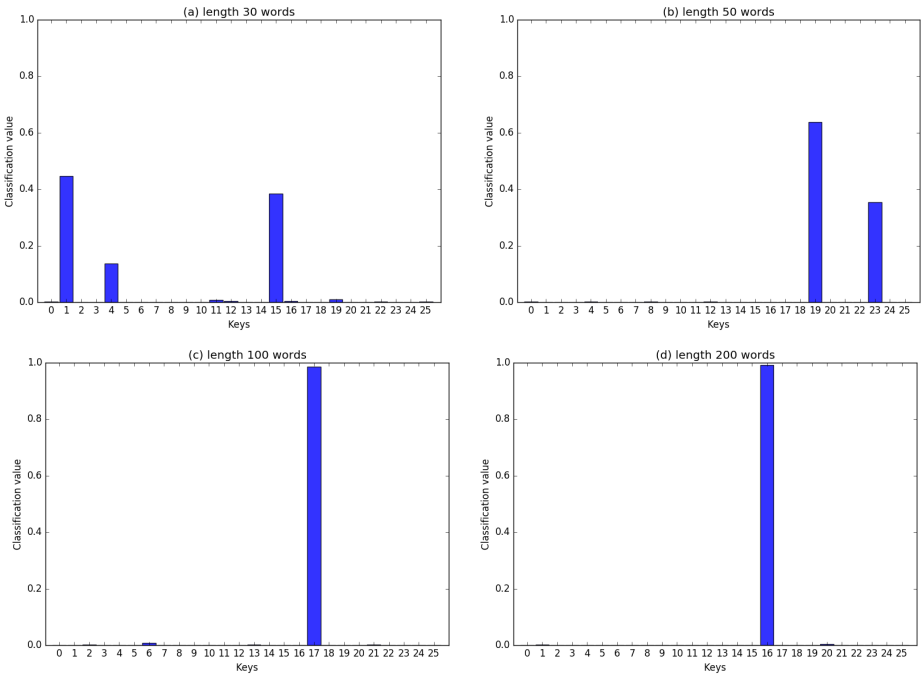

Fig. 2: Worst-case classification value for different text sizes.

typical of mono-alphabetic ciphers (cf. subsection 3.1). Figure 3(b) shows the histogram of the frequencies after a Vigenère encryption. Compared to Figure 3(a) we clearly notice a more uniform shape due to the poly-alphabetic nature of the cipher.

A Vigenère cipher (cf. Definition 2) adds a password of length m to each plaintext block of length m. A very efficient cryptanalysis technique for this cipher is due to Friedman [10,11,12]. The idea is to use statistical measures of coincidence such as the *Index of Coincidence* (IC) to check weather a subtext of the given cipher looks like English in terms of frequencies. IC is the probability that two randomly selected letters are equal and is computed as the sum of the squared probabilities of the letters. It is preserved by mono-alphabetic transformations. Thus, in order to detect m it is then enough to try all possible key lengths and compute the IC of the subtext only composed of the letters at distance m in the text. The IC will be close to the one of English only when m is correct, i.e., when subtext letters are shifted by the same value. After m has been found, Friedman adopted a variant of IC called *Mutual Index of Coincidence* (MIC) to detect the relative shift due to the different password letters. Friedman's cryptanalysis is very efficient. Notice, in fact, that brute-forcing the key length m by repeating an efficient computation (whose complexity is constant) m times is perfectly fine since encryption and decryption have a complexity polynomial in m.

Here, we resort to the same idea of brute-forcing m but we fully reuse the Caesar classifier of subsection 3.1 to check the correctness of m by applying it to the m subtexts.

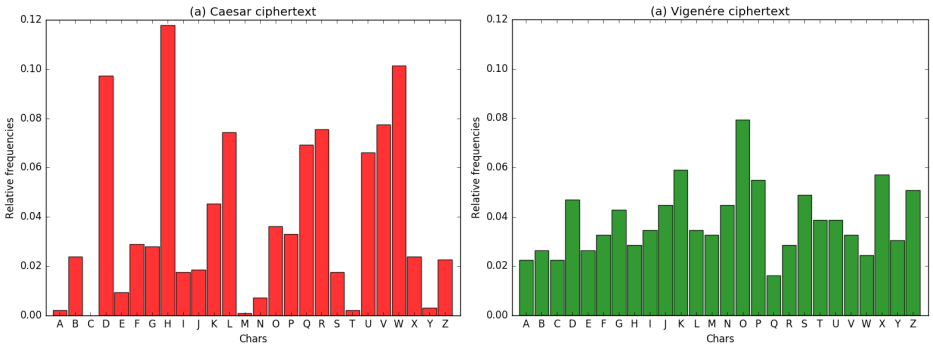

Fig. 3: Comparison between Caesar's and Vigenère's ciphertext frequencies.

So our attack requires $O(m^2)$ invocations of the neural network Caesar classifier which, in turn, is very efficient. In fact, the Vigenère cipher is the application of m different Caesar ciphers on m subtexts composed of the letters at distance m. Interestingly, when we apply the Caesar classifier on Vigenère (poly-alphabetic) encryption we typically get a rather uniform distribution without a neat choice of a key. Thus, our Caesar classifier can be used to check that a certain m is correct and, when it is the case, it directly returns the Vigenère key.

More precisely our attack proceeds as follow. Let $C = c_0, \ldots, c_{n-1}$ be the Vigenère ciphertext to crack and let MAX be the maximum key length the attacker wants to try. Then:

For each integer m in $[1, MAX]$:

1. Consider all m subtexts S_i of C composed of the letters at distance m, i.e., c_0, c_m, $c_{2m}, \ldots, c_1, c_{m+1}, c_{2m+1}, \ldots, \ldots, c_{m-1}, c_{2m-1}, c_{3m-1}, \ldots$;
2. Apply the Caesar classifier to all the S_i's;
3. If the classifier returns a value bigger than a threshold t for all the S_i's output the found key.

The complexity of this attack is $O(MAX^2)$ and can break a cipher almost immediately.

Experimental results. We have tested the attack on texts of length 400 and 1000 words with thresholds 0.95 and 0.98, respectively. Keys were chosen at random from length 5 to length 8. For the texts of 400 words we had about 1% of spurious keys, i.e., keys surviving the thresholds but incorrect. This usually happens when the key is particularly redundant (equal letters) and the classifier detects a wrong key size. It is worth noticing that even when spurious keys are found the attack always outputs also the correct key. For texts of 1000 words the attack is very accurate and we did not get any spurious keys in the tests. As for the Ceasar cipher, cracking a ciphertext is almost instantaneous.

3.3 Neural-cryptanalysis of the substitution cipher

Even if it belongs to the mono-alphabetic class of ciphers, the substitution cipher (cf. Definition 3) presents the very challenging issue of an extremely large key-space. Recall

that keys are permutations of the alphabet, thus giving $26! \approx 2^{88.38}$ possible keys, i.e., more than 88-bits keys. This cipher suffers from the same problem of the shift cipher for what concerns letter frequencies: they are preserved by the encryption. However, differently from the shift cipher, here the histogram is permuted based on the substitution key and the trivial attack of choosing the permutation that best fits the English histogram does not produce any useful plaintext. This is due to the fact that many letters in English present very similar frequencies and it is very likely that they get swapped when shorter texts are sampled.

Example 1 (Failure of trivial frequency cryptanalysis). Consider the following plaintext of 200 words (with spaces removed, and cut down to only three lines):

```
MAYWANTTOMODIFYSUCHFORMULATIONSALONGTHELINESAREADERISLIKEL
YTOFEELATTHISPOINTORMANYREADERSMAYRESPONDBYORPOSSIBLYBETTE
RBYTACKLINGTHEISSUEOFHETEROGENEOUSREADERRESPONSESMOREDI...
```

The following ciphertext has been obtained by encrypting the plaintext under the random substitution `VETISLFMBDGNCYQHJPXZAORKUW` (meaning that `A` becomes `V`, `B` becomes `E`, and so on):

```
CVURVYZZQCQIBLUXATMLQPCANVZBQYXVNQYFZMSNBYSXVPSVISPBXNBGSN
UZQLSSNVZZMBXHQBYZQPCVYUPSVISPXCVUPSXHQYIEUQPHQXXBENUESZZS
PEUZVTGNBYFZMSBXXASQLMSZSPQFSYSQAXPSVISPPSXHQYXSXCQPSIB...
```

By mapping the letters of the ciphertexts depending on the relative frequencies with respect to the relative frequencies in English we obtain the following unsatisfactory result, that only matches 5 of the plaintext letters B, E, H, K and V:

```
GOFYORIITGTLAPFNCUHPTSGCDOIATRNODTRWIHEDARENOSEOLESANDAKED
FITPEEDOIIHANMTARITSGORFSEOLESNGOFSENMTRLBFTSMTNNABDFBEIIE
SBFIOUKDARWIHEANNCETPHEIESTWERETCNSEOLESSENMTRNENGTSELA...
```

This is a particularly unfortunate case but it is very hard to get more than 10-11 correct letters out of 26 with this method.

It is well known that by looking for n-grams, i.e., sequences of n letters, it is possible to have a much more accurate model of a language. However, since each letter is replaced with a different one independently of the other letters, there is no direct way to use n-gram frequencies and come up with a possible substitution key, as we did in the example above for frequencies of single letters.

Instead, the commonly adopted idea is to use n-grams to evaluate how far a given text is from an English one, so that it is possible to look for a better substitution, e.g., by swapping two letters. Intuitively, n-grams are used to define a score function that tells how good is a given key and allows for searching a better one by random swaps.

As we did before, we leverage on this idea but we leave to neural networks the task of evaluating a text based on n-grams. In particular we proceed as follows:

1. we take a big enough dataset of English plaintexts encrypted under random keys;
2. we compute the frequencies of 3-grams of both plaintexts and ciphertexts;

For $i \in [1, MAXITER]$:

1. Pick a random key_i (or the one maximising the match with single frequencies for $i = 1$);
2. Use the neural network to get a $goodness_i$ value;
3. For $MAXSWAPS$ iterations:
   (a) swap two letters in the key and recompute $goodness_i$;
   (b) if $goodness_i$ is better save the new key in key_i;

Output key_j such that $goodness_j = max(\{goodness_i \mid i \in [1, MAXITER]\})$.

Table 1: Attack strategy

3. we train a neural network by providing the frequencies as input and one bit of output: 1 when the input is a plaintext, and 0 when it is a ciphertext;
4. we test the neural network on an independent dataset.

As before, we only provided the neural network with what we know is a feature used by cryptanalysis and we leave to the network the job of devising a classifier for us. In fact, the neural network is trained to distinguish a plaintext by a ciphertext encrypted under a random permutation of the alphabet. We have no guarantees that this will provide a good classifier which is able to tell "how" similar is a text to a plaintext and, consequently, how good is a key, but experimental results have confirmed that the method is effective.

Experimental results. To limit the dimension of the network we have limited the 3-grams frequencies to the most likely 3-grams in English, that we obtained by the National British Corpus [1]. Our previous experiment was based on 2-grams, so we decided to keep the same number of inputs, i.e., $26^2 = 676$. More specifically, the neural network has a hidden layer of 676 neurons and a single neuron as output, giving the float value in the interval [0,1]. Experiments have shown that increasing the size just slowed down training without improving accuracy.

The attack strategy picks a random key, decrypts the ciphertext and uses the neural network to estimate how close the obtained plaintext is to English and, consequently, to the target plaintext. It then swaps random letters looking for a better plaintext. If the neural network gives a better score the new key is kept otherwise a new random swap is tried. The attacker can select the bound $MAXSWAPS$, that is the maximum number of possible swaps. To prevent that the attack only finds a local maximal solution, the attack can be restarted with a new initial random key. The attack can be improved by picking as starting key for the first iteration the one maximizing the similarity of single letter frequencies illustrated in Example 1. Table 1 summarizes the attack strategy.

Example 2 (Attack strategy for substitution cipher). Consider again the ciphertext of Example 1. We start from the (unsatisfactory) decryption that is obtained by making the frequency of single letters match:

```
GOFYORIITGTLAPFNCUHPTSGCDOIATRNODTRWIHEDARENOSEOLESANDAKED
FITPEEDOIIHANMTARITSGORFSEOLESNGOFSENMTRLBFTSMTNNABDFBEIIE
```

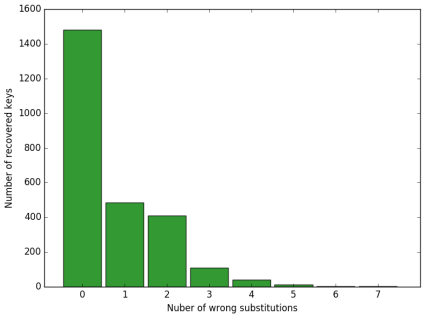

Fig. 4: Number of wrong substitutions in the recovered keys out of 2500 texts.

SBFIOUKDARWIHEANNCETPHEIESTWERETCNSEOLESSENMTRNENGTSELA...

The neural network gives a $goodness$ value of about 0.38, confirming that the text is far from being an English sentence. The attack strategy picks a random swap of letter C with H, giving the following text with $goodness = 0.78$:

GOFYORIITGTLAPFNHUCPTSGHDOIATRNODTRWICEDARENOSEOLESANDAKED
FITPEEDOIICANMTARITSGORFSEOLESNGOFSENMTRLBFTSMTNNABDFBEIIE
SBFIOUKDARWICEANNHETPCEIESTWERETHNSEOLESSENMTRNENGTSELA...

Since $goodness$ is improved the new key is kept and the process goes on up to the following text with $goodness = 0.9998$ recovering the full plaintext:

MAYWANTTOMODIFYSUCHFORMULATIONSALONGTHELINESAREADERISLIKEL
YTOFEELATTHISPOINTORMANYREADERSMAYRESPONDBYORPOSSIBLYBETTE
RBYTACKLINGTHEISSUEOFHETEROGENEOUSREADERRESPONSESMOREDI...

We have tested the technique on 2500 texts of 200 words. As detailed in Figure 4, in about 58% of the cases the key has been fully recovered. In 93% of the cases the key has at most 2 wrong mappings, which are trivially recoverable by manual inspection. Interestingly, in about 70% of the cases the best key has been found at the first iteration. The attacks have been performed with $MAXITER = 10$ and $MAXSWAPS = 400$ and took about 30 seconds for each ciphertext. However, since in 70% of the cases the key is found at the first iteration, the attack only takes 3 seconds to find the correct key.

4 Conclusions

In this paper we have shown how to leverage on existing cryptanalysis techniques for classic ciphers in order to devise neural networks that can automate part of the cryptanalytic attack. We have pointed out that neural networks allow the cryptanalyst to only focus on the interesting features that characterize the weakness of the cipher, leaving to the networks the job of correlating the features and producing either the key material (in case of Caesar and Vigenère) or a measure of the *goodness* of the brute-forced key. We have also presented the first attack on substitution ciphers based on neural networks.

As a future work we plan to investigate the general applicability of our approach by identifying classes of ciphers that can be analyzed using techniques similar to the ones presented here. In particular, we intend to target already broken ciphers, such as the ones in the automotive industry [21,22], and reduced-round standard ciphers.

References

1. British national corpus. `http://www.natcorp.ox.ac.uk/`.
2. Keras: The python deep learning library. `https://keras.io/`.
3. Tensorflow: An open-source machine learning framework for everyone. `https://www.tensorflow.org/`.
4. M. Al-Ubaidy. Black-box attack using neuro-identifier. *Cryptologia*, 24(8):358–372, 2004.
5. A. Albassal and A. Wahdan. Neural network based cryptanalysis of a Feistel type block cipher. In *Int. Conf. on Electrical, Electronic and Computer Eng.*, pages 231–237, 2004.
6. W.S. Awad and E.M.El-Alfy. Computational intelligence in cryptology. In *Improving Information Security Practices through Computational Intelligence*, chapter 2, pages 28–44. Hershey, PA: IGI Global, 2015.
7. E. Biham and A. Shamir. Differential cryptanalysis of DES-like cryptosystems. *Journal of Cryptology*, 4(1):3–72, Jan 1991.
8. A. Biryukov, O. Dunkelman, N. Keller, D. Khovratovich, and A. Shamir. Key Recovery Attacks of Practical Complexity on AES-256 Variants with up to 10 Rounds. In *Advances in Cryptology – EUROCRYPT 2010*, pages 299–319. Springer Berlin Heidelberg, 2010.
9. M. Danziger and H. M. A. Henriques. Improved cryptanalysis combining differential and artificial neural network schemes. In *International Telecommunications Symposium*, August 2014.
10. M. Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32:675–701, 1937.
11. M. Friedman. A correction: The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 34:109, 1939.
12. M. Friedman. A comparison of alternative tests of significance for the problem of m rankings. *The Annals of Mathematical Statistics*, 11:86–92, 1940.
13. A.N. Gomez, S. Huang, I. Zhang, B.M. Li, M. Osama, and L. Kaiser. Unsupervised cipher cracking using discrete GANs. In *Int. Conf. on Learning Representations*, 2018.
14. I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.
15. S. Greydanus. Learning the Enigma with recurrent neural networks. *CoRR*, abs/1708.07576, 2017.
16. Y. Liu, J. Chen, and L. Deng. Unsupervised sequence classification using sequential output statistics. In *31st Annual Conf. on Neural Information Processing Systems*, 2017.
17. H. Maghrebi, T. Portigliatti, and E. Prouff. Breaking cryptographic implementations using deep learning techniques. In *SPACE*, LNCS 10076, pages 3–26. Springer, 2016.
18. RL. Rivest. Cryptography and machine learning. In *Advances in Cryptology (ASIACRYPT'91)*, volume 739 of *Lecture Notes in Computer Science*. Springer, 1993.
19. G. Sivagurunathan, V. Rajendran, and T. Purusothaman. Classification of substitution ciphers using neural networks. *Int. Jour. of Comp. Sc. and Network Security*, 10(3):274–279, 2007.
20. D.R. Stinson. *Cryptography, Theory and Practice*. CRC Press, 2003. Third edition.
21. R. Verdult, F.D. Garcia, and J. Balasch. Gone in 360 seconds: Hijacking with Hitag2. In *21st USENIX*, pages 237–252, 2012.
22. R. Verdult, F.D. Garcia, and B. Ege. Dismantling megamos crypto: Wirelessly lockpicking a vehicle immobilizer. In *22th USENIX*, pages 703–718, 2013.