UNIVERSITY OF TORINO

M.Sc. in Stochastics and Data Science

Final dissertation

Thesis Title

Supervisor: Elena Cordero
Co-supervisor: Stefano Barbero

Candidate: Francesco Moraglio

ACADEMIC YEAR 2019/2020

Summary

Insert here a summary of your thesis

Acknowledgements

You can insert here possible thanks and acknowledgements

Contents

List of Tables

List of Figures

Chapter 1

Neural Cryptography 1: History

1.1 Pioneers of Neural Cryptography

1.2 Neural Networks in Cryptography: an interesting attempt

This section describes one of the first attempts in designing a neural network to be practically used in both cryptography and cryptoanalysis. It must be said that the results attained in Volná (2000), the paper to which I refer, are still quite rough. However, its importance lies in influencing subsequent works. (CITATIONS NEEDED!!!!!).

1.2.1 Genetic Algorithms in Neural Network Design

Main element in this research are feedforward neural nets with backpropagation, but the most interesting characteristic of Volna's approach is that it relies on EP (Evolutionary Programming): genetic algorithms are used for optimization of the designed NN topology. This is based on a previous work of the same author, that is Volná (1998).

The criterion of choice is the minimization of the sum of square of deviation of output from neural network. At first, the maximal architecture of the nets is proposed, then, at each step, to optimize the population it is necessary to solve the cryptographic problems of interest. Thereafter the process of genetic algorithms is applied. An optimal population is found either when it achieves the maximal generation or when fitness function achieves the maximal defined value.

At this point, it is required to complete the "best" architecture by adapting weights and hence three digits are generated for every connection coming out from a unit. If the connection does not exist, three zeroes are assigned, else weights are computed this way:

$$w_{i,j,k,l} = \eta[e_2(e_1 2^1 + e_0 2^0)], \tag{1.1}$$

where $w_{i,j,k,l} = w(x_{i,j}, x_{k,l})$ is the weight value between the j-th unit in the i-th layer and the l-th unit in the k-th layer and

$$\begin{aligned} \eta &= \text{learning parameter}; \quad \eta \in (0,1) \\ e_i &= \text{random digits} \quad (i = 0,1) \\ e_2 &= \text{sign bit}. \end{aligned}$$

Error between the desired and the real output is the computed and stored in the vector $\vec{E}$. On the basis of it, the algorithm computes the fitness precursor value $f_i^\star$, for each individual $i = 1, \ldots, N$, that is

$$f_i^\star = k_1(E_i)^2 + k_2(U_i)^2 + k_3(L_i)^2, \tag{1.2}$$

where k_j, $j = 1{,}2{,}3$ are fixed constants and

$$\begin{aligned} E_i &= \text{error for network } i \\ U_i &= \text{number of hidden units} \\ L_i &= \text{number of hidden layers.} \end{aligned}$$

The general fitness function f is then calculated as follows:

$$f_i = \begin{cases} k - (f_i^\star + k_5) & \text{if} \quad E_i > k_4 \\ k - f_i^\star & \text{otherwise.} \end{cases}$$

In the above expressions, k, k_4 and k_5 also denote constants. The genetic algorithm used by Volná makes use of standard crossover and mutation procedures, as the ones described in the specific chapter. Here we omit details. Adaptation of the best found network architecture is finished with back-propagation.

1.2.2 Volná's experiment

In this work, the parameters of the adapted neural network become the key of an encryption/decryption algorithm. Topology of such NN clearly depends on the training set that, in Volná's case, is represented in table 1.1, while the chain of chars of the plain text is equivalent to a binary value, that is 96 less than its ASCII code. The cipher text is a randomly generated chain of bits. Thus, the decrypting neural network has six input units and five output ones, with an unspecified number of hidden units. Viceversa, the net that performs encryption has five input neurons and six output ones.

This encryption scheme is symmetric: it uses a single key for both encryption and decryption. It is interesting to notice that Volná, in his publication, thought that this feature was very bad for his encryption system, due to the popularity and goodness of asymmetric, non-neural cryptography. In fact, this model has many limits, but we'll see in next chapters that most modern (and secure) neurocryptographic protocols still are symmetric. Leaving aside asymmetric protocols is indeed one of the main strengths of this new approach to cryptography.

Going back to the protocol, the key will include the adapted neural network parameters; that is its topology (architecture) and its configuration (the weight values on connections). Uniquely identifying the NN is hence equivalent to uniquely characterizing the encryption/decryption function.

Plaintext			**Cyphertext**
Char	*ASCII Code*	*Bit String Representation*	*Bit String Representation*
a	97	00001	000010
b	982	00010	100110
c	99	00011	001011
d	100	00100	011010
e	101	00101	100000
f	102	00110	001110
g	103	00111	100101
h	104	01000	010010
i	105	01001	001000
j	106	01010	011110
k	107	01011	001001
l	108	01100	010110
m	109	01101	011000
n	110	01110	011100
o	111	01111	101000
p	112	10000	001010
q	113	10001	010011
r	114	10010	010111
s	115	10011	100111
t	116	10100	001111
u	117	10101	010100
v	118	10110	001100
w	119	10111	100100
x	120	11000	011011
y	121	11001	010001
z	122	11010	001101

Table 1.1: The training set.

1.3 The KKK Key Exchange Protocol

We now deal with the first complete cryptosystem based on neural networks. It has been later referred to as *KKK*, from the surnames of its inventors. It first appeared in Kanter, Kinzel and Kanter (2001), a year later than Volná's work.

Here, a new concept appears in neurocryptography: synchronization of two nets to build a secure communication channel.

1.3.1 The Protocol

Object of the above cited work is a key-exchange protocol based on a learning process of feedforward neural networks. The two NN's participating in the communication start from private key vectors $E_k(0)$ and $D_k(0)$. Mutual learning from the exchange of public information leads the two nets to develop a common, time dependent key: $E_k(t) = -D_k(t)$. This is then used for both encryption and decryption.

This phenomenon, known as synchronization of synaptic weights, has the core feature of speed. In fact, experiments of the authors show that such synchronizing is faster than the process of tracking the weights of one of the networks by an eavesdropper. It must be said that the inventors weren't able to find a mathematical proof of this, that instead was published little later by other cryptographers in Klimov, Mityagin and Shamir (2002). In this same work, all of the limitations of *KKK* are also shown, but we'll deal with this in the following subsections (Synchronization: 1.3.2; Attacks: 1.3.3).

Going back to the model, the architecture used by both sender and recipient is a tree parity machine, a two-layered perceptron with K hidden units, $K \times N$ input neurons and a single output. Input units take binary values $x_{k,j} = \pm 1$, $k = 1, \ldots, K$ and $j = 1, \ldots, N$. The K binary hidden units are denoted by y_k, $k = 1, \ldots, K$, while the integer weight from the j-th input unit from the k-th hidden unit is denoted $w_{k,j} \in \{-L, \ldots, L\}$. Output O is the product of the state of the hidden neurons. (ADD FIGURE!!!)

Fix for simplicity $K = 3$ and let $w^S_{k,j}$, $w^R_{k,j}$ be the secret information of sender and recipient, respectively (that is, the initial values for the weights). Hence this consists of $3N$ integer numbers for each of the two participants.

Each network is then trained with the output of its partner. At each step, both for synchronization and for encryption/decryption steps, a new common public input vector is needed. Given $\vec{x_{k,j}}$, output is computed in two steps. In the first one, states of hidden units are computed as

$$y_k^{S/R} = sgn\left(\sum_{j=1}^{N} w_{k,j}^{S/R} x_{k,j}\right), \tag{1.3}$$

with the convention $y^S_k = 1$ and $y^R_k = -1$ whenever argument of the sign function is zero. In second step, output is computed as the product of the hidden units:

$$O^{S/R} = \prod_{k=1}^{3} y_k^{S/R}. \tag{1.4}$$

Sender and recipient send their outputs to each other and in case they do not agree on them (if $O^S O^R < 0$), weight are updated according to the following Hebbian rule:

$$\begin{aligned} \text{if} \quad \left(O^{S/R} y_k^{S/R} > 0\right) \quad \text{then} \quad & w_{k,j}^{S/R} \leftarrow w_{k,j}^{S/R} - O^{S/R} x_{k,j}; \\ \text{if} \quad \left(|w_{k,j}^{S/R}| > L\right) \quad \text{then} \quad & w_{k,j}^{S/R} \leftarrow sgn\left(w_{k,j}^{S/R}\right) L. \end{aligned} \tag{1.5}$$

Note that this algorithm only updates weights belonging to the hidden units which are in the same state as that of their output unit.

Synchronizing time depends on the choice of the parameters, but this learning rule implies that, as soon as the two networks are synchronized, so they stay forever. Moreover, it was already clear for these experimenter that those times were relatively short compared to the task of externally intercepting weights of one of the two participants (using the same net structure). As soon as the weights are antiparallel, the initialization of the cryptosystem is completed and the secure communication may start. Here we have two possibilities: either use a conventional algorithm, for example a stream cipher, or use the parity machine itself. In the first case, we can build the seed for a pseudo-random number generator basing on the weight vector after synchronization. In the other case, we directly use the output bit of the net for a stream cipher. Note that, using this approach, the complexity of encryption/decryption is linear.

1.3.2 More details on Synchronization

In Kanter's work, many experiments are reported to show the effectiveness of synchronization process. However, I decided to omit these details since, as I anticipated in previous section, the "sacred cow" of cryptography Adi Shamir provided us with a mathematical proof of it.

The full treatment of this problem found in Klimov, Mityagin and Shamir (2002) relies on the properties of random walks in bounded domains. Another assumption: learning rule 1.5 would complicate the notation, as it forces the mutually learning NN's into anti-parallel states. Instead, a dual scheme in which the two parties eventually become identical is proposed. Given input vectors $\vec{x_{k,j}}$, hidden outputs are calculated as

$$y_k^{S/R} = sgn\left(\sum_{j=1}^{N} w_{k,j}^{S/R} x_{k,j}\right), \tag{1.6}$$

but with the standard convention

$$sgn(x) = \begin{cases} 1 & \text{if} \quad x \geq 0 \\ -1 & \text{otherwise.} \end{cases} \tag{1.7}$$

Final output is computed in the same way as before, that is $O^{S/R} = \prod_{k=1}^{3} y_k^{S/R}$. In this version of the algorithm, after the output exchange, each party updates its weights only if $O^S = O^R = O$ and in this case

$$\text{if} \left(y_k^{S/R} = O \right) \quad \text{then} \quad w_{k,j}^{S/R} \leftarrow B_{-L,L} \left(w_{k,j}^{S/R} - y_k^{S/R} x_{k,j} \right), \tag{1.8}$$

where

$$B_{-L,L}(w) = \begin{cases} w & \text{if} \quad |w| < L \\ L & \text{if} \quad w > L \\ -L & \quad \text{otherwise.} \end{cases} \tag{1.9}$$

In other words, above function rescales weights to the prefixed bounds whenever any of them exceeds the allowed values.
After these premises, we can show how the two participants converge and why a third net cannot converge to the same parameters by following the same learning procedure.
We start off by considering an oversimplified model of a single perceptron with a single weight. Let $w_t^{S/R}$ be the current weights and denote inputs $x_t \in \{-1,1\}$. Each weight's update process can be described as a random walk with absorbing boundaries $-L, L$, starting from a random point $w_0^{S/R} \in \{-L, \dots, L\}$.
At each round t, sender and receiver decide either to move their respective weights in the same direction (determined by x_t), or not to vary them. In first case, if any of $w_t^{S/R}$ tries to step beyond the fixed boundaries, it remains stuck at it (due to 1.9), while the other one gets nearer. If none of the the weights is absorbed, we have $|w_{t+1}^S - w_{t+1}^R| = |w_t^S - w_t^R|$, else their distance is reduced by one. Since a random walk is expected to hit its boundaries infinitely often, the weights'paths admit a mixing time.
A simple generalization: consider the case of a single perceptron with multiple weights. Here, the two weight vectors move in the same direction determined by inputs in a multidimensional rectangle, and along each coordinate the distance is either preserved or reduced by one. When all these distances are reduced to zero, the two random walks mix.
The general case of neural networks with multiple perceptrons is more complicated, since the two parties may update different subsets of their perceptrons in each round. The complete treatment of such problem can be

found in the publication to which I'm referring, but I decided to omit these details.

1.3.3 Attacking KKK

The creators of *KKK* claimed its security basing on their experiments. In fact, simulations shown clearly that it was computationally unfeasible for an attacker to intercept any of the parties' parameters, if such eventual attacker relied on the same neural network structure as the ones used in communication.

In Shamir's paper there's also a mathematical proof of this impossibility, but most interesting content of such work is in its last section. This is dedicated to the cryptanalysis of *KKK*, which can indeed be broken by using other types of attack.

For example, many successful cryptanalytic applications of Genetic Algorithms can be found in literature.

In this case, a large population of NN's with the same structure as the two participants is simulated. These nets are then trained with the same inputs of the two communicant ones. Networks whose outputs mimic those of the two parties breed and multiply, while unsuccessful networks die.

Shortly after sender S and recipient R synchronize, they can be certain of this fact (since their outputs keep coinciding) and the same can be checked for any of the attacker's neural networks.

An interesting experimental result: in the majority of tests led by the authors, at least one of the attacking nets became synchronized with S even before S and R became fully synchronized.

Other examples of successful attacks to the *KKK* protocol are the one based on geometrical reasoning and the probabilistic case. The former exploits the geometrical representation of inputs and weight in an algorithm that guesses hidden outputs in the two parties'nets. The latter, instead, is again based on the properties of generalized random walks: these allow us to compute conditional probabilities for the values of hidden states.

Chapter 2

Chapter title

2.1 Section title

Body of text, with unnumbered equations

$$Y = \beta_0 + \beta_1 X + \varepsilon$$

if not referenced, or numbered equations

$$Y = \beta_0 + \beta_1 X + \varepsilon \tag{2.1}$$

to be referenced like this (2.1) if needed.

Start of a new paragraph here, where you can have inline math $y = a + bx$.

2.2 Another section

Tex of the section with example of theorem

Theorem 2.2.1. *Example of theorem*

Proof. proof of the theorem □

to be referenced like Theorem 2.2.1.

Same for proposition

Proposition 2.2.2. *Example of proposition*

or lemma

Definition 2.2.3. Example of definition □

Remark 2.2.4. Example of remark □

Lemma 2.2.5. *Example of lemma*

Algorithm 1: Algorithm title

```
Data: $y_{t_j} = (y_{t_j,1}, \dots, y_{t_j,m_{t_j}})$
Set parameters $\alpha = \theta P_0$, $\theta > 0$, $P_0 \in M_1(\mathbb{Y})$
  Initialise
    $y \leftarrow \emptyset$, $y^* = \emptyset$, $m \leftarrow 0$, $M \leftarrow 0$, $M \leftarrow \{0\}$, $K_m \leftarrow 0$, $w_0 \leftarrow 1$
  For $j = 0, \dots, J$
      Title set of instructions 1
          read data $y_{t_j}$
          $m \leftarrow m + \text{card}(y_{t_j})$
          $y^* \leftarrow$ distinct values in $y^* \cap y_{t_j}$
          $K_m = \text{card}(y^*)$
      Title set of instructions 2
          for $M \in M$
              $n \leftarrow t(y_{t_j}, M)$
              $w_n \leftarrow w_M \, \text{PU}_\alpha(y_{t_j} \mid y)$
          $M \leftarrow t(y_{t_j}, M)$
          for $M \in M$
              $w_M \leftarrow w_M / \sum_{\ell \in M} w_\ell$
          $X_{t_j} \mid y, y_{t_j} \sim \sum_{M \in M} w_M \Pi_{\alpha + \sum_{i=1}^{K_m} m_i \delta_{y_i^*}}$
      Return $y \leftarrow y \cup y_{t_j}$
```

Example of pseudo code of algorithm which is referred as Algorithm 1.

Example of table

Temperatura °C	Densità t/m^3
0	13,8
10	13,6
50	13,5
100	13,3

Table 2.1: Densità del mercurio. Si può fare molto meglio usando il pacchetto booktabs.

Items in the bibliography to be referenced like this Ethier and Kurtz (1986) and this Ethier and Kurtz (1981), check the different style for books and articles.

Abbreviations of Journal names can be found at this link msc2010.org/MSC2010-CD/extras/serials.pdf

Bibliography

Ethier, S.N. and Kurtz, T.G. (1981). The infinitely-many-neutral-alleles diffusion model. *Adv. Appl. Probab.* **13**, 429–452.

Ethier, S.N. and Kurtz, T.G. (1986). *Markov processes: characterization and convergence.* Wiley, New York.

Mitchell, M. (1998). *An introduction to Genetic Algorithms.* MIT Press.

Volnà, E. (2000). *Using Neural Network in Cryptography.* University of Ostrava.

Volnà, E. (1998).*Learning algorithm which learns both architectures and weights of feedforward neural networks.* Neural Network World. Int. Journal on Neural and Mass-Parallel Compo and Inf. Systems.

Kanter, I., Kinzel, W. and Kanter, E. (2001). *Secure exchange of information by synchronization of neural networks.* Bar Ilan University.

Klimov, A., Mityagin, A. and Shamir, A. (2002). *Analysis of Neural Cryptography.* Weizmann Institute.

Abadi, M. and Andersen, D. G. (2016). *Learning to protect communications with Adversarial Neural Cryptography.* Google Brain.

Coutinho, M., Robson de Oliveira Albuquerque, R., Borges, F. , Villalba, L. J. G. and Kim T. H. (2018). *Learning Perfectly Secure Cryptography to Protect Communications with Adversarial Neural Cryptography.* University of Brasília.

Jayachandiran, K. (2018). *A Machine Learning Approach for Cryptanalysis.* Rochester Institute of Technology.

Gohr, A. (2019). *Improving Attacks on Round-Reduced Speck32/64 Using Deep Learning.* Bundesamt für Sicherheit in der Informationstechnik (BSI).

NIELSEN, M. A. and CHUANG, I. L. (2000). *Quantum Computation and Quantum Information*. Cambridge University Press.

BERNSTEIN, D. J., BUCHMANN, J. and DAHMEN, E. (2009). *Post-Quantum Cryptography*. Springer.

SHI, J., CHEN, S., LU, Y., FENG, Y., SHI, R., YANG, Y. and LI, J. (2020). *An Approach to Cryptography Based on Continuous-Variable Quantum Neural Network*. Nature.