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

Introduction to Neural Networks

First chapter of this work is dedicated to its main ingredient: Artificial Neural Networks. These are models of computation based loosely on the way in which the brain is believed to work. Since their invention, biologists are interested in using such networks to model biological brains, but most of the impetus comes from their employment in applied sciences: NN's allow us to create machine that can perform "cognitive" tasks.

1.1 Biological Perspective

To start off, an overview on biological neural network is needed, as ANN's in principle attempt to simulate them. An extensive treatment of both principles and applications of neural networks can be found in Graupe (2007), to which I make reference for this chapter.
The biological neural network consists of interconnected nerve cells (called neurons), whose bodies are where most of the neural "computation" takes place. Neural activity passes from one neuron to another through electrical signals which travel from one cell to the following down the neuron's axon, that can be seen as a connection wire. However, the mechanism of signal transmission in not via electrical conduction, but via charge exchange that is transported by diffusion ions. At the terminal end of the axon, a synaptic gap takes place and an electro-chemical process allows communication with the other cell.
Since a given neuron may have many synapses, it can connect to many other cells. Similarly, since there are many dendrites (input connections) per each

neuron, a single nerve cell can receive signals from many other neurons. It is very important to notice that not all of such interconnections are equally weighted: some have a higher priority than others. Also some are inhibitory and some are excitory. These facts are also key feature of artificial neural networks, as I'll discuss in the following sections.

1.2 Principles

The theoretical principles of the artificial neural networks first appeared in McCulloch and Pitts (1943). In this early work, five assumptions were formulated:

1. the activity of an artificial neuron is all-or-nothing;

2. a certain fixed number of synapses greater than one must be excited for a neuron to be excited;

3. the only significant delay within the neural system is the synaptic delay;

4. the activity of any inhibitory synapse absolutely prevents the excitation of the neuron at that time;

5. the structure of the interconnection network does not change over time.

Another widely applied principle is the so-called *Hebbian Rule* (or *Hebbian Learning Law*), that was first stated in Hebb (1949) as follows.
"When an axon of cell A is near-enough to excite cell B and when it repeatedly and persistently takes part in firing it, then some growth process or metabolic change takes place in one or both these cells such that the efficiency of cell A is increased".
Roughly speaking, such rule can be summarized as "cells that fire together wire together", that is the connections between two neurons might be strengthened (in terms of weights) if the neurons fire simultaneously.
Notice that above historical principles do not all apply to most modern neural network architectures. Following sections contain the description of the neural network models which have been employed (also in) neurocryptography.

1.3 The Perceptron

1.3.1 Basic structure

The first neural computation model, that is called Perceptron, first appeared in (Rosenblatt , 1958) and serves as a building block to most later models. The Perceptron posses the fundamental structure of a neural cell, of several weighted input connections and a single output channel. The input/output relations are defined to be

$$z = \sum_i w_i x_i \tag{1.1}$$

$$y = f_N(z), \tag{1.2}$$

where w_i is the weight at the input x_i. Such weights, according to the biological model, need to be adjustable. y, instead, denotes the cell's output, that is a nonlinear activation function f_N (see 1.3.2 below) of the node output z.
It is customary to consider a network of n Perceptrons; in this case, by letting z_i be the summation output of the i-th Perceptron and its inputs $x_{i,j}$, the summation relation becomes

$$z_i = \sum_{j=1}^{m} w_{ij} x_{ij}, \tag{1.3}$$

or, in vector form

$$z_i = \vec{w}_i^{\mathsf{T}} \vec{x}_i, \tag{1.4}$$

where

$$\vec{w}_i = [w_{i1}, \ldots, w_{in}]^{\mathsf{T}}$$
$$\vec{x}_i = [x_{i1}, \ldots, x_{in}]^{\mathsf{T}}.$$

1.3.2 Activation functions

The Perceptron's cell's output differs from the summation output 1.4 by the activation operation of the neuron's body, just as the output of the biological cell differs from the weighted sum of its inputs. Such operation is in terms of an activation function $f_N(z_i)$, which is a non-linear operator yielding i-th neuron's output y_i to satisfy certain limiting properties. Different functions are in use, but the most common choice is the sigmoid function, that is

$$y_i = f_N(z_i) = \frac{1}{1 + e^{-z_i}}. \tag{1.5}$$

Under this choice, the following relations are satisfied (see figure TO ADD!!).

$$\begin{aligned} z_i \to -\infty &\iff y_i \to 0; \\ z_i = 0 &\iff y_i = 0.5; \\ z_i \to \infty &\iff y_i \to 1. \end{aligned}$$

Another popular activation function is

$$y_i = \frac{1 + tanh(z_i)}{2} = \frac{1}{1 - e^{-2z_i}}, \tag{1.6}$$

whose shape is similar to the one of the previous function. The simplest activation one could choose is the Heaviside step function:

$$y_i = H(z_i) = \begin{cases} 1 & \text{if} \quad z_i \geq 0 \\ 0 & \text{otherwise.} \end{cases}$$

Main advantage of the former two over the latter is that they're (mathematically) smooth.
In many applications the activation function's output is translated so that it ranges in $(-1, 1)$, rather than in $(0, 1)$. In particular, sigmoid function 1.5 becomes

$$y_i = \frac{2}{1 + e^{-z_i}} - 1 = tanh(z_i/2), \tag{1.7}$$

while activation 1.6 is transformed to

$$y_i = tanh(z_i) = \frac{1 - e^{-2z_i}}{1 + e^{-2z_i}}. \tag{1.8}$$

1.3.3 Multi-Layer Perceptron

ADD A FEW WORDS ABOUT MLP, JUST FOR COMPLETENESS

1.4 Back Propagation Training Algorithm

Back Propagation (BP) algorithm was first proposed in Rumelhart, Hinton and Williams (1986), as a solution for setting weights (and hence for the training) of multi-layer Perceptrons. The availability of a rigorous method to set intermediate weights, namely to train the hidden layers of neural networks, gave a major boost to the further development of such models.
The BP algorithm starts with computing values of the output layer, which

is by definition the only one whose desired outputs are available. Let ϵ be the the *error-energy* at the output layer, that is

$$\epsilon = \frac{1}{2}\sum_{k=1}^{N} e_k^2 = \frac{1}{2}\sum_{k=1}^{N} (d_k - y_k)^2 , \tag{1.9}$$

where N is the number of neurons in the output layer. Consider the gradient of ϵ:

$$\nabla\epsilon_k = \frac{\partial\epsilon}{\partial w_k j}. \tag{1.10}$$

Recall that $w_k j$ is denoting the weight of the j-th input to the k-th neuron. By the gradient descent procedure (ADD SOMETHING ABOUT THIS?), we have that

$$w_{kj}(m+1) = w_{kj}(m) + \Delta w_{kj}(m), \tag{1.11}$$

where

$$\Delta w_{kj} = -\eta\frac{\partial\epsilon}{\partial w_{kj}} \tag{1.12}$$

and $\eta \in (0,1)$ is the rate parameter. Note that the minus sign indicates a down-hill direction towards a minimum.
With the notations of 1.3.1, we can now substitute

$$\frac{\partial\epsilon}{\partial w_{kj}} = \frac{\partial\epsilon}{\partial z_k}\frac{\partial z_k}{\partial w_{kj}} \tag{1.13}$$

and

$$\frac{\partial z_k}{\partial w_{kj}} = x_j(p) = y_j(p-1), \tag{1.14}$$

with p denoting the output layer. This way, equation 1.13 becomes

$$\frac{\partial\epsilon}{\partial w_{kj}} = \frac{\partial\epsilon}{\partial z_k}x_j(p) = \frac{\partial\epsilon}{\partial z_k}y_j(p-1). \tag{1.15}$$

We now have to define

$$\phi_k(p) = -\frac{\partial\epsilon}{\partial z_k}, \tag{1.16}$$

so that 1.15 yields

$$\frac{\partial\epsilon}{\partial w_{kj}} = -\phi_k(p)x_j(p) = -\phi_k(p)y_j(p-1). \tag{1.17}$$

So by this last equation and 1.12 we get

$$\Delta w_{kj} = \eta\phi_k(p)x_j(p) = \eta\phi_k(p)y_j(p-1). \tag{1.18}$$

Furthermore, by 1.16:

$$\phi_k = -\frac{\partial \epsilon}{\partial z_k} = -\frac{\partial \epsilon}{\partial y_k}\frac{\partial y_k}{\partial z_k}. \tag{1.19}$$

But, going back to error-energy definition 1.9

$$\frac{\partial \epsilon}{\partial y_k} = -\left(d_k - y_k\right) = y_k - d_k, \tag{1.20}$$

where we recall

$$y_k = f_N(z_k) = \frac{1}{1 + e^{-z_k}} \tag{1.21}$$

and so we have that

$$\frac{\partial y_k}{\partial z_k} = y_k(1 - y_k). \tag{1.22}$$

Consequently, by equations 1.19 and subsequent ones,

$$\phi_k = y_k(1 - y_k)(d_k - y_k), \tag{1.23}$$

such that, at the output layer, by 1.12 and 1.13

$$\Delta w_{kj} = -\eta\frac{\partial \epsilon}{\partial w_{kj}} = -\eta\frac{\partial \epsilon}{\partial z_k}\frac{\partial z_k}{\partial w_{kj}}, \tag{1.24}$$

where, by 1.19

$$\Delta w_{kj}(p) = \eta\phi_k(p)y_j(p - 1), \tag{1.25}$$

with ϕ_k being as in 1.23, to complete the derivation of the setting of output layer's weights.

Chapter 2

Genetic Algorithms: a brief overview

Chapter 3

Neural Cryptography 1: History

3.1 Pioneers of Neural Cryptography

3.2 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!!!!!).

3.2.1 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:

$$w_{ij,kl} = \eta[e_2(e_1 2^1 + e_0 2^0)], \tag{3.1}$$

where $w_{ij,kl} = w(x_{ij}, x_{kl})$ 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

$$f_i^\star = k_1(E_i)^2 + k_2(U_i)^2 + k_3(L_i)^2, \tag{3.2}$$

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

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

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.

3.2.2 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 3.1, 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 3.1: The training set.

3.3 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.

3.3.1 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: 3.3.2; Attacks: 3.3.3).
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 $x_{kj} = \pm 1$, $k = 1, \ldots, K$ 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 $w_{kj} \in \{-L, \ldots, L\}$. Output O is the product of the state of the hidden neurons. (ADD FIGURE!!!)
Fix for simplicity $K = 3$ and let w^S_{kj}, w^R_{kj} 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 $\vec{x_{kj}}$, output is computed in two steps. In the first one, states of hidden units are computed as

$$y_k^{S/R} = sgn\left(\sum_{j=1}^{N} w_{kj}^{S/R} x_{kj}\right), \tag{3.3}$$

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

$$O^{S/R} = \prod_{k=1}^{3} y_k^{S/R}. \tag{3.4}$$

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:

$$\begin{aligned} \text{if} \quad & \left(O^{S/R} y_k^{S/R} > 0\right) \quad \text{then} \quad w_{kj}^{S/R} \leftarrow w_{kj}^{S/R} - O^{S/R} x_{kj}; \\ \text{if} \quad & \left(|w_{kj}^{S/R}| > L\right) \quad \text{then} \quad w_{kj}^{S/R} \leftarrow sgn\left(w_{kj}^{S/R}\right) L. \end{aligned} \tag{3.5}$$

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.

3.3.2 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 3.5 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 $\vec{x_{kj}}$, hidden outputs are calculated as

$$y_k^{S/R} = sgn\left(\sum_{j=1}^{N} w_{kj}^{S/R} x_{kj}\right), \tag{3.6}$$

but with the standard convention

$$sgn(x) = \begin{cases} 1 & \text{if} \quad x \geq 0 \\ -1 & \text{otherwise.} \end{cases} \tag{3.7}$$

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

$$\text{if} \left(y_k^{S/R} = O \right) \quad \text{then} \quad w_{kj}^{S/R} \leftarrow B_{-L,L} \left(w_{kj}^{S/R} - y_k^{S/R} x_{kj} \right), \tag{3.8}$$

where

$$B_{-L,L}(w) = \begin{cases} w & \text{if} \quad |w| < L \\ L & \text{if} \quad w > L \\ -L & \text{otherwise.} \end{cases} \tag{3.9}$$

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 $w_0^{S/R} \in \{-L, \ldots, L\}$.
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 3.9), 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.

3.3.3 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 4

Adversarial Neural Cryptography

After a long period without any substantial contribution to neural cryptography, a new, revolutionary approach made its appearance in 2016, thanks to Google's researchers. That is, neural networks can learn to protect the secrecy of their data from other neural networks: they discover forms of encryption and decryption, without being taught specific algorithms for these purposes. This new approach to cryptography has been named ANC (Adversarial Neural Cryptography).

4.1 Google's Model

The research I'm considering is Abadi and Andersen (2016). In this paper, a new cryptosystem is proposed. At its core there are adversarial neural networks and in this section I'm examining this protocol.

4.1.1 Cryptosystem Organization

Let's start with some notation. In this scenario, we consider the problem of the secure communication between Alice and Bob, denote them $\mathcal{A}$ and $\mathcal{B}$, respectively. Consider also a third participant, Eve ($\mathcal{E}$)), who wishes to eavesdrop on their communication. Such adversary is passive, that means it can only intercept communications.
In this scenario, Alice wishes to send a private message P ("Plaintext") to Bob. Such message must be regarded as an input to $\mathcal{A}$, together with a key K. This key is also known by $\mathcal{B}$, that uses it to decipher $\mathcal{A}$'s output; denote it C ("Ciphertext"). Also Eve knows C and tries to recover P from it, but

E cannot know K. Hence let P_{Bob} and P_{Eve} be the respective outputs.
Alice, Bob, and Eve are represented by, guess what, neural networks, based on a "Mix and Trasform" architecture. This structure has a first fully-connected (FC) layer, where the number of outputs is equal to the number of inputs. P and K are fed into this layer, that enables mixing between the bits of these two vectors. Such layer is followed by a sequence of convolutional layers, the last of which produces an output of a size suitable for a plaintext or ciphertext. These neural networks have parameters, which we write $\theta_{\mathcal{A}}$, $\theta_{\mathcal{B}}$ and $\theta_{\mathcal{E}}$. Moreover, P, C, K, P_{Bob} and P_{Eve} are vectors of float numbers; in particular values are allowed to range in $(-1, 1)$.
Eve's objective is to reconstruct P, that is, to minimize the distance between P and P_{Eve}. Alice and Bob want to communicate clearly (to minimize the error between P and P_{Bob}), but also to hide their communication from $\mathcal{E}$. $\mathcal{A}$ and $\mathcal{B}$ are hence trained jointly to communicate securely and to defeat Eve: here lies the biggest innovation in ANC.
Since we want Alice and Bob to be trained against the best possible version of Eve, we need to assume a probability distribution on plaintexts and keys. After that, we can rephrase the parties' objectives in terms of expectation. Denote Alice and Bob's output as $A(\theta_{\mathcal{A}}, P, K)$ and $B(\theta_{\mathcal{B}}, C, K)$, respectively. Similarly, write $E(\theta_{\mathcal{E}}, C)$ for Eve's output. As anticipated above, we need to choose a distance function on the space of plaintexts. In this model, assuming such P's have length N, we take

$$d(P, P') = \sum_{i=0}^{N} |P_i - P'_i|, \tag{4.1}$$

that is, the L^1 distance. Note that this choice is not crucial. After defining an example-specific loss function for Eve,

$$L_{\mathcal{E}}\left(\theta_{\mathcal{A}}, \theta_{\mathcal{E}}, P, K\right) = d\left(P, E(\theta_{\mathcal{E}}, A(\theta_{\mathcal{A}}, P, K))\right),$$

we can define the "true" loss function for $\mathcal{E}$ by taking expectations over the distributions of the P's and K's:

$$L_{\mathcal{E}}\left(\theta_{\mathcal{A}}, \theta_{\mathcal{E}}\right) = \mathbb{E}_{P,K}\left[d\left(P, E(\theta_{\mathcal{E}}, A(\theta_{\mathcal{A}}, P, K))\right)\right]. \tag{4.2}$$

The optimal configuration for Eve, $O_{\mathcal{E}}$, can hence be found by minimizing the above loss (given a configuration of $\mathcal{A}$) .

$$O_{\mathcal{E}}\left(\theta_{\mathcal{A}}\right) = \text{argmin}_{\theta_{\mathcal{E}}} L_{\mathcal{E}}\left(\theta_{\mathcal{A}}, \theta_{\mathcal{E}}\right) \tag{4.3}$$

Similarly, a per-example loss for Bob is defined and then extended to the distributions of plaintexts and keys:

$$\begin{aligned} L_{\mathcal{B}}\left(\theta_{\mathcal{A}}, \theta_{\mathcal{B}}, P, K\right) &= d\left(P, B(\theta_{\mathcal{B}}, A(\theta_{\mathcal{A}}, P, K), K)\right) \\ L_{\mathcal{B}}\left(\theta_{\mathcal{A}}, \theta_{\mathcal{B}}\right) &= \mathbb{E}_{P,K}\left[d\left(P, B(\theta_{\mathcal{B}}, A(\theta_{\mathcal{A}}, P, K), K)\right)\right]. \end{aligned} \tag{4.4}$$

We now need to define a joint loss function for Alice and Bob, which takes into account $\mathcal{B}$'s loss and the optimal value of Eve's loss (by maximizing $\mathcal{E}$'s reconstruction error). This is:

$$L_{\mathcal{AB}}(\theta_{\mathcal{A}}, \theta_{\mathcal{B}}) = L_{\mathcal{B}}\left(\theta_{\mathcal{A}}, \theta_{\mathcal{B}}\right) - L_{\mathcal{E}}(\theta_{\mathcal{A}}, O_{\mathcal{E}}\left(\theta_{\mathcal{A}}\right)). \tag{4.5}$$

Note that also this choice may be subject to variations. Optimal configurations for Alice and Bob can hence by found as

$$(O_{\mathcal{A}}, O_{\mathcal{B}}) = \text{argmin}_{(\theta_{\mathcal{A}}, \theta_{\mathcal{B}})} L_{\mathcal{AB}}(\theta_{\mathcal{A}}, \theta_{\mathcal{B}}) \tag{4.6}$$

It must be said that such optimal configurations are not necessarily unique. A few words about training. Such procedure is based upon stochastic gradient descent and on estimated values calculated over examples (not on expected values over a distribution). In addition, $O_{\mathcal{E}}$ is not computed for given values of $\theta_{\mathcal{A}}$, but simply approximated. This is done by alternating the training of Eve with that of Alice and Bob.
Roughly speaking, training proceeds follows. At first, $\mathcal{A}$'s outputs may be totally incomprehensible for both $\mathcal{B}$ and $\mathcal{E}$. After a few steps, Bob could discover a way to decrypt Alice's messages at least partially, without Eve being able to do the same. Later, however, $\mathcal{E}$ may start to break this code. Alice and Bob should then find improvements to their security, in a way such that Eve would find impossible to adjust to those ciphers. Note that this kind of alternation is typical in game theory.

4.1.2 ANC and selective protection

Authors of ANC described an experiment to study to test this cryptosystem over selective protection: the question of whether neural networks can learn what information to protect. For example, a plaintext may be made up of several components, of which only few of them are required to be kept secret to the adversary. In this case, selective protection would mean a maximization in utility.
Consider a dataset consisting of vectors of for values, namely (A, B, C, D). The target is to build and train a system that, given as inputs the first three

values, outputs two predictions of D: $\hat{D}$ and D_{public}. The former is most accurate possible estimate, while the latter is defined as the best possible estimate of D that does not reveal any information about the value of C.
As before, Alice and Bob share a key and both $\mathcal{B}$ and Eve have access to $\mathcal{A}$'s outputs, that are D_{public} and a ciphertext T. Her input is (A, B, C). Bob produces $\hat{D}$, while Eve tries to recover C. Using strongly correlated values for (A, B, C) and D, the authors were able to show that the adversarial training permits approximating D without revealing C.

4.2 Improvements: CPA-ANC

This section is dedicated to an extension of Abadi and Andersen's model. In fact, also this last model has some limitations, that are shown in Coutinho et al. (2018). Authors of this paper elaborated an extension to ANC, that is discusses in the following.

4.2.1 Chosen-Plaintext Attack ANC

Main limitation to the security of original ANC model lies in Eve's job, that appears to be too hard. It must decrypt a random message having access only to the ciphertext. The consequence is that, under this methodology, Alice and Bob learn to protect against a weak adversary. It is however possible to strengthen Eve by letting it mount a CPA.
Eve chooses two plaintext messages, namely P_0 and P_1, and sends them to Alice. $\mathcal{A}$ chooses one of the messages randomly, encrypts it to C and sends it to $\mathcal{B}$ and $\mathcal{E}$. The former decrypts the message using a NN, while the latter only outputs 0 if it believes P_0 was encrypted or 1 if it believes P_1 was encrypted.

4.2.2 Neural Network Architecture

A specific network architecture for this model is proposed, based on the continuous extension of logic operator XOR, built to learn an OTP algorithm.
Consider mapping bit 0 to angle 0 and bit 1 to angle π (XOR can be seen as the sum of the angles). Generalizing bits to a continuous space, the following defines the mapping of a bit b into an angle:

$$f(b) = arccos(1 - 2b) \tag{4.7}$$

Then consider its inverse:

$$f^{-1}(a) = \frac{1 - cos(a)}{2} \tag{4.8}$$

The new net, named *CryptoNet*, takes as input P and K and, for each bit, applies 4.7. Next step is a weight matrix (write W) multiplication, followed by the inverse transformation 4.8; output is C. Note bits $C_i \in (0,1)$. *CryptoNet* will be denote mathematically as the function

$$C = \zeta_n(W, P, K), \tag{4.9}$$

where we recall n is the size of vectors P and K.

4.2.3 Method

Let $E_{\mathcal{A}}(\cdot, K)$ be Alice's encryption function and let $D_{\mathcal{B}}(\cdot, K)$ be Bob's decryption function. In this setup, these maps are defined to be the same *CryptoNet*:

$$E_{\mathcal{A}}(\cdot, K) = D_{\mathcal{B}}(\cdot, K) = \zeta_n(W, \cdot, K). \tag{4.10}$$

This means, such network must be the inverse of itself. More details on the training procedure can be found in the original paper.

Chapter 5

Chapter title

5.1 Section title

Body of text, with unnumbered equations

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

if not referenced, or numbered equations

$$Y = \beta_0 + \beta_1 X + \varepsilon \tag{5.1}$$

to be referenced like this (5.1) if needed.

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

5.2 Another section

Tex of the section with example of theorem

Theorem 5.2.1. *Example of theorem*

Proof. proof of the theorem □

to be referenced like Theorem 5.2.1.
Same for proposition

Proposition 5.2.2. *Example of proposition*

or lemma

Definition 5.2.3. Example of definition □

Remark 5.2.4. Example of remark □

Lemma 5.2.5. *Example of lemma*

Algorithm 1: Algorithm title

Data: $y_{t_j} = (y_{t_j,1}, \ldots, y_{t_j,m_{t_j}})$
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 $j = 0, \ldots, J$
  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 5.1: 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.

Graupe, D. (2007). *Principles of Artificial Neural Networks (2nd Edition)*. Word Scientific Publishing, Singapore.

Anthony, M. (2001). *Discrete Mathematics of Neural Networks*. Society for Industrial and Applied Mathematics, Philadelphia.

McCulloch, W. and Pitts, W. (1943). *A logical calculus of the ideas immanent in nervous activity*. Bulletin of Mathematical Biophysics 5, 115-133.

Hebb, D. O. (1949). *The organization of behavior; a neuropsychological theory*. Wiley, New York.

Rosenblatt, F. (1958) *The perceptron: A probabilistic model for information storage and organization in the brain*. Psychological Review, 65.

Rumelhart, D., Hinton, G. and Williams, R. (1986). *Learning representations by back-propagating errors*. Nature 323, 533–536.

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.