A survey on machine learning applied to symmetric cryptanalysis

Rendiconti Sem. Mat. Univ. Pol. Torino
Vol. 80, 2 (2022), 107 – 122

E. Bellini, A. Hambitzer, M. Rossi

A SURVEY ON MACHINE LEARNING APPLIED TO SYMMETRIC CRYPTANALYSIS

Abstract. In this work we give a short review of the recent progresses of machine learning techniques applied to cryptanalysis of symmetric ciphers, with particular focus on artificial neural networks. We start with some terminology and basics of neural networks, to then classify the recent works in two categories: "black-box cryptanalysis", techniques that not require previous information about the cipher, and "neuro-aided cryptanalysis", techniques used to improve existing methods in cryptanalysis.

1. Introduction

The similarities between retrieving secret information from a symmetric cipher and finding the unknown weights of a neural network have been known since long time, for example in Rivest's survey at Asiacrypt 1991 [1]. Due to the constant progress of technology, the adoption of machine learning techniques, in particular neural networks, is becoming increasingly popular and effective in solving more and more complex problems. This success of these techniques has tempted many cryptographers to exploit them for cryptanalysis.

The goal of this survey is to provide an overview of the literature involving machine learning techniques in cryptanalysis of symmetric ciphers. The focus will be set on theoretical cryptanalysis using neural networks, so, in particular, works on side-channel analysis are not taken into account. The analyzed works are classified following a similar scheme to [2], that will be clarified in the following sections.

The survey is organized as follows. In section 2 we give a short introduction of machine learning techniques which have found application in theoretical cryptanalysis of symmetric primitives, with particular focus on neural networks (NN). In section 3 the literature on pure blackbox cryptanalysis via machine learning techniques is reviewed. In section 4 we review the literature on classical cryptanalytic methods enhanced by neural networks, that we will call *neuro-aided cryptanalysis*. In section 5 we conclude the paper.

1.1. Related Works

In an invited talk during Asiacrypt 1991 [1] on the relationship between cryptography and machine learning, Rivest discussed for the first time how each field has contributed ideas and techniques to the other. In 2017, a short survey on automated design and cryptanalysis of cipher systems was given [3], showing that computational intelligence methods, such as genetic algorithms, genetic programming, Tabu search, and memetic

computing, are effective tools to solve many cryptographic problems. In his master thesis [4] (2018), Lagerhjelm presented a series of experiments using neural networks to try to decrypt some data encrypted with DES algorithm. In 2020, Baek and Kim [2] published a survey categorizing the several possible uses of Deep Learning techniques in the area of block cipher cryptanalysis. They identified two major categories: *neural cryptanalysis* and *neuro-aided cryptanalysis*. In neural cryptanalysis, a block cipher is seen as a black box, and different types of networks are designed depending on three main different targets: key recovery, cipher emulation, or cipher identification. The survey reviews the main works in neural cryptanalysis and concludes that neural networks can only be used to effectively attack toy or reduced round ciphers. In neuro-aided cryptanalysis, conventional cryptanalysis techniques are improved by the adoption of neural network. A special and successfull branch of neuro-aided cryptanalysis is *neuro-aided side channel attacks*, which falls out of the scope of both this work and [2].

2. Preliminaries on machine learning

The field of machine learning is concerned with the *study of computer algorithms that improve automatically through experience* [5]. Experience is provided in the form of training data and improvement is measured in terms of how well the algorithm can generalize the experience. In other words, improvement corresponds to a better algorithm performance, as a previously unknown sample is submitted to the algorithm.

Depending on the nature of the experience by which the algorithm learns, machine learning is divided into different learning types: *i)* supervised (learn a general rule by being presented example inputs and outputs), *ii)* unsupervised (find structure in an unlabeled input) or *iii)* reinforcement learning (achieve a goal by receiving rewards and learn from mistakes). Different statistical, probabilistic or optimization techniques have been used as *machine learning techniques*. Among them linear and logistic regression, artificial neural networks (ANNs), k-nearest neighbor (kNN), decision trees, random forests, support vector machines (SVMs) and Naive Bayes.

In the following we give a short introduction of machine learning techniques based on artificial neural networks which have found application in theoretical cryptanalysis.

2.1. Artificial Neural Networks

Artificial neural networks are used for image classification (e.g. Google Images), speech recognition (e.g. Siri), recommender systems (e.g. YouTube) or to win Go against the world champion (AlphaGo). A book which covers the mathematical background in greater detail is [6] while a practically oriented introduction is given in [7].

All artificial neural networks feature *artificial neurons* organized in layers as their elementary building blocks. In its minimum configuration an ANN contains an *input* and an *output layer*. At the input layer the unlabeled data is presented to the network, while at the output layer each neuron represents a possible outcome. Ad-

ditionally to the input and output layers the ANN usually contains a stack of *hidden layers*. A *deep* neural network is a special kind of ANN, in which the number of hidden layers is especially high.

The computational model of an artificial neuron has been inspired by animal brains and was proposed in 1943 by McCulloch and Pitts [8]. Each neuron in the ANN's hidden and output layers has trainable parameters: on one hand the connection weights to other neurons and on the other hand a bias term b.

The basic working principle of the ANN is that it approximates an arbitrary continuous function by utilizing its large parameter space. The parameter space of the neural network is determined by hyperparameters which relate to the design of the ANN (e.g. number of neurons, number of layers, ...) and network parameters (weights and biases of the neurons) obtained from the learning process on the training data. For continuous functions it has been shown in 1989 by Cybenko [9] that any continuous function of n real variables can be approximated to any desired precision by a neural network with as few as one hidden layer of sufficient size and a sigmoidal nonlinearity. The result was extended by Hornik et al. in 1989 to a more general form w.r.t. the used activation functions [10, 11], which is known today as *The Universal Approximation Theorem* of neural networks.

Neural Network Training

During the training phase of neural networks, first, the parameters of the neural network are initialized to a starting value. Then the parameters are refined in learning *epochs*: in each epoch the input of the neural network is the complete training data set. The training dataset consists of an input for the ANN and the corresponding labels y. Usually, the training dataset is not presented all at once but in *batches*.

During the learning process in a supervised setting a *loss function* $\mathcal{L}(z,y)$ is used to quantify the difference of the neural network output z to the labeled training dataset y. During the backpropagation [12] the contribution of each network parameter to the overall loss is evaluated. Afterwards the network parameters are updated to minimize the loss by gradient descent and the next batch is presented at the input.

The hyperparameter with largest influence on the gradient descent is the *learning rate*. It determines the step size with which the network parameters are updated during gradient descent. A popular policy to choose the learning rate follows the *cyclic learning rate* approach, proposed by Smith in 2015 [13]. The idea is to escape local minima and saddle points with a high learning rate and still be able to descend into lower loss areas without divergence.

The regular gradient descent optimizer can be substituted by more sophisticated choices: momentum techniques like momentum optimization [14] or Nesterov Accelerated Gradient (NAG) [15] take previous gradients into account and allow for faster propagation. They use the accumulated gradient for acceleration and a friction parameter to allow for stopping in a found minimum. Adaptive learning rate techniques like AdaGrad [16] (adaptive gradient descent) and RMSProp adapt not the momentum, but the learning rate to allow for faster propagation towards the optimum. RMSProp adds a decay parameter to accumulate only the most recent gradients. Adam [17] stands for adaptive moment estimation and combines the advantages of momentum optimization and RMSProp. Nadam [18] additionally combines the NAG method with Adam.

Activation Functions and Initialization
Neurons compute the weighted sum of their inputs plus one bias term b for each input and obtains $\vec{z} = \vec{x}^T \vec{w} + \vec{b}$. Afterwards an activation function $\phi(z)$ is applied and the signal propagates to the neurons in the subsequent layer. Since the backpropagation is based on gradient descent, the activation function has to *provide a gradient*. Historically, step functions or sigmoids have been used, that may lead to problems like *vanishing* or *exploding gradients*.

In 2010 it has been found that this problem relates to weight initialization as well as the use of the then popular sigmoid activation function [19]. The sigmoid function saturates, i.e. it has *zero gradient for* $z \gg 0$ for large absolute values of z and therefore doesn't provide a large enough gradient for backpropagation. Since [19] the ReLU activation function and other weight initialization schemes (like Glorot after the author of [19]) are more commonly used. An improvement to circumvent *dead neurons* has been proposed with the *LeakyReLU* variant [20], which provides a *nonzero gradient for* $z \ll 0$.

Convolutional Neural Networks (CNNs)
The most important building block of a CNN is the *convolutional layer*. Each neuron in the convolutional layer is only connected to neurons located within a small rectangle, or field of view, in the previous layer. A 2D mask is applied to the field of view of the convolutional-layer-neuron. The 2D mask acts as a filter which performs a 2D convolution. The field of view is determined by the size of the convolution kernel (*kernel size*), or, equivalently, by the size of the filter. Depending on the applied filter the neuron becomes more sensitive to certain patterns in the previous layer. Since many such patterns may exist, there is a range of predefined filters available and each of the filters will produce its own 2D layer, called a *feature map*. Hence, the convolutional layer is actually a 3D object with a depth corresponding to the number of chosen filters. However, all neurons within one feature map share the same weights and bias term, which leads to drastically less parameters than in a fully connected network.

In some works, e.g. Baksi et al. [21], it is shown that CNNs are not suitable for the purpose of finding a cryptographic distinguisher. This because CNNs are aimed at recognizing patterns in input data, which helps in image recognition or natural language processing, but does not work for cipher input where the bits are not related in any way.

Recurrent Neural Networks (RNNs)
A recurrent neural network (RNN) is a neural network with an active data memory. It is applied to a sequence to guess the next step in the sequence. In contrast to a feedforward neural network, where the activations flow only in one direction from input to the output layer, the recurrent neural network also contains connections backwards. Therefore a recurrent neuron receives inputs from the previous layer, but also its own output from one or more previous timesteps. Since the neuron's output depends on its state in previous timesteps it retains a *memory*. The weight of the connection of the recurrent input becomes another network parameter to be adjusted during training.

RNNs are trained similarly to feedforward ANNs, however, because of the additional time dynamic they first need to be "unrolled through time" and the training strategy is called *backpropagation through time* (BPTT). The "unrolling" actually creates a deep network which may have more error back-flow difficulties with vanishing and exploding gradients. ReLU activations can lead to even larger instability, which is why the standard activations in an RNN are saturating functions like tanh. A popular remedy is the usage of *Long Short-Term Memory* (LSTM) cells in the network. LSTMs were proposed by Hochreiter and Schmidhuber in 1997 [22] and as the name suggests, these cells keep track of the long term memory in a way compatible with BPTT. A simplified version of the LSTM cells are the *Gated Recurrent Units* (GRU) proposed by Cho et al. in 2014 [23].

As noted by Baksi et al. [21], LSTMs perform better than CNNs for cryptographic distinguishers, but worse than fine-tuned MLP. The main drawback of LSTMs seems the training speed.

2.2. Complexity of training neural networks

The complexity of training a neural network is determined by (see e.g. [24]): *i) the expressiveness* of the neural network, i.e. which prediction rules can be theoretically learned by a given network architecture, *ii) the sample or data complexity*, i.e., how many examples of a certain class are required and *iii) the training time or complexity*, i.e., the computation time required to learn a certain class.

To generally quantify the expressive power of a classifier the Vapnik-Chervonenkis (VC) dimension [25] may be used. However, the problem of successfully training a network with sufficient expressive power can still be NP-hard [24], resulting in practically unmanageable training times. Attempts to alleviate the training difficulty include to allow for *improper* training, where the found solution is not the optimal one and to over-specify the network, i.e. making the network larger than needed [24].

The question of the needed data complexity is an active field of research. It is not easy to exactly determine how many samples of a given class will be needed to learn it with a certain accuracy. For example Wang et al. [26] demonstrated the concept of *dataset distillation* in 2020: Instead of training a neural network on 60,000 MNIST images of handwritten digits, they "distilled" the whole dataset into just 10 distilled images, only one single image per class. In another work from 2020 [27] 'Less Than One'-shot learning is proposed, i.e., a method in which N classes are learned by the usage of $M < N$ examples.

Assuming the expressive power of the neural network is large enough and training data of sufficient size and quality is available, the *training complexity* is closely related to the number of trainable parameters of the neural network and the number of multiply-accumulate operations needed to pass an input through the network, see e.g. [28]. The training time needed to perform these operations will depend on the hardware configuration (e.g. GPU vs CPU) and on the possibility of parallelization for the network architecture.

3. Neural black-box cryptanalysis

By *black-box neural cryptanalysis* (or direct attacks with no prior information), we mean attacks that can be performed to any cipher, regardless of the cipher structure, except the input/output/key size. The aims of these attacks mainly are

- *Cipher identification:* distinguishing the output of the cipher from the output of another cipher, or distinguishing the output of the cipher from a random string;
- *Cipher simulation:* simulating the behaviour of a cipher;
- *Key recovery:* finding the key of the cipher.

Of course, an attacker able to perform key recovery can also perform cipher simulation, and being able to perform cipher simulation implies being able to perform cipher identification.

Usually, these kind of attacks are performed in the known plaintext scenario, so the attacker is given access to an oracle that can provide plaintext-ciphertext pairs encrypted under a certain key only known by the oracle. Furthermore, the attack is repeated for several keys, and in the case of key recovery, a new key (different from the ones used in the training) needs to be predicted.

Cipher identification (full ciphers)
Neural networks can be used to distinguish the output of a cipher from random bit strings or from the output of another cipher, by training the network with pair of plaintext, ciphertext, obtained from a single key (*single known-key distinguisher* [29] or from multiple keys (*single secret-key distinguisher*). Variations of this attack might exist in the *related key scenario*, but we are not aware of any work in this direction related to neural networks.

A direct application of ML to distinguishing the output produced by modern ciphers operating in a reasonably secure mode such as CBC was explored in [30]. The ML distinguisher did not have prior information on the cipher structure, and the authors conclude that their method was not successful in the task of extracting useful information from the ciphertexts when CBC mode was used and not even distinguish them from random data. Better results were obtained in ECB mode, as one may easily expect, due to the lack of semantic security (non-randomization) of the mode. The main tools used in the experiment are Linear Classifiers and Support Vector Machine with Gaussian Kernel. To solve the problem of cipher identification, the authors focused on the bag-of-words model of feature and the common classification framework previously used in [31,32], where the features of the input examples are mostly related to the varying length words. In [30], the features that are considered are the entropy, the number of symbols appearing in the ciphertext, 16-bit histograms with 65536 dimensions, the varying length words proposed in [31].

Similar experiments to the one of [30] have also been presented, essentially, with similar results. For example, in [33], the authors consider 8 different plaintext languages, 6 block ciphers (DES, Blowfish, ARC4, Rijndael, Serpent and Twofish) in

ECB and CBC mode and a "CBC"-like variation of RSA, and perform the identification on a more performing machine (40 computational nodes, each with a 16-core Opteron 6276 CPU, a NVIDIA Tesla K20 GPU and 32GB of central memory) compared to [30], by means of different classical machine learning classifiers: C4.5, PART, FT, Complement Naive Bayes, Multilayer Perceptron and WiSARD. The NIST test suite was applied to the ciphertexts to guarantee the quality of the encryption. The authors conclude that the influence of the idiom in which plain texts were written is not relevant to identify different encryption. Also, the proposed procedures obtained full identification for almost all of the selected cryptographic algorithms in ECB mode. The most surprising result reported by the author is the identification of algorithms in CBC mode, which showed lower rates than the ECB case, but, according to the authors, "not insignificant", because "greater than the probabilistic bid". Moreover, the authors pointed out that rates increased monotonically, and thus can be increased by intensive computation. The most efficient classifier was Complement Naive Bayes, not only with regard to successful identification, but also in time consumption.

Another recent work is the master thesis of Lagerhjelm [4], in 2018. In this work, long short-term memory networks are used to (unsuccessfully) decrypt encrypted text, and convolutional neural network to perform classification tasks on encrypted MNIST images, again with success when distinguishing the ECB mode, and with no success in the CBC case.

Cipher simulation (reduced/small/full ciphers)

Neural networks can be used to simulate the behaviour of a cipher, by training the network with pairs of plaintext and ciphertext generated from the same key. Without knowing the secret key, one could either aim at predicting the ciphertext given a plaintext (encryption simulation), as done, for example, by Xiao et al. in [34], or to predict a plaintext given a ciphertext (decryption simulation), as done, for example, by Alani in [35, 36].

In 2012, Alani [35, 36] implemented a known-plaintext attack based on neural networks, by training a neural network to retrieve plaintext from ciphertext without retrieving the key used in encryption, or, in other words, finding a functionally equivalent decryption function. The author claimed to be able to use an average of 211 plaintext-ciphertext pairs to perform cryptanalysis of DES in an average duration of 51 minutes, and an average of only 212 plaintext-ciphertext pairs for Triple-DES in an average duration of 72 minutes. These results, though, could not be reproduced by, for example, Xiao et al. [34], and no source code was provided to reproduce the attack. The adopted network layouts were 4 or 5 layers perceptrons, with different configurations. The average size of data sets used was about 2^{20} plaintext-ciphertext pairs. The training algorithm was the scaled conjugate-gradient. The experiment, implemented in MATLAB, was run on single computer with AMD Athlon X2 processor with 1.9 Gigahertz clock frequency and 4 Gigabytes of memory.

In 2019, Xiao et al. [34] try to determine the output of a cipher treating it as a black box using an unknown key. Their method was based on training a neural network. The error function chosen to correct the weights during the training was mean-squared error. Weights were initialized randomly. The maximum numbers of training cycles

(epochs) was set to 10^4. Then, the measure of the strength of a cipher was given by three metrics: cipher match rate, training data, and time complexity.

They performed their experiment on reduced-round DES and Hitaj2 [37], a 48-bit key and 48-bit state stream cipher, developed and introduced in late 90's by Philips Semiconductors (currently NXP), primarily used in Radio Frequency Identification (RFID) applications, such as car immobilizers.

Note that Hitaj2 has been attacked several times with algebraic attacks using SAT solvers (e.g. [38, 39]) or by exhaustive search (e.g. [40, 41]).

Xiao et al. tested three different networks: a deep and thin fully connected network (MLP with 4 layers of 128 neurons each), a shallow and fat network (MLP with 1 layer of 1000 neurons), and a cascade network (4 layers with 128, 256, 256, 128 neurons). All three networks end with a softmax binary classifier. Their experiments showed that the most powerful attack based on neural networks varies from cipher to cipher. While a fat and shallow shaped fully connected network is the best to attack the round-reduced DES (up to 2 rounds), a deep-and thin shaped fully connected network works best on Hitag2.

Three common activation functions, sigmoid, tanh and rectified linear unit (ReLU), were tested, only for the shallow-fat network. The authors concluded that the sigmoid function allows a faster training, though all functions eventually reach the same accuracy.

Training and testing were performed on a personal laptop (no details provided), so the network used cannot be too large. The training has been performed with up to 2^{30} samples.

In 2022 Bellini, et al. [42] showed some insights on why, in general, black-box cipher simulation via neural networks gives little hope of success on modern symmetric ciphers. The main idea is to represent a block cipher using a set of boolean functions, and then prove that learning boolean functions is hard for neural networks. This was supported both by practical experiments on random boolean functions and theoretical calculations to estimate the resources needed to emulate round-reduced versions of AES, a task that has never been done successfully due to its complexity.

The authors also provided a review of the work of Xiao et al. [34], using their framework. Results on both DES and Hitag2 were confirmed to be plausible. On the other hand, results in [35, 36], were considered unlikely to be reproducible by the authors, confirming Xiao et al.'s thesis.

Key recovery attacks (simplified ciphers)

Neural networks can be used to predict the key of a cipher, by training the network with triples of plaintext, ciphertext and key (different from the one that needs to be found).

In 2014, Danziger and Henriques [43] successfully mapped the input/output behaviour of the Simplified Data Encryption Standard (S-DES) [44], a simplified version of DES with 10 bit keys and 8 bit messages, with the use of Single Hidden Layer Perceptron (MLP) neural network. They also showed that the effectiveness of the MLP network depends on the nonlinearity of the internal s-boxes of S-DES. Indeed, the main goal of the authors was to understand the relation between the differential cryptanaly-

sis results and the ones obtained with the neural network. In their experiment, given the plaintext P and ciphertext C, the output layer of the neural network is used to predict the key K. Thus, for the training of the weights and biases in the neural network, training data of the form (P,C,K) are needed. After training has finished, the neural network was expected to predict a new value of K (not appearing in the training phase) given a new (P,C) pair as input.

Prior works on S-DES include [45, 46], where Alallayah et al. proposed the use of Levenberg-Marquardt algorithm rather than the Gradient Descent to speed up the training. Beside key recovery, they also used a single layer perceptron network to emulate the behaviour of S-DES, modelling the network with the plaintext as input, and the ciphertext as output. Their results were positive due to the small size of the cipher, and a thorough analysis of the techniques used is lacking.

In 2020, So [47] proposed the use of 3 to 7 Layer Perceptron Neural Networks to perform a known plaintext key recovery attack on S-DES (8 bit block, 10 bit key, 2 rounds), Simon32/64 (32 bit block, 64 bit key, 32 rounds), and Speck32/64 (32 bit block, 64 bit key, 22 rounds). Besides considering random keys, So additionally restricted keys to be made of ASCII characters. In this second case, the NN was able to recover keys for all the non-reduced ciphers. It is important to notice that the largest cipher analyzed by So has a key space of 2^{64} keys, which is reduced to $2^{48} = 64^8$ keys when only ASCII keys are considered. The number of hidden layers adopted in this work ranges between 3,5,7, while the number of neurons per layer ranges between 128, 256, 512. In the training phase, So used 5000 epochs and the Adam adaptive moment algorithm for the learning rate optimization of the NN. The training and testing were run on GPU-based server with Nvidia GeForce RTX 2080 Ti and its CPU was Intel Core i9-9900K.

Key-schedule inversion

As for the simulation of cipher decryption described in section 3, one might try to invert the behavior of the key schedule routine, as done for example by Pareek et al. [48], in 2020. In their work, they considered the key schedule of PRESENT and tried to retrieve the 80-bit key from the last 64-bit round key, using a MLP network with 3 hidden layers of 32, 16, and 8 neurons. Unfortunately, the authors concluded that, using this type of network, the accuracy of predicting the key bits, were not significantly deviating from 0.5.

4. Neuro-aided cryptanalysis

By *neuro-aided cryptanalysis* we refer to methods that improve conventional cryptanalysis techniques by means of neural networks. In the recent and existing literature covering this case, neural networks are used only to provide more effective and efficient distinguishers, that can be used later to perform key recovery attacks using conventional techniques.

Works in this direction focus on extending the commonly used model of differential distinguisher by using ML techniques. In the case of differential distinguisher,

the attacker Eve XORs a chosen input difference δ to the input of the state of the (reduced round) cipher and watches for a particular output difference Δ, with randomly chosen inputs. If the (δ, Δ) pair occurs with a probability significantly higher for the (reduced round) cipher than what it should be for a random case, the (reduced round) cipher can be distinguished from the random case. In conventional cryptanalysis, this probability distribution of $\delta \mapsto \Delta$ is modeled by the *differential branch number* [49] or by automated tools like Mixed Integer Linear Programming (MILP) [50]. The modeling for differential distinguisher can be extended by incorporating machine learning algorithms.

In 2019, the work by Gohr [51] was the first that compared cryptanalysis performed by a deep neural network to solving the same problems with strong, well-understood conventional cryptanalytic tools. It was also the first paper to combine neural networks with strong conventional cryptanalysis techniques and the first paper to show a neural network based attack on a symmetric cryptographic primitive, improving upon the published state of the art. All this is applied to Speck32/64, a lightweight cipher designed by the NSA, with a 32 bit block input and a 64 bit key. Other similar works appeared following Gohr's approach.

In order to transform the differential distinguisher to a classification problem, Baksi et al. [21] proposed two models. In the first model, the attacker chooses t input differences. In the training (offline) phase, the attacker tries to learn whether there is any pattern in the cipher outputs that the machine learning tool is capable of finding. To test that, Eve creates t differentials with those input differences and feeds all the data to the machine learning. If the accuracy of ML training is higher than what it should be for random data (i.e., $1/t$), the attacker is able to extract pattern from the cipher outputs and proceeds to the online phase. Otherwise (if the training accuracy is $1/t$), the process is aborted. While the first model can work with an arbitrary number, $t \geq 2$, of input differences, in the second model the authors propose a different model that can work with only one input difference.

Baksi et al. applied the first model to round-reduced Gimli-Hash and Gimli-Cipher (8 round distinguisher), Ascon 320-bit permutation (3 round distinguisher), and Knot-256 (10 round distinguisher) and Knot-512 (12 round distinguisher). They also showed the effectiveness of the second model over the lightweight MAC Chaskey. In general, they were able to reduce the online data complexity of the conventional distinguisher, up to the cube root in the case of Gimli (from 2^{-52} to $2^{-14.3}$), at the cost of processing offline data for the training phase ($2^{-17.6}$). The authors also investigated effects of choosing different neural network architectures with respect to 8-round GIMLI-PERMUTATION as the target cipher, concluding that MLP networks are the most performing, followed by LSTM. On the other hand CNN seems not suited for the task of building a crytpographic distinguisher. The tested networks, use from 3 to 6 layers, and up to 1024 neurons per layer, for a total number of paramenters that ranges from $90,818$ to $2,249,858$. They used ReLU and LeakyRELU for MLP and CNN, and tanh and sigmoid for LSTM.

Jain et al. [52] adopted the same approach of Baksi et al. to analyze 3-6 round reduced PRESENT lightweight block cipher. They used 2 MLP networks with three

hidden layers of 128, 1024, and 1024 neurons (the most successful configuration in Baksi), and two 2-hidden layer MLP networks, which seemed to produce the same results in less time and better chances of avoiding data over-fitting. They used batch size of 200, 25 epochs, samples of 10000, Adam optimizer, MSE loss function, and learning rate of 0.001. The authors also showed, as one might expect, that randomly generated input differences generate a worse distinguisher than using differences found by means of conventional cryptanalysis.

Again, following Gohr and Baksi et al., Yadav and Kumar [53] obtained a multi-layer perceptron distinguisher for 12 rounds SIMON, 9 rounds SPECK, and 8 rounds GIFT. In their work they also propose to extend a classical distinguisher with a neural one. They used two hidden layers having 1024 neurons each.

Still on the line of Gohr's idea, Bellini and Rossi [54] made a framework to compare the performance of blackbox and differential conventional distinguishers, and neuro-aided distinguisher exploiting the knowledge of differential trails found by means of conventional cryptanalysis. They used TEA and RAIDEN ciphers as a case study, and showed the superiority of the neural distinguisher. In their experiment, they use a MLP and CNN of 3 to 5 hidden layers, with only 32 and 64 neurons. All previous deep learning works focus on xor-differential cryptanalysis, while in [54] the authors analyze two ciphers based on additive-differential cryptanalysis. One peculiarity of their approach is the use of an MLP that is divided in two parts, a "time distributed" network and a fully connected MLP. The idea of splitting the network in two parts comes from the fact that in both ciphers (Feistel structure) the output is calculated separately as two different words (although not independently), so they want the network to see these words alone. The expected effects for the network is to emphasize the key features of each word independently from the others. They also tried to use only the fully connected part of the network (without the time distributed one) getting worse results on the accuracy metric.

In [55] the authors proposed an improvement of Gohr's work called *Neural Aided Statistical Attack* (NASA). The main difference with Gohr's work is the lack of dependency from the quantity of neutral bits in a cipher, allowing them to attack more ciphers. This improvement was obtained via statistical techniques on the processed data, while the neural network part is basically the same as Gohr's. Their main results were an attack on a 10-round recuced version of DES, and a 13-round reduced version of SPECK32/64. Their results are further improved in [56].

At EUROCRYPT'21, Benamira et al. [57] analyzed Gohr's work to try to give an explanation of why the new distinguisher is outperforming classical cryptanalysis. They found out that Gohr's distinguisher relies not only on the distribution of differences in the ciphertext pairs, but also on their distribution in the two previous rounds. Moreover, they analyzed Gohr's network architecture and found out that it is possible to reach the same accuracy with a simpler model, simplifying the prediction head (the final MLP block in Gohr's network) and the first convolutional layer. Another work in this direction has been done by Chen et al. [58], introducing the *Extended Differential-Linear Connectivity Table*, a tool from which they claim to be able to extract features that are learnable by neural distinguishers, and so to be able to explain better how are

they learning. However, at the moment of writing this survey, the work from Chen et al. is still a preprint, so no peer-review of their methods has been given.

At ASIACRYPT'22, Bao et al. [59] extended again Gohr's work, studying the key-recovery strategies associated with neural network. Moreover, as already did in [57], they study the dependence from the differences in previous rounds to exploit new input formats for the neural network. They also introduce a new network architecture, based on *Squeeze-and-Excitation networks* (SENet) [60], and show that it outperforms Gohr's residual network.

Hou et al. [61] used SAT solvers to enhance the performances of neural distinguishers and extend their use to larger-state block ciphers. Using a technique similar to Bao's and Gohr's they perform key-recovery on 13-round SIMON32/64, 14-round SIMON48/96 and 13-round SIMON64/128.

Zhang et al. [62] proposed an improvement of Gohr's pipeline by training the distinguisher for more rounds. Their strategy is based on using multiple parallel convolutional layers with different kernel sizes in the convolutional block, and then give the output to the residual part of Gohr's network, taking big inspiration from GoogLeNet [63]. They obtain a 9-round differential-neural distinguisher for SPECK32/64 and a 12-rounds one for SIMON32/64. Moreover, they reach a key-recovery attack on 17-rounds SIMON32/64 for the first time. In a subsequent work [64], a subset of the authors extend these results to DES, Chaskey and PRESENT, altough not being able to mount a complete key recovery attack.

Bellini et al. [65] created a framework to automatically perform neural cryptanalysis of ciphers, independently of their size. The framework is composed by two main parts: as a first component, an evolutionary algorithm for the search of input differences that are good for Gohr's-like neural distinguisher; this algorithm enables the search for larger ciphers than the original Gohr's optimizer, fixing its scaling issues. The second component is a neural distinguisher architecture called DBitNet, that is independent from the structure of the cipher. This architecture is based on dilated convolutions [66], to tackle the problem of investigating both near and long-range dependencies between the network inputs. The authors obtain improvements on the state-of-the-art neural distinguishers for SPECK64, SPECK128, SIMON64, SIMON128, GIMLI-PERMUTATION and analyze for the first time in the neural cryptanalysis settings the ciphers HIGHT, LEA, TEA, XTEA and PRESENT.

In 2022 Chen et al. [67] proposed a multi-stage key-recovery framework to perform the key-recovery task on large state block ciphers. The authors claim to be able to combine classical distinguishers with neural ones and a Gohr-like key-recovery attack to obrain the key for 19-rounds SPECK128 and 13-rounds SPECK64. However, the complexity of their attacks is too high to be executed and tested, so these attacks remain theoretical.

5. Conclusions

In this work, we gave a short overview of the literature regarding the application of machine learning techniques to modern symmetric ciphers. We found out that most of the applications in the black-box settings are not suitable to attack real world modern symmetric ciphers, while some of the recent works on neuro-aided cryptanalysis seem to be promising on round-reduced versions of them.

References

[1] R. L. Rivest. Cryptography and machine learning. In Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto, editors, *Advances in Cryptology — ASIACRYPT '91*, pages 427–439, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

[2] Seunggeun Baek and Kwangjo Kim. Recent advances of neural attacks against block ciphers. In *2020 Symposium on Cryptography and Information Security (SCIS 2020)*. IEICE Technical Committee on Information Security, 2020.

[3] Wasan Awad and El-Sayed M El-Alfy. Computational intelligence in cryptology. In *Artificial Intelligence: Concepts, Methodologies, Tools, and Applications*, pages 1636–1652. IGI Global, 2017.

[4] L. Lagerhjelm. Extracting information from encrypted data using deep neural networks, 2018. Available at: `http://urn.kb.se/resolve?urn=urn:nbn:se:umu:diva-155904`.

[5] T.M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*, volume 19. The MIT Press, 2017.

[7] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras and TensorFlow: concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, 2019.

[8] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133, dec 1943.

[9] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4):303–314, dec 1989.

[10] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 1989. `https://doi.org/10.1016/0893-6080(89)90020-8`.

[11] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 1991. https://doi.org/10.1016/0893-6080(91)90009-T.

[12] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[13] L. N. Smith. No more pesky learning rate guessing games. *CoRR*, abs/1506.01186, 2015. Available at: `https://arxiv.org/pdf/1506.01186v1.pdf`.

[14] B. T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 1964.

[15] Y. Nesterov. A method for solving the convex programming problem with convergence rate $O\left(\frac{1}{k^2}\right)$. In *Dokl. Akad. Nauk SSSR*, volume 269, pages 543–547, 1983.

[16] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. In *COLT 2010 - The 23rd Conference on Learning Theory*, 2010.

[17] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 2015.

[18] Timothy Dozat. Incorporating Nesterov Momentum into Adam. *ICLR Workshop*, 1(1):2013–2016, 2016.

[19] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Journal of Machine Learning Research*, volume 9, pages 249–256, 2010.

[20] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical Evaluation of Rectified Activations in Convolutional Network. *CoRR*, abs/1505.0, may 2015.

[21] Anubhab Baksi, Jakub Breier, Xiaoyang Dong, and Chen Yi. Machine learning assisted differential distinguishers for lightweight ciphers, 2020. Available at: `https://eprint.iacr.org/2020/571.pdf`.

[22] Sepp Hochreiter and J Urgen Schmidhuber. Long Shortterm Memory. *Neural Computation*, 9(8):17351780, 1997.

[23] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, pages 1724–1734, 2014.

[24] Roi Livni, Shai Shalev-Shwartz, and Ohad Shamir. On the computational efficiency of symmetric neural networks. *Advances in neural information processing systems*, 27:855—-863, 2014.

[25] M. Anthony. *Neural Networks and Boolean Functions*, page 554–576. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2010. `https://doi.org/10.1017/CBO9780511780448.01`.

[26] Tongzhou Wang, Jun-Yan Zhu, Antonio Torralba, and Alexei A. Efros. Dataset Distillation. pages 1–14, nov 2018. Available at: `http://arxiv.org/abs/1811.10959`.

[27] Ilia Sucholutsky and Matthias Schonlau. 'less than one'-shot learning: Learning n classes from $m < n$ samples. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(11):9739–9746, May 2021.

[28] Timothy Foldy-Porto, Yeshwanth Venkatesha, and Priyadarshini Panda. Activation Density driven Energy-Efficient Pruning in Training. *arXiv*, feb 2020.

[29] Lars R Knudsen and Vincent Rijmen. Known-key distinguishers for some block ciphers. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 315–324. Springer, 2007.

[30] Jung-Wei Chou, Shou-De Lin, and Chen-Mou Cheng. On the effectiveness of using state-of-the-art machine learning techniques to launch cryptographic distinguishing attacks. In *Proceedings of the 5th ACM Workshop on Security and Artificial Intelligence*, pages 105–110, 2012.

[31] Aroor Dinesh Dileep and Chellu Chandra Sekhar. Identification of block ciphers using support vector machines. In *The 2006 IEEE International Joint Conference on Neural Network Proceedings*, pages 2696–2701. IEEE, 2006.

[32] Sammireddy Swapna, AD Dileep, C Chandra Sekhar, and Shri Kant. Block cipher identification using support vector classification and regression. *Journal of Discrete Mathematical Sciences and Cryptography*, 13(4):305–318, 2010.

[33] Flávio Luis de Mello and José AM Xexéo. Identifying encryption algorithms in ECB and CBC modes using computational intelligence. *J. UCS*, 24(1):25–42, 2018.

[34] Ya Xiao, Qingying Hao, and Danfeng Daphne Yao. Neural cryptanalysis: Metrics, methodology, and applications in CPS ciphers. In *2019 IEEE Conference on Dependable and Secure Computing (DSC)*, pages 1–8. IEEE, 2019.

[35] Mohammed M Alani. Neuro-cryptanalysis of des. In *World Congress on Internet Security (WorldCIS-2012)*, pages 23–27. IEEE, 2012.

[36] Mohammed M Alani. Neuro-cryptanalysis of DES and Triple-DES. In *International Conference on Neural Information Processing*, pages 637–646. Springer, 2012.

[37] Sean O'Neil and Nicolas Courtois. Reverse-engineered Philips/NXP Hitag2 Cipher, 2008. Available at: `http://fse2008rump.cr.yp.to/00564f75b2f39604dc204d838da01e7a.pdf`.

[38] Henryk Plötz and Karsten Nohl. Breaking Hitag2. *HAR2009*, 2011, 2009.

[39] Nicolas T Courtois, Sean O'Neil, and Jean-Jacques Quisquater. Practical algebraic attacks on the Hitag2 stream cipher. In *International Conference on Information Security*, pages 167–176. Springer, 2009.

[40] Petr Štembera and Martin Novotny. Breaking Hitag2 with reconfigurable hardware. In *2011 14th Euromicro Conference on Digital System Design*, pages 558–563. IEEE, 2011.

[41] Vincent Immler. Breaking Hitag2 revisited. In *International Conference on Security, Privacy, and Applied Cryptography Engineering*, pages 126–143. Springer, 2012.

[42] Emanuele Bellini, Anna Hambitzer, Matteo Protopapa, and Matteo Rossi. Limitations of the use of neural networks in black box cryptanalysis. In Peter Y.A. Ryan and Cristian Toma, editors, *Innovative Security Solutions for Information Technology and Communications*, pages 100–124, Cham, 2022. Springer International Publishing.

[43] Moisés Danziger and Marco Aurélio Amaral Henriques. Improved cryptanalysis combining differential and artificial neural network schemes. In *2014 International Telecommunications Symposium (ITS)*, pages 1–5. IEEE, 2014.

[44] Edward F Schaefer. A simplified data encryption standard algorithm. *Cryptologia*, 20(1):77–84, 1996.

[45] Khaled M Alallayah, Waiel FA El-Wahed, Mohamed Amin, and Alaa H Alhamami. Attack of against simplified data encryption standard cipher system using neural networks. *Journal of Computer Science*, 6(1):29, 2010.

[46] Khaled M Alallayah, Alaa H Alhamami, Waiel AbdElwahed, and Mohamed Amin. Applying neural networks for simplified data encryption standard (sdes) cipher system cryptanalysis. *Int. Arab J. Inf. Technol.*, 9(2):163–169, 2012.

[47] Jaewoo So. Deep learning-based cryptanalysis of lightweight block ciphers. *Security and Communication Networks*, 2020, 2020. https://doi.org/10.1155/2020/3701067.

[48] Manan Pareek, Dr. Girish Mishra, and Varun Kohli. Deep learning based analysis of key scheduling algorithm of present cipher. Cryptology ePrint Archive, Report 2020/981, 2020. https://eprint.iacr.org/2020/981.

[49] Joan Daemen and Vincent Rijmen. *The design of Rijndael: AES-the advanced encryption standard*. Springer Science & Business Media, 2013.

[50] Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. Differential and linear cryptanalysis using mixed-integer linear programming. In *International Conference on Information Security and Cryptology*, pages 57–76. Springer, 2011.

[51] Aron Gohr. Improving attacks on round-reduced speck32/64 using deep learning. In *Advances in Cryptology – CRYPTO 2019*, pages 150–179. Springer, 2019.

[52] Aayush Jain, Varun Kohli, and Girish Mishra. Deep learning based differential distinguisher for lightweight cipher present, 2020. Available at: https://eprint.iacr.org/2020/846.pdf.

[53] Tarun Yadav and Manoj Kumar. Differential-ml distinguisher: Machine learning based generic extension for differential cryptanalysis, 2020. Available at: https://eprint.iacr.org/2020/913.pdf.

[54] Emanuele Bellini and Matteo Rossi. Performance comparison between deep learning-based and conventional cryptographic distinguishers. In Kohei Arai, editor, *Intelligent Computing*, pages 681–701, Cham, 2021. Springer International Publishing.

[55] Yi Chen, Yantian Shen, and Hongbo Yu. Neural-Aided Statistical Attack for Cryptanalysis. *The Computer Journal*, 07 2022. bxac099.

[56] Yi Chen and Hongbo Yu. Improved neural aided statistical attack for cryptanalysis. Cryptology ePrint Archive, Paper 2021/311, 2021. https://eprint.iacr.org/2021/311.

[57] Adrien Benamira, David Gerault, Thomas Peyrin, and Quan Quan Tan. A deeper look at machine learning-based cryptanalysis. page 805–835, Berlin, Heidelberg, 2021. Springer-Verlag.

[58] Yi Chen and Hongbo Yu. Bridging machine learning and cryptanalysis via edlct. Cryptology ePrint Archive, Paper 2021/705, 2021. https://eprint.iacr.org/2021/705.

[59] Zhenzhen Bao, Jian Guo, Meicheng Liu, Li Ma, and Yi Tu. Enhancing differential-neural cryptanalysis. In *Advances in Cryptology – ASIACRYPT 2022: 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5–9, 2022, Proceedings, Part I*, page 318–347, Berlin, Heidelberg, 2023. Springer-Verlag.

[60] Jie Hu, Li Shen, Samuel Albanie, Gang Sun, and Enhua Wu. Squeeze-and-excitation networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(8):2011–2023, 2020.

[61] ShaoZhen Chen ZeZhou Hou, JiongJiong Ren. Improve neural distinguishers of simon and speck. *Security and Communication Networks, Hindawi*, 2021.

[62] Liu Zhang, Zilong Wang, Baocang wang, and Boyang Wang. Improving differential-neural cryptanalysis with inception. Cryptology ePrint Archive, Paper 2022/183, 2022. `https://eprint.iacr.org/2022/183`.

[63] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

[64] Liu Zhang and Zilong Wang. Improving differential-neural distinguisher model for des, chaskey and present. Cryptology ePrint Archive, Paper 2022/457, 2022. `https://eprint.iacr.org/2022/457`.

[65] Emanuele Bellini, David Gerault, Anna Hambitzer, and Matteo Rossi. A cipher-agnostic neural training pipeline with automated finding of good input differences. Cryptology ePrint Archive, Paper 2022/1467, 2022. `https://eprint.iacr.org/2022/1467`.

[66] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. In *International Conference on Learning Representations (ICLR)*, May 2016.

[67] Chen Yi, Bao Zhenzhen, Shen Yantian, and Yu Hongbo. A deep learning aided key recovery framework for large-state block ciphers. *SCIENTIA SINICA Informationis*, pages –, 2022.

AMS Subject Classification: 94A60, 68T07, 68P25.

Emanuele BELLINI, Anna HAMBITZER
Technology Innovation Institute
Abu Dhabi, United Arab Emirates
e-mail: `name.surname@tii.ae`

Matteo ROSSI
Politecnico di Torino
Torino, Italy
e-mail: `matteo.rossi@polito.it`

Lavoro pervenuto in redazione il 15.04.2022.