Enhancing Privacy in Remote Data Classification

Claudio Orlandi[1]*, Alessandro Piva[2], Michele Caini[2], and Mauro Barni[3]

[1] Department of Computer Science, University of Aarhus;
orlandi@daimi.au.dk
[2] Department of Electronics and Telecommunications, University of Florence;
{piva,caini}@lci.det.unifi.it
[3] Department of Information Engineering, University of Siena;
barni@dii.unisi.it

Abstract. Neural networks are a fundamental tool in data classification since they represent a universal tool enabling a great variety of applications. A protocol whereby a user may ask a service provider to run a neural network on an input provided in encrypted format is proposed here, in such a way that the neural network owner does not get any knowledge about the processed data. At the same time, the knowledge embedded within the network itself is protected.
With respect to previous works in this field, the interaction between the user and the NN owner is kept to a minimum without resorting to general secure multi-party computation protocols.

Key words: Privacy-Preserving Computation, Neural Networks, Homomorphic Encryption, Signal Processing in Encrypted Domain, CryptoComputing

1 Introduction

Recent advances in signal and information processing together with the possibility of exchanging and transmitting data through flexible and ubiquitous transmission media such as internet and wireless networks, have opened the way towards a new kind of services whereby a provider sells its ability to process and interpret data remotely, e.g. through an internet web service. Examples in this sense include access to remote databases, processing of personal data, processing of multimedia documents, interpretation of medical data for remote diagnosis. In this last scenario, a patient may need a diagnosis from a remote medical institute that has the knowledge needed to perform the diagnosis. Health-related data are of course sensitive, and the patient may do not want to let the institute to know the data he owns; on the other hand, the medical institute is interested in protecting his expertise.

In 2000 two different papers proposed the notion of privacy preserving data mining, meaning the possibility to perform data analysis on a distributed database, under some privacy constraints. Lindell and Pinkas [LP00] presented a way to securely and efficiently compute a decision tree using cryptographic protocols; at the same time, Agrawal and Srikant [AS00] presented another solution to the same problem using data randomization.

After the publication of these papers, security constraints were added to several techniques from machine learninn, including: decision trees [LP00], neural networks [CL05], support vector machines [LLM06], naive bayes classifiers [KV03], belief networks [YW05, WY04], clustering [JKM05]. In all these works, we can identify two major scenarios: in the first one Alice and Bob share a dataset and want to extract knowledge from it without revealing their own data (privacy preserving data mining). In the other scenario, which is the one considered in this paper, Alice owns her private data x, while Bob owns an evaluation function C (in most cases C is a classifier). Alice is interested in having her data processed by Bob, but she does not want that Bob learns either her input or the output of the computation. At the same time Bob does

* Work done while working at University of Florence.

not want to reveal the exact form of C, representing his knowledge, since, for instance, he sells a classification service through the web (as in the remote medical diagnosis example).

A fundamental brick in data classification task is represented by neural networks (NNs), becayse of their approximation and generalization capabilities. For this reason, it can be of interest to design a protocol whereby a user may ask a service provider to run a neural network on an input provided in encrypted format. Previous works on privacy preserving NN computing are limited to the systems presented in [BOP06, CL05]. However, such studies resort extensively to highly inefficient general secure multi-party computation (SMC) [Yao86] for the computation of the non-linear activation functions implemented in the neurons.

This is not the case with our new protocol which does not resort to general SMC for the evaluation of the activation functions. In a nutshell, the protocol has been designed to ensure that the data provided by the user (say Alice), representing the input of the neural network are completely protected and, at the same time, to not disclose Bob's classifier (the NN). The proposed protocol relies on homomorphic encryption that allows to perform directly in the encrypted domain all the linear computations. For the non linear functions that can not be handled by means of homomorphic encryption, a limited amount of interaction between the NN owner and the user is introduced to delegate the user (say Alice) to perform some of the computation.

Comparing our work with previous ones, another advantage can be highlighted: the proposed protocol can handle every kind of feedforward NN (not only simple layered networks), without disclosing neither the number of neurons nor the way they are connected.

The rest of this paper is organized as follows. In Section 2, a brief overview on Neural Networks that can be used with our protocol is given. In Section 3 our scenario will be described, focusing on the privacy constraints and reviewing the properties to be achieved by our protocol. The way the protocol works is described in Section 4. Section 5 is devoted to the experimental results obtained developing a distributed application that runs the protocol. Some concluding remarks are given in Section 6, while in Appendix how to deal with non integer computation will be discussed.

2 Neural Networks

Neural networks have a great ability to model any given function [LF88, HSW89]. Moreover neural networks are provided with good learning algorithms, are robust against noise and generalize well on unseen examples.

In this section we will introduce several types of network that can be used with our protocol. The notation is consistent with the one in [Bis95], where a detailed treatment of neural networks functioning and applications is given.

2.1 Perceptron

The simplest neural network is the *perceptron* (Figure 1 (a)). It can discriminate between two different classes of instance, and its classification function consists of a linear combination of the input variables, the coefficients of which are the parameters of the model. The discriminant is of the form $a(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + w_0$ where $\boldsymbol{x}$ is the input vector, $\boldsymbol{w}$ is the vector of weights and w_0 is a threshold[4]. The instance $\boldsymbol{x}$ is assigned to class c_1 if $a(\boldsymbol{x}) \geq 0$ and to class c_2 if $a(\boldsymbol{x}) < 0$. This method can easily be extended to the multiclass case using one discriminant function $a_k(x)$ for each class C_k such that $a_k(\boldsymbol{x}) = \boldsymbol{w}_k^T\boldsymbol{x}$.

[4] From now on we can forget about w_0 simply appending it at the end of the vector and obtaining $a(\boldsymbol{x}) = [\boldsymbol{w}\ w_0]^T[\boldsymbol{x}\ 1]$

2.2 Feed-Forward Networks

To allow for more general classification we consider a network of interconnetted neurons.

Layered Networks. Networks consisting of successive layers of adaptive weights are called *layered networks*: in such a network every unit in one layer is connected to every unit in the next layer, but no other connections are permitted, as shown in Figure 1 (b) The units that are not treated as output units are called *hidden* units.

The output of the j-th hidden unit in the i-th layer is obtained by first forming a weighted linear combination of its l input values to give

$$a_j^{(i)} = \sum_{k=1}^{l} w_{jk}^{(i)} x_k$$

Here $w_{jk}^{(i)}$ denotes the weight associated to an edge going from the k-th neuron of the previous layer to the neuron j of i-th layer. The activation of the hidden unit is then obtained by transforming the linear sum using an activation function $g(\cdot)$ to obtain $z_j^{(i)} = g\left(a_j^{(i)}\right)$.

This procedure is iterated until the output layer is reached, obtaining the final output $y_k = g(a_k)$.

We will refer to a L-layer network as a network having L layers of adaptive weights, regardless of the input units.

General topologies. Since there is a direct correspondence between a network diagram and its mathematical function, we can develop more general network mappings by considering more complex network diagrams. To be able to run the computation, however, we shall restrict our attention to the case of *feed-forward* networks, in which there are no feed-back loops, as shown in Figure 1 (c).

Activation Function. The activation function of the hidden neurons are usually sigmoidal functions, i.e. $g(a) = \frac{1}{1+e^{-a}}$.

The differentiability of this kind of functions leads to a powerful and computationally efficient learning method, called *error backpropagation* [RHW86].

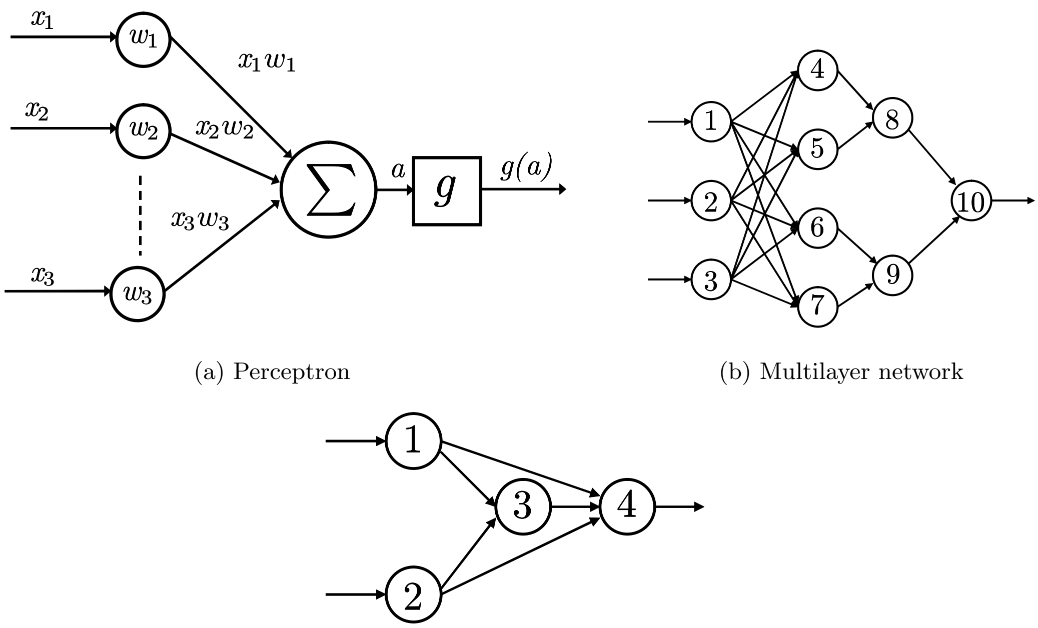
We allow output neurons to have both sigmoid activation functions or linear activation functions. The latter is sometimes desirable because the use of sigmoid units at the outputs would limit the range of possible outputs to the range attainable by the sigmoid.

3 Remote Data Classification

The scenario we are going to consider is the following: Alice owns a vector of data, while Bob owns a classifier. Alice is interested to classify her data by exploiting Bob knowledge.

There are two trivial solutions for this problem: Alice sends her data and Bob performs the classification, or Bob sends his classifier and Alice performs the classification. Both these solutions have a clear drawback. Alice don't want to disclose Bob her data or the output of the classification (in the case of the previous medical example, Alice don't want to reveal Bob if she is actually healthy or not). On the other hand, Bob might not be happy to disclose the classifier, since he probably invested time and money training his network.

Again we can find a trivial solution, that is to find a trusted third party (TTP) that takes inputs from Alice and Bob, and gives back to Alice the output of the computation. Of course



(a) Perceptron

(b) Multilayer network

(c) General feedforward network

Fig. 1: Three kinds of different networks that can be used with the proposed protocol. Figure (a) shows a single layer network, known as *perceptron*; Figure (b) shows a *multi-layer feedforward network* , while Figure (c) shows a *general feedforward network*. Note that in feed-forward networks it is possible to number neurons such that every neuron gets inputs only from neurons with smaller index.

this scenario is not the ordinary one. If Alice and Bob distrust each other, it could be hard to find a common TTP.

Our goal is to design a protocol for remote data classification that respects the above privacy constraints without resorting to a TTP, and where cryptography and some randomization will play the role of the TTP. As many other protocols in literature, we will make the assumption that both Alice and Bob are semi-honest. This means that they will follow the protocol properly, but they can later analyze the protocol transcript trying to discover information about other party inputs. We will also assume that Alice and Bob can communicate over a secure (private and authenticated) channel, that can be implemented in practice as a standard SSL connection.

To grant protection against malicious adversary (i.e. adversary that can behave differently from what they are supposed to do), there are standard constructions to produce a protocol secure in the malicious scenario from a semi-honest one [GMW87]. Moreover in our protocol, there is no evidence that a cheating player can discover significant information about other party inputs, and therefore we can assume that semi-honest behaviour is forced by external factors i.e. Bob being a service provider doesn't want to lose his reputation, and Alice is interested in having her data correctly classified.

3.1 Protocol Requirements

Correctness. It means that Alice wants to have a correct classification of her data. As Bob gets no output, only Alice has interest in this property. We are not concerned here with the accuracy of Bob's NN, nor we could be. What we mean here is that Bob could maliciously give an incorrect answer to Alice, i.e. in the medical example, a corrupted Bob is instructed to reply "ill" to a given list of people. We could avoid this kind of cheating by running our protocol over an anonymized network as is [DMS04]. In general as our protocol is only secure against semi-honest adversaries, we can't ensure correctness: to do so, we have to transform our protocol into a protocol secure against malicious adversary using standard compiler techniques [GMW87].

Alice's Privacy. Alice's data are completely protected. In fact, Alice gives her input in an encrypted format and receives the output in an encrypted format. So Alice's privacy relies on the security of the underlying cryptosystem that is, in our case, semantic security.

Bob's Privacy. While it's clear what we mean with Alice's privacy, it's not the same with Bob's privacy. We need again to refer to the TTP ideal scenario: as long as Alice gets her data classified, she's learning something about Bob classifier. She learns in fact how Bob classifies a vector data. Suppose that she runs the protocol an arbitrarily number of times: she can submit every (or a very large database of) instances to be classified by Bob. She can later easily build a table with the classification of every vector, building in that way a classifier that works the same of Bob's one for classified vector, and she will be able to classify also new vectors, maybe building a new neural network with this data as the training set.

This result does not depend on the protocol security (we are talking about the TTP ideal scenario) but simply on the fact that from the input and the output of a computation it is always possible to learn something about the computed function (and this is exactly the goal of machine learning).

A natural question is therefore: is it possible to model this kind of attack, where a user interact many times with the classifier to understand his structure? How many interactions do the attacker needs? This question is partially answered in watermarking literature under the name "sensititvity attack". For a linear classifier (perceptron) the number of iterations to completely disclose it is very low [CL97, KLvD98]. For a more complex function, like a multi-layer network, it's harder to say, but there are still solutions allowing to extract a local boundary even without knowing anything about the nature of the classifier [CPFPG06].

Here is what our protocol protects about Bob's NN:

Structure of the Network: Bob network is composed of a certain number of neurons, connected in some feed-forward way. The protocol is designed to completely protect the way the neuron are connected, and to partially protect the number of neurons actually present in the network, so that Alice will get an upper bound B for the number of neurons. Bob can adjust this bound B in a way to achieve a good level of privacy and at the same time an efficient protocol.

Hidden Neurons Output: we also have to protect the outputs of the hidden neurons, as this is an unneeded leakage of information.

Given a hidden neuron with sigmoid activation function, it is possible to split its output in two parts, one that we can perfectly protect, while the other will be partially disclosed.

- *State of the Neuron:* this is the most important part of the output. Depending on the sign of the weighted sum of the inputs, every neuron can be *on* (i.e. output > 0.5) or *off* (i.e. output < 0.5). We will perfectly protect this information, flipping it with probability one half, achieving a one-time pad kind of security.
- *Associated Magnitude:* the activation of every neuron has also a magnitude, that gives information about "how much" the neuron is *on* or *off*. We will hide those values in a large set of random values, in such a way that Alice will not learn which values are actually part of the computation and which ones are just random junk. Of course the bigger the set is, the more privacy it comes, and more computational resources are needed.

– *Neuron's Position:* the last trick to achieve security is to completely randomize the position of the hidden neurons in such a way that Alice will not discover which outputs correspond to which neurons. Therefore Alice may learn a bit of information at every execution (meaning something about the magnitudes of them), but she'll not be able to correlate those information in order to gain advantage from repeated execution of the protocol.

Round Complexity. We would like to run the computation with no interaction (except the minimum needed to input the vector and get the output). Unluckily, there are only few kinds of algorithms that can be computed in such a way, that is NC^1 circuits [SYY99], and $NLOGSPACE$ problem [Bea00]. Our protocol has a constant round complexity, that is the number of layers of the network.

4 Privacy-Preserving Protocol for Remote Data Classification

We will continue to use the notation introduced before, together with some new symbols to refer to the encrypted versions of the data. The input of the neuron is $\boldsymbol{x}$ and its encrypted version is $\boldsymbol{c}$, while the encrypted version of the activation a of every neuron will be referred as d.

4.1 Building blocks

Homomorphic Encryption. The chosen cryptosystem to instantiate our protocol is the Damgård-Jurik modification [DJ01] of the Paillier encryption scheme [Pai99].

This cryptosystem is based on the hardness to decide the n-th residuosity of elements modulo n^{s+1}, where n is an RSA modulo. At the end, the encryption and the decryption procedures are the following:

Set-up: select p, q big primes. Let $n = pq$ be the public key, while the secret key, called λ, is the least common divisor between $(p-1)$ and $(q-1)$.

Encryption: let $m \in \mathbb{Z}$ be the plaintext, and s such that $n^s > m$. Select a random value $r \in \mathbb{Z}^*_{n^s}$; the encryption c of m is:

$$c = E_{pk}(m, r) = (1+n)^m r^{n^s} \mod n^{s+1}$$

Decryption: the decryption function D_{sk} depends only on the ciphertext, and there is no need to know the random r in the decryption phase. We refer to the original paper for the complete description.

The main advantage of this cryptosystem is that the only parameter to be fixed is n, while s can be adjusted according to the plaintext. In other words, unlike other cryptosystems, where one has to choose the plaintext m to be less than n, here one can choose an m of arbitrary size, and then adjust s to have $n^s > m$ and the only requirement for n is that it must be unfeasible to find its factorization.

The trade-off between security and arithmetic precision is a crucial issue in secure signal processing applications. As we will describe in Appendix, a cryptosystem that offers the possibility to work with an arbitrary precision allows us to neglect that the cryptosystem works on integer modular numbers, and we can consider this cryptosystem as an arbitrarily accurate non-integer homomorphic encryption scheme.

For the sake of simplicity from now on we will indicate the encryption just as $c = E(m)$, as the keys are chosen once and are the same for all the protocol length, and the random parameters r are just to be chosen at random. If $x_1 = x_2$ we will write $E(x_1) \sim E(x_2)$. The encryption of a vector $\boldsymbol{c} = E(\boldsymbol{x})$ will be simply the vector composed of the encryption of every component of the plaintext vector.

As said, this cryptosystem satisfies the homomorphic property so, given two plaintexts m_1 and m_2, the following equalities are satisfied:

$$D(E(m_1) \cdot E(m_2)) = m_1 + m_2 \tag{1}$$

and

$$D(E(m)^a) = am. \tag{2}$$

Privacy Preserving Scalar Product (PPSP). A secure protocol for the scalar product allows Bob to compute an encrypted version of the scalar product between an encrypted vector given by Alice $\boldsymbol{c} = E(\boldsymbol{x})$, and one vector $\mathbf{w}$ owned by Bob. The protocol guarantees that Bob gets nothing, while Alice gets an encrypted version of the scalar product that she can decrypt with her private key. Such a protocol is easily achievable exploiting the two homomorphic properties (see Equations 1 and 2):

Input: $\mathbf{c} = E(\mathbf{x})$; Bob: $\mathbf{w}$
Output: Alice: $d = E(\boldsymbol{x}^T \boldsymbol{w})$
PSPP($\mathbf{c}$;$\mathbf{w}$)
(1) Bob computes $d = \prod_{i=1}^{N} c_i^{w_i}$
(2) Bob sends d to Alice

After receiving d, Alice can decrypt this value with her private key to obtain the weighted sum a.

It is worth observing that though the above protocol is a secure one in a cryptographic sense, some knowledge about Bob's secrets is implicitly leaked through the output of the protocol itself. If, for instance, Alice can interact N times with Bob (where $N = |\mathbf{x}| = |\mathbf{w}|$ is the size of the input vectors), she can completely find out Bob's vector, by simply setting the input of the i-th iteration as the vector with all 0's and a 1 in the i-th position, for $i = 1, \ldots, N$. If we use the scalar product protocol described above to build more sophisticated protocols, we must be aware of this leakage of information. This is again a *sensitivity attack*, as introduced before. Note that the problems stemming from sensitivity attacks are often neglected in the privacy preserving computing literature.

Evaluation of the Activation Function If the function g is known and it's invertible, like in the case of the sigmoid function, the information given by a or $y = g(a)$ is the same. So Bob can simply give a to Alice that can compute y by herself.

4.2 Perceptron Protocol

Now we have described all the tools that allow us to run a privacy preserving remote data classification protocol for a single layer network. The network has I inputs and 1 output. If the network has more than one output neuron, say O, just run in parallel O instances of the following protocol.

Input: $\mathbf{c} = E(\mathbf{x})$; Bob: $\mathbf{w}$
Output: Alice: classification of $\boldsymbol{x}$
PERCPETRON($\mathbf{c}$;$\mathbf{w}$)
(1) Alice and Bob run the PPSP protocol.
(2) Alice decrypts the output $a = D(d)$
(3) Alice computes $g(a)$

4.3 Handling with Hidden Neurons

We consider now the more interesting case of a feedforward network.

As already defined, a feedforward network is composed by N neurons that can be ordered in a way that neuron j gets in input the output of a finite set I_j of neurons having index lower than j, like the ones in Figure 1. We use this ordering to label every neuron. The weight of the connection from neuron i to neuron j is indicated by w_{ij}. The input vector of neuron j is $\boldsymbol{x}_j$ while the associated weights vector $\boldsymbol{w}_j$. So now we need to protect the output of hidden neurons and the network topology.

Hidden Neurons Output Protection. In the perceptron protocol, Bob gives to Alice the linear sum a of the output neurons, and then Alice computes by herself the activation function output. The simple iteration of the perceptron protocol for every neuron in the network will disclose the activation value of every hidden neuron, but Alice is not supposed to get this information. The activation of every a can be viewed as $a = sign(a) \cdot |a|$. Depending on $sign(a)$ the output of the sigmoid function will be "almost 1" (i.e. $0.5 \leq y < 1$) or "almost 0" (i.e. $0 < y \leq 0.5$). We can perfectly protect $sign(a)$ by exploiting the fact that the sigmoid function is actually antisymmetric as shown in Figure 2, i.e. $g(-a) = 1 - g(a)$.

Bob can randomly change the sign of a just before send it to Alice thanks to the homomorphic property, since $E(a)^{-1} = E(-a)$. Then Alice can decrypt as usual the value received, compute the activation function on the received input $g(-a)$ and send $E(g(-a))$ back to Bob. Bob can recover the value he needs simply performing the subtraction, that is $E(g(a)) = E(1-g(-a)) = E(1)E(g(a))^{-1}$. In this way Alice will not discover which (nor how many) neurons are actually activated or not. We will deal with the protection of the value $|a|$ later.

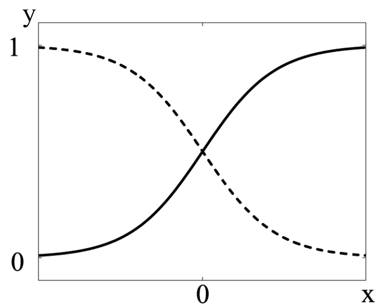


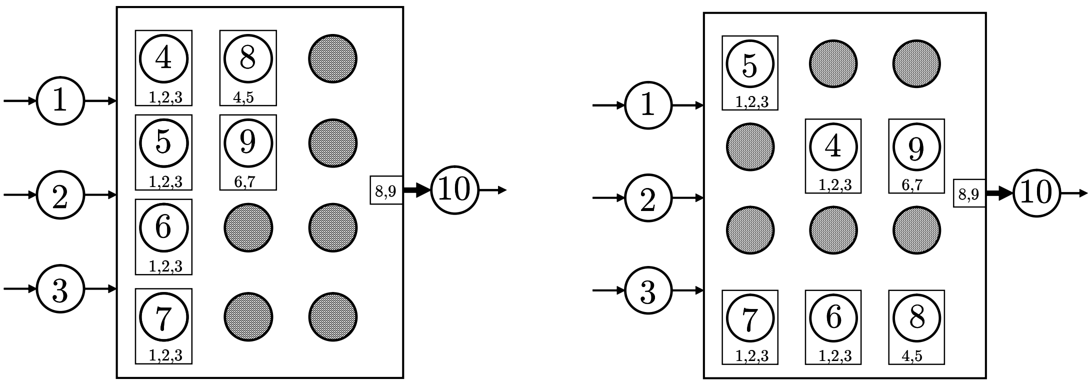
Fig. 2: Sigmoid functions is antisymmetric with respect to $(0, 1/2)$ as shown. That is $g(-a) = 1 - g(a)$.

Network Embedding. To protect the network topology, i.e. the number of neurons and the way they are connected, we will embed the network in a multilayer network, composed of L layers of M neurons each. Of course $LM \geq N$. The added $LM - N$ neurons will be called *fake neurons.* They have to look the same of the real neuron and so they will be initialized with some incoming connection from other neurons, with random weights. They will not influence the computation as no real neurons will take their output as input.

A good embedding is one where every neuron only get inputs from neurons that are in previous layers. An example of a legal embedding of the network in Figure 1 (b) is given in Figure 3.

In this way Alice will only learn an upper bound LM for the actual number of neurons N, while she will not learn the way they are connected or the connection weights, as Bob can perform the PPSP for every neuron by himself just picking up the necessary inputs and make the weighted sum with his private weights. The number L also gives a bound about the longest path between input and output neurons. Instead M is not the upper bound of the incoming connection for a neuron, given that we can split one original layer into two or more layer in the embedded network. Every neuron in layer l can take inputs from any neuron belonging to any previous layer.

The more we increase L, the more we protect the network topology. At the same time L will be the number of interaction round of the protocol, so we have to find a tradeoff between round complexity and network topology protection.



(a) Network Embedding (b) Network Scrambling

Fig. 3: Figure (a) is a natural embedding of the network in Figure 1 (b) into a 3×4 network. The big box refers to the protected zone. The list of numbers under the neurons indicates which inputs are given to that neuron. Filled neurons refer to fake neurons, that are meaningless for the actual computation, as no other neuron takes them in input. In the setup phase we need to assign some inbound connection for this neuron, to be sure that their output will be undistinguishable from that of real hidden neurons. Figure (b) shows a legal scrambling of the network, where every neuron gets input only from inputs belonging to previous layers.

Network Randomization. At this point we have to deal with the protection of the value $|a|$. We are concerned with the disclosing of $|a|$ because if Alice is allowed to run the protocol several times, she can use this additional information to actually understand the network weights. The solution we propose is to scramble all the network every different execution. A legal scrambling is one that preserves the property of the embedding, i.e. every neuron only gets input from neurons belonging to previous layers, as the one shown in Figure 3 (b). We note that there are at least $L \cdot M!$ legal scrambling, that are the ones that permute only neurons between the same layer. Increasing the number of neurons per layer M we can get a higher number of different permutations and so a better protection for the network, at the cost of some more computation to do.

We define the *embedding ratio* as the ratio between the number of hidden neurons of the embedded network and the number of hidden neurons in the original network LM/N. As this ratio increase, Alice will see more meaningless values, and therefore she won't be able to understand which values refers to real hidden neurons and which ones don't. Together with the network scrambling, this is a countermeasure to Alice's attempt to run a sensitivity attack against the protocol.

4.4 Multi-Layer Network Protocol

The final protocol is the following:

System Setup: Bob chooses L, M and finds a legal embedding of his network in the $L \times M$ one.

Execution Setup: Alice and Bob agree on a quantization factor Q, and on a parameter s for the used cryptosystem. Alice generates a pair of public and private keys and gives the public one to Bob. Bob will randomly scramble the neurons position in the network to reach another legal configuration.

Execution: in the protocol description we will consider real neurons and fake ones as the same, as there's no difference, but the fake one's output will never be used again.

Input: Alice: $\mathbf{c}$; Bob: $\boldsymbol{w}_j^{(i)}$ with $i = 1, \ldots, L$, $j = 1, \ldots, M$
Output: Alice: classification of x
$\mathrm{PPDC}(\mathbf{c}; \{\mathbf{w}_j^{(i)}\}_{i=1,\ldots,L; j=1,\ldots,M})$
for $i = 1$ **to** L
  for $j = 1$ **to** M
    Bob runs $\mathrm{PPSP}(\boldsymbol{c}_j^{(i)}; \boldsymbol{w}_j^{(i)})$ and gets $d = E(a)$
    Bob picks $t_j \in \{+1, -1\}$ at random
    if $t_j = -1$
      Bob sets $d = d^{-1}$
    Bob sends d to Alice
    Alice decrypts d, evaluates $g(a)$, and sends $z = E(g(a))$ back to Bob
    if $t_j = -1$
      Bob sets $z = E(1)z^{-1}$
for $j = 1$ **to** O //out-degree of the network
  Bob runs $\mathrm{PPSP}(\boldsymbol{c}_j; \boldsymbol{w}_j)$ and gets $d = E(a)$
  Bob sends d to Alice
  Alice decrypts d and evaluates her output $g(a)$

The security of this protocol follows from the previous considerations.

5 Implementation of the Protocol

In this section a practical implementation of the proposed protocol is described, and a case study execution that will give us some numerical results in term of computational and bandwidth resource needed is analyzed.

Client-Server Application. We developed a Java application based on Remote Method Invocation technology[5]. The software, which makes use of a modified implementation of the Damgård-Jurik cryptosystem available on Jurik's homepage[6], is composed of two parts: the client and the server. The former has to set the environment creating a couple of public/private keys and choosing the number of bits of the modulus, and it choose a server it wants to connect to. The latter can load several neural networks into the system and choose an appropriate quantization factor and embedding ratio; it also provide the clients a list of available networks.

[5] `http://java.sun.com/javase/technologies/core/basic/rmi/`
[6] `http://www.daimi.au.dk/~jurik/research.html`

Experimental Data. Two datasets were selected from the UCI Machine Learning Repository[7], and two kinds of neural network were trained starting from those data set:

- *Sonar:* this dataset refer to classification of sonar signals by means of a neural network [R$^+$97], which task is to obtain a network able to discriminate between sonar signals bounced off a metal cylinder (bombs) and those bounced off a roughly cylindrical rock; we have trained a NN with the standard backpropagation algorithm, containing 60 input neurons, 12 hidden neurons, and 1 output neuron.
- *Nursery:* this dataset was derived from a hierarchical decision model originally developed to rank applications for nursery schools [G$^+$88]; we have trained a NN with 8 input neurons, 20 hidden neurons, and 5 output neurons.

Experimental Setup. We loaded these NNs into the server changing the embedding ratio at every execution. The quantization factor has been set to $Q = 10^9$, obtaining a reasonable accuracy in the computation to prevent quantization error. In the key generation algorithm the key lenght has been set to $n = 1024$ to obtain a reasonable level of security [BBB$^+$05].

Then we deployed the application on two mid-level notebooks, connected on a LAN network.

The execution time increase linearly wrt the embedding ratio, as also the communication overhead does. Choosing carefully the embedding ratio, we can find the desired trade off between execution time and achieved security. Setting the embedding ratio to 10 offer a reasonable execution time. Of course the level of security we need is mostly application-based.

Results are pictorially represented Figure 4, where we can distinguish between the running time on the client, on the server, and the total amount of time. Given that the application was executed on a LAN network, the communication time is negligible.

The amount of exchanged bytes is reported in Table 1 for some values of the embedding ratio.

It is worthy to note that even if the *Sonar* NN has 73 neurons (60 input + 12 hidden + 1 output) while the *Nursery* NN has only 33 neurons (8 input + 20 hidden + 5 output), the former is computed in shorter time. This is due to the fact that the only piece of the network that has to be embedded are the hidden neurons[8]. Therefore the *Nursery* NN, that has more hidden neurons than the *Sonar* NN, need more time and bandwidth to be computed.

Neural Network	**embedding ratio**				
	5	10	15	20	25
sonar	153kB	256kB	359kB	470kB	579kB
nursery	232kB	368kB	544kB	722kB	894kB

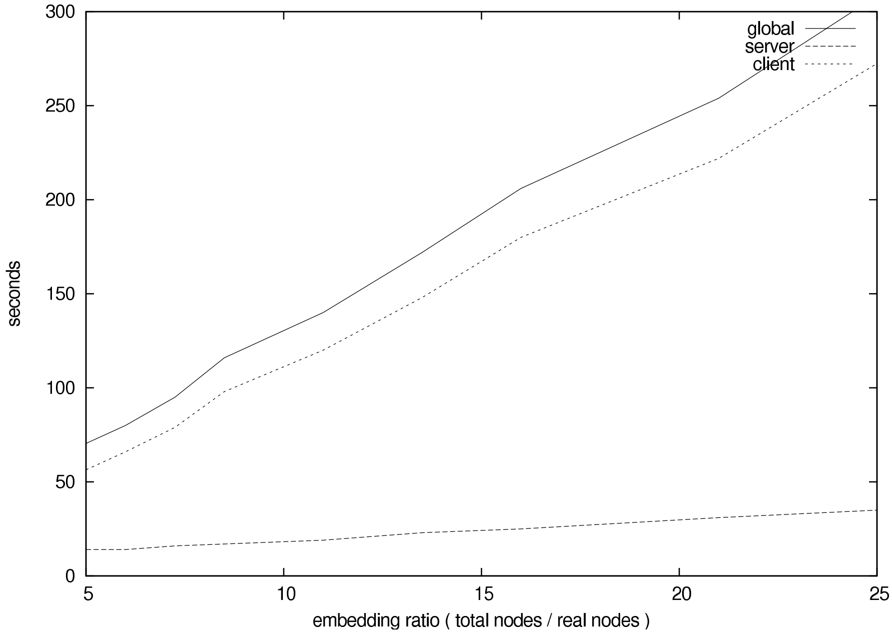
Table 1: bandwidth occupation

Workload. The client side workload is greater than the server side workload. This depends on the weight of the operations that server and client perform during the evaluation of every hidden layers:

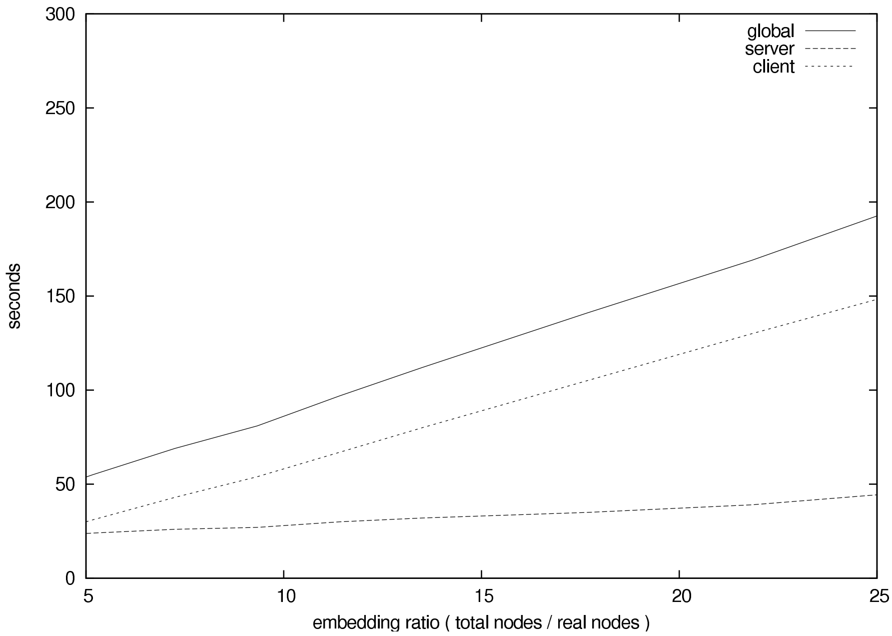
- the former performs simpler operations, like multiplication and exponentiation.
- the latter performs heavier operations, i.e. encryptions and decryptions.

[7] `http://www.ics.uci.edu/~mlearn/MLRepository.html`

[8] The input and output layers represent the input and the output of the function: adding fake inputs or outputs, or scrambling their position will results in a meaningless computation.



(a) *Nursery* neural network



(b) *Sonar* neural network

Fig. 4: Execution time

As far as we know this is the first protocol of this kind that was implemented, as no experimental results are mentioned in [CL05]. Therefore we can't compare our approaches.

6 Conclusions

Several modern artificial intelligence applications require the protection of the privacy of the data owner, escaping to reveal his/her input data to the owner of the classifier. In this framework, the availability of tools to process data and signals directly in the encrypted domain allows to build secure and privacy preserving protocols solving the mentioned problem. Given the central role that neural network computing plays in artificial intelligence field, a protocol for NN-based privacy-preserving computation has been designed, where the knowledge embedded in the NN as well as the data the NN operates on are protected. The proposed protocol relies on homomorphic encryption; for those tasks that cannot be handled by means of homomorphic encryption, a limited amount of interaction between the NN owner and the user is introduced; however, the interaction is kept to a minimum, without resorting to general multiparty computation protocols. Any unnecessary disclosure of information has been avoided, prtecting all the internal computations, so that at the end the user will only learn the final output of the NN computation.

Acknowledgments. The work described in this paper has been supported in part by the European Commission through the IST Programme under Contract no 034238 - SPEED.

References

[AS00] Rakesh Agrawal and Ramakrishnan Srikant. Privacy-preserving data mining. In *SIGMOD '00: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 439–450, New York, NY, USA, 2000. ACM Press.

[BBB+05] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid. Recommendation for Key Management–Part 1: General. *NIST Special Publication*, pages 800–57, 2005.

[Bea00] D. Beaver. Minimal-latency secure function evaluation. *Proc. of EUROCRYPT00, LNCS*, pages 335–350, 2000.

[Bis95] C.M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, 1995.

[BOP06] M. Barni, C. Orlandi, and A. Piva. A privacy-preserving protocol for neural-network-based computation. In *MM&Sec '06: Proceeding of the 8th Workshop on Multimedia and Security*, pages 146–151, New York, NY, USA, 2006. ACM Press.

[CL97] I. J. Cox and J. P. M. G. Linnartz. Public watermarks and resistance to tampering. In *Proc. 4th IEEE Int. Conf. on Image Processing, ICIP'97*, volume III, pages 3–6, Santa Barbara, CA, USA, 26-29 October 1997.

[CL05] Y.C. Chang and C.J. Lu. Oblivious polynomial evaluation and oblivious neural learning. *Theoretical Computer Science*, 341(1):39–54, 2005.

[CPFPG06] P. Comesana, L. Perez-Freire, and F. Perez-Gonzalez. Blind newton sensitivity attack. *Information Security, IEE Proceedings*, 153(3):115–125, 2006.

[DJ01] Ivan Damgård and Mats Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In *Public Key Cryptography*, pages 119–136, 2001.

[DMS04] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, August 2004.

[FSW03] P.A. Fouque, J. Stern, and G.J. Wackers. CryptoComputing with rationals. *Financial-Cryptography.-6th-International-Conference, Lecture-Notesin-Computer-Science*, 2357:136–46, 2003.

[G+88] P. Gorman et al. UCI machine learning repository, 1988.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC '87: Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 218–229. ACM, 1987.

[HSW89] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Netw.*, 2(5):359–366, 1989.

[JKM05] S. Jha, L. Kruger, and P. McDaniel. Privacy Preserving Clustering. *10th ESORICS*, 2005.
[KLvD98] Ton Kalker, Jean-Paul M. G. Linnartz, and Marten van Dijk. Watermark estimation through detector analysis. In *ICIP98*, volume I, pages 425–429, Chicago, IL, USA, October 1998.
[KV03] M. Kantarcioglu and J. Vaidya. Privacy preserving naive bayes classifier for horizontally partitioned data. In *In IEEE Workshop on Privacy Preserving Data Mining*, 2003.
[LF88] A. Lapedes and R. Farber. How neural nets work. *Neural information processing systems*, pages 442–456, 1988.
[LLM06] Sven Laur, Helger Lipmaa, and Taneli Mielikäinen. Cryptographically private support vector machines. In *KDD '06: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge discovery and data mining*, pages 618–624, New York, NY, USA, 2006. ACM Press.
[LP00] Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. *In Advances in Cryptology - CRYPTO '00, volume 1880 of Lecture Notes in Computer Science*, 1880:36–54, 2000.
[Pai99] P. Pailler. Public-key cryptosystems based on composite degree residuosity classes. In *Proceedings of Eurocrypt'99, Lecture Notes is Computer Science vol. 1592*, pages 223–238. Springer-Verlag, 1999.
[R+97] V. Rajkovic et al. UCI machine learning repository, 1997.
[RHW86] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning internal representations by error propogation. *Parallel distributed processing: Explorations in the microstructure of cognition*, 1:318–362, 1986.
[SYY99] T. Sander, A. Young, and M. Yung. Non-Interactive CryptoComputing For NC 1. *IEEE Symposium on Foundations of Computer Science*, pages 554–567, 1999.
[WY04] Rebecca Wright and Zhiqiang Yang. Privacy-preserving bayesian network structure computation on distributed heterogeneous data. In *KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 713–718, New York, NY, USA, 2004. ACM Press.
[Yao86] A. Yao. How to generate and exchange secrets. In *IEEE FOCS'86 - Foundations of Computer Science*, pages 162–167, 1986.
[YW05] Zhiqiang Yang and Rebecca N. Wright. Improved privacy-preserving bayesian network parameter learning on vertically partitioned data. In *ICDEW '05: Proceedings of the 21st International Conference on Data Engineering Workshops*, page 1196, Washington, DC, USA, 2005. IEEE Computer Society.

Appendix: Handling Non-integer Value

In section 4.1 we have highlighted that the Damgård-Jurik cryptosystem allows to work with integer values of arbitrary size. In this appendix we will describe how to exploit this property to obtain an arbitrarily accurate real number homomorphic cryptosystem.

First of all we map, in a classic way, the positive numbers in $\{0, \ldots, \frac{n^s-1}{2}\}$, and the negative ones in $\{\frac{n^s-1}{2} + 1, \ldots, n^s - 1\}$, with $-1 = n^s - 1$. Then, given a real value $x \in \mathbb{R}$, we can quantize it with a quantization factor Q, and approximate it as $\bar{x} = \left\lfloor \frac{x}{Q} \right\rfloor \simeq \frac{x}{Q}$ for a sufficiently thin quantization factor. Clearly the first homomorphic property still stands i.e.

$$D(E(\bar{x}_1) \cdot E(\bar{x}_2)) = \bar{x}_1 + \bar{x}_2 \simeq \frac{x_1 + x_2}{Q}.$$

This allows Bob to perform an arbitrarily number of sums among ciphertexts. Also the second property holds, but with a drawback. In fact:

$$D(E(\bar{x})^{\bar{a}}) = \bar{a} \cdot \bar{x} \simeq \frac{a \cdot x}{Q^2}$$

The presence of the Q^2 factor has two important consequences:

1. the size of the encrypted numbers grows exponentially with the number of multiplications;
2. Bob must disclose to Alice the number of multiplications, so that Alice can compensate for the presence of the Q^2 factor.

The first drawback is addressed with the availability of Damgård-Jurik cryptosystem that allows us, by increasing s, to cipher bigger numbers. The second one imposes a limit on the kind of secure computation that we can perform using the techniques proposed here. Luckily in our application we will perform only one multiplication for each ciphertext in the scalar product protocol.

The s parameter has to be chosen in a way that the value inside the ciphertext after the computation will fit into n^s.

Let X be the upper bound for the norm of Alice's input vector, and W an upper bound for the weight vectors norm. Every scalar product computed in the protocol is then bounded by $|\boldsymbol{x}||\boldsymbol{w}|\cos(\hat{\mathbf{xw}}) \leq XW$. Given a modulo n sufficiently large for security purposes, it is possible to select the parameter s such that:

$$s \geq \left\lceil \log_n \frac{2XW}{Q^2} \right\rceil$$

where the 2 is due to the presence of both positive and negative values.

Other solutions for working with non integer values can be found in [CL05] where a protocol to evaluate a polynomial on floating-point numbers is defined (but the exponent must be chosen in advance), and [FSW03], where a sophisticated cryptosystem based on lattice properties allowing computation with rational values is presented (even in this case, however, a bound exists on the number of multiplications that can be carried out to allow a correct decryption).