A Neural Network-Based Cryptographic System

Misti Clark and Douglas Blank

Department of Computer Science
University of Arkansas
232 Science Engineering
Fayetteville, Arkansas 72701

msavage@comp.uark.edu
dblank@comp.uark.edu

Abstract

We present a cryptographic system based on Pollack's recursive auto-associative memory (1991). This self-organizing learning system offers many unique properties not found in traditional crypto systems. This paper examines the methodology and issues of such a system.

Introduction

Cryptographic systems fall into two main categories: those that rely on secret mechanisms (such as the German World War II Enigma machine) and those that rely on public, yet hard-to-solve, mathematical problems (such as RSA). Most modern cryptographic systems are variations of the latter. This allows all of the associated algorithms to be public, since knowing the method does not allow one to break the code. However, given enough computer time, any such code can be broken. On the other hand, systems like the Enigma machine are virtually unbreakable, given that the mechanism is kept secret. However, creating a machine like the Enigma is a hard design problem. In this paper, we introduce a method of *learning* such a system through the use of a recursive auto-associative artificial neural network.

Our system is self-organizing and therefore requires no design decisions from the user. A unique property of the system allows for a variable number of symbols to be represented in each coded unit. To illustrate the properties of such a cryptographic system, we introduce the methodology in the next section through an example.

Sending and Receiving Messages

Imagine that we want to send the following message: "**We are in Havana; send troops.**" The first step of our encryption process is to break the message up into smaller chunks. Each message chunk (or *m-chunk* for short) is a portion of the entire message. However, each m-chunk contains a random number of symbols. For example, our message might be broken into {**We are**} {**in Havana send troops**}. Another way the message might be broken into m-chunks is {**We**} {**are in**} {**Havana send troops**}. For a message with n symbols, there are 2^{n-1} ways of breaking that message into m-chunks. Let us use {**We**

are} {**in Havana Send troops**} for the current encoding.

We next encode the m-chunks into sets of floating-point values called *packets*. Each packet is composed of a non-varying number of floating point numbers. For example, The m-chunk {**We are**} might be encoded by {.21 .87 .04}, and the m-chunk {**in Havana send troops**} could encoded by {.99 .34 .27}. Finally, the packets are concatenated to read {.21 .87 .04 .99 .34 .27}. This is the encoded message, and this is sent to military headquarters.

The decoding process operates like the encoding process in reverse. First, we break the string of values up into packets of the appropriate length (three in this case, which must be previously agreed upon between sender and receiver). This gives {.21 .87 .04} {.99 .34 .27}. The packets are then decoded into the following m-chunks: {**We are**} {**in Havana send troops**}. The message is finally assembled into **We are in Havana send troops**.

Notice that there is no way *a priori* to know how many symbols are contained in each packet; one group of three numbers represented two words, while the other represented four. Therefore, it is impossible to tell how much data is encoded in a message.

Our procedure requires the following elements: 1) a key, 2) message, 3) packet length, 4) terminal key, and 5) code translation table. These five requirements will be discussed in detail in the next section.

Network Details

The system is built on the concept of the *recurrent auto-associative neural network,* or RAAM (Pollack, 1991). An auto-associative network is one in which an input pattern is associated with itself. The goal of this paradigm is pattern completion (Rumelhart and McClelland, 1986). A neural network (sometimes called a *connectionist network*) is made up of *layers*. Inside each layer, there are *units*. The units of each layer are connected to every unit in the layer directly above them. These connections are referred to as *weights*. In an auto-associative network, the input layer is associated with the first n units of the output layer, where n is the number of units in the input layer, and the context layer is associated with the remaining units of the output layer (see Figure 1). The goal is that, given some pattern on the input layer, the network must reproduce that same pattern on the output layer. The context layer is treated as an input layer as well. The hidden layer must be smaller than the sum of the input and the context layers in order to force the network to accomplish a compressed representation of the sequences. The set of connections, or weights, from the input layers to the hidden layer represent the encoding method, and the set of connections from the hidden layer to the output layer represent the decoding method. The back-propagation learning algorithm is used to perform the auto-association. For an in-depth examination of this process, see (Blank, Meeden, and Marshall, 1992).

Let us step through the mechanism of encryption and decryption. To encrypt the original message, each word or symbol of the m-chunk is fed into the input layer sequentially. In our previous example, we start with the m-chunk {**We are**}. The representation of the symbol **We** is placed into the network on the input layer, and the context layer is set to the *terminal key* (this is explained in more detail in the next few paragraphs). The representation of **We** is then propagated through the network. Afterward, the hidden layer will contain the floating-point representation for the symbol **We**, combined with the terminal key. Next, the symbol **are** is ready to be input. Here is

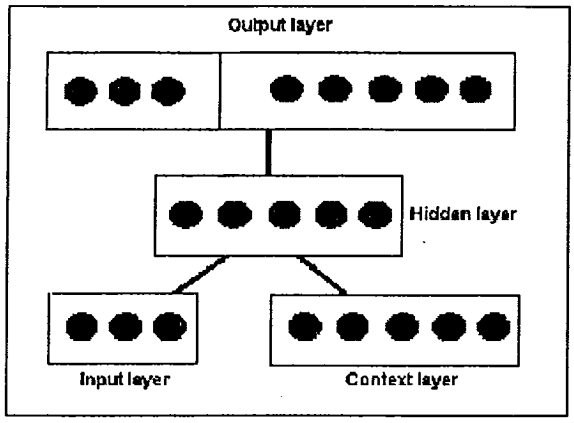

Figure 1. Recursive Auto-Associative Memory Network Architecture.

where the context layer is important. Before **are** is input, the hidden unit representation (the symbol **We** combined with the terminal key) is copied into the context layer. The symbol **are** is then sent through the network. This continues until all of the symbols in the m-chunk are fully input into the network. The hidden layer will contain a compressed representation for the entire chunk. In this case, the hidden units contain the values {.21 .87 .04}. Notice that the only portion of the network that is needed for the encryption is the connections between the input, context, and hidden layers.

For decryption, a packet is placed into the hidden layer and propagated through the network. Recall that a packet is an encrypted m-chunk. We place the packet {.21 .87 .04} into the hidden units and propagate. On the output layer, two things will appear: on the first *n* units of the output layer, we will receive the code that represents the word **are**. On the remaining units of the output layer we receive a terminal key. If this terminal key matches, within some epsilon, the one that was chosen to start the encryption, then we are finished. As the terminal key does not match at this point, we copy the activations in the last units of the output layer to the hidden layer, and then we send those numbers through the network. This time, **We** comes up in the first units of the output layer, and the remaining units match the terminal key. The decryption of this packet is then complete.

As shown, the process requires five elements. The first, the key, is the set of the connections, or weights, on each of the edges. Without these weights, neither encryption nor decryption can be accomplished. The second requirement, the message, refers to either the original message or the encoded message. The third item is the packet length. The packet length must be the same as the number of units in the hidden layer, as this is where a packet is placed for decryption. The fourth element, the terminal key, is important as it signifies the end of the decryption. Once there is a match on the last output units with the terminal key, within some epsilon, the decryption terminates. There may not be an exact match to the terminal key due to noise in the network. The final requirement is a code translation table. The output during decryption is a binary representation of a symbol, i.e. 1 0 0 0. A code translation table is necessary to the map the output to the pre-specified symbol.

The network is trained on binary representations of letters and words. For a network to learn the alphabet, the network would have 26 units on the input layer. There is no predetermined number of units in the hidden layer, but the output layer will be 26 plus the number of units in the hidden layer.

Discussion

Is this type of encryption secure? Standard techniques, such as usage statistics, will not work on this encryption method. For example, there is no way to find the most used symbol, (i.e., the letter "e" or the word

"the"). The encoded message is not a one-to-one map from symbol to symbol, so a cryptographic analysis is not feasible.

If a network can be trained to create these encoded messages, could another network be trained to break it? Assuming that someone received the encoded message .21 .87 .04 .99 .34 .27, *and* they knew that the message was, **We are in Havana Send troops**, it appears to be infeasible to train a new network to produce the correct encoding. First, what would the packet length be? For an encoded message of length 6 (as we have above), there are 4 logical guesses for the packet length: 1, 2, 3, or 6. But even if the correct packet length were guessed what output would be used to train the network? With a packet length of three, there are five possibilities for the output: **We, We are, We are in, We are in Havana, and We are in Havana Send**. In addition, because the number of symbols in a packet are chosen randomly, without knowing how the message was broken into m-chunks, one cannot obtain the correct output for training. Also, one would need many messages to train a network on, not just one.

What are the limitations of the system? The limitations of this type of system are few, but potentially significant. This is effectively a secret-key system, with the key being the weights and architecture of the network. With the weights and the architecture, breaking the encryption becomes trivial. However, *both* the weights *and* the architecture are needed for encryption and decryption. Knowing only one or the other is not enough to break it. At the moment, this does not appear to be commercially valid as a form of encryption, as it cannot be distributed. However, it does have promise as a compression tool and several other uses.

What are the advantages to this system? The advantages to this system are that it appears to be exceedingly difficult to break without knowledge of the methodology behind it, as shown above. In addition, it is tolerant to noise. Most messages cannot be altered by even one bit in a standard encryption scheme. This system allows the encoded message to fluctuate and still be accurate. For example, if our encoded message were altered during transmission to .22 .87 .07 .99 .35 .27, this could still produce the correct output.

Summary

This paper has introduced a new method of encryption. It improves upon standard techniques because it represents variable length sequences in a fixed-length notation. It is also seemingly impossible to break using current techniques for codebreaking, as the same message can be represented in many different encodings. In addition, it allows for noise during transmission, a possible advantage over standard encryption schemes.

References

Blank, D., Meeden, L., and Marshall, J. (1992). Exploring the Symbolic/ Subsymbolic Continuum: A Case Study of RAAM. In *The Symbolic and Connectionist Paradigms: Closing the Gap*. J. Dinsmore, ed. Lawrence Earlbaum and Associates.

Pollack, Jordan (1990). Recursive Distributed Representations. In *Connectionist Symbol Processing*, G. Hinton, ed, MIT Press.

Rumelhart, D., and McClelland, J. (1986). *Parallel Distributed Processing, Vol 1*. MIT Press.