Hyper Heuristic Cryptography With Mixed Adversarial Nets

Thesis Defense Monologue

Author: Aly Shmahell

Each work of science has a fundamental question it tries to answer.

My thesis is trying to solve neural cryptography.

Neural Cryptography is the implementation of crypto-systems based on heuristic methods using neural nets.

My work is based on a paper released by Google Brain in 2016 entitled "Learning to Protect Communications with Adversarial Neural Cryptography".

In the Google paper the focus was on building a system that can guarantee some form of confidentiality of information between a sender and a receiver, from an eavesdropper.

My work extends that by adding more focus on information integrity between the sender and the receiver.

In the Google paper, they trained two networks: a sender called Alice and a receiver called Bob, in adversarial mode against a known adversary named Eve.

I extended that by adding an unknown malicious adversary named Alan as a metric of how well the system performs in real life.

As you can see, in the Google paper, symmetric cryptography does well enough to guarantee confidentiality, but to a limit, The sender encrypts 16 bits, and Eve decrypts 8 of them wrong, that is 50% confidentiality. Bob decrypts all bits right and gets to 0% error only after 15,000 training iterations.

Also in the Google paper, asymmetric cryptography does a bare minimum in confidentiality, as Eve decrypts 6 bits wrong out of 16, that is 37% of confidentiality. While bob decrypts 1 bit wrong, that is 6.25%, which means bob guarantees only 93.75% of information integrity.

What my thesis also adds, is a testing mode, in which back-propagation and weight correction stop, but data keep flowing to the nets, in order to know if the error rates achieved during training can hold outside of training.

When we ask ourselves about the merit of trying to teach neural nets how to do cryptography, we have to think about the latest developments in neural nets and what they bring to the table:

- **Neural Cryptography Is Viable:**
  convolutional nets can construct local spatial relations in data.
- **Neural Cryptanalysis Is Viable:**
  fully connected layers can detect global spatial relations in data.
- **Neural Cryptography Can Be Fast:**
  convolutional nets share weights using their filters.
- **Neural Cryptography Is Evolved, Not Patched:**
  using adversary in training evolves weights which serves to tweak the cryptographic functionality.

In the thesis as a whole I present:

- **A Prototype Blueprint:**
  for a software-engineered neural crypto-system.
- **An Analysis Of How Neural Components Work:**
  when the objective is to achieve cryptographic functionality.
- **An Enhancement In The Neural Structures:**
  which yields a boost in cryptographic robustness.
- **Transfer Learning:**
  to get symmetric neural cryptography on par with asymmetric neural cryptography.

If we move on to the experiments conducted in the thesis we have 3 cryptographic schemes implemented:

- **a symmetric scheme.**
- **an asymmetric scheme**
- **a hybrid scheme**

In the symmetric scheme, while training, the error rate of bob reaches 0%, after only 1000 iterations in most test cases, and that is 100% information integrity rate in a short training period.
while eve has an error rate of 80%, and Alan 60%, both metrics are an improvement over the Google paper.

In testing though, in this test case presented here, bob has an error rate of 5%, but in most cases it reaches 0%, and this kind of contrast between testing and training is why it is essential to evaluate neural performance outside training.

The hybrid scheme aims to import the performance of asymmetric neural cryptography to symmetric neural cryptography by employing transfer learning.

Transfer learning is the neural reconstruction of a source learning task in a source domain, to become a target task in a target domain, provided that either the two domains are not the same or the two tasks are not the same, but not both.
because if both domains and both tasks are different then there is no portability, and if both domains and both tasks are the same then there is nothing to import from one to the other.

When I implemented my designs, I had great help from Prof. De Gasperis in doing so in a software-engineered OOP manner.
I used the Tensorflow Python API to implement almost everything, with some help of Matplotlib and Numpy.
I also did some experiments with random key-message generation done by audio sampling, by employing the hashing library in python.

The general OOP structure has 2 common classes, general_hyper_parameters which loads those from a JSON file, and a base class which handles generic net construction, training, and data processing.

Each scheme has its own specific hyper parameters class which readjusts the values of these parameters to suit the scheme.
each scheme has its own specific training model builder class, testing model builder class, and model tester class.

At the end there is an inheritor class which inherits the right modules for the scheme.

The exception to this is the hybrid scheme inheritor which inherits the asymmetric hyper parameters class, because it needs to train in asymmetric mode, it also inherits from the symmetric tester class because it tests in the symmetric mode.

The last class or module we have is the bare neurencoder class which handles user input that is which scheme to activate, once that is set, it composes that scheme internally after it has been inherited.

Finally, when we talk about improvement in performance, a huge part of this comes down to which activation functions have been chosen after each layer in each net.

As you can see, there is an activation function after the input layers, one after the convolutional layers, and one at the end of the output layer.

The Google paper used sigmoid after input and convolution, and tanh after output, however, I've seen a room for improvement, since it's defacto to say after convolution it's best to use leaky relu
because it saves compute power, and yields better results,
So I've contemplated the following options:

- ”Sigmoid → LeakyRealu → Sigmoid”.
- ”Tanh → LeakyRealu → Tanh”.
- ”Sigmoid → LeakyRealu → Tanh”.

The Empirically Reliable Choice:
”Sigmoid → LeakyRealu → Tanh”.

To get a sense of why they work, I did some numerical analysis of each combination, by feeding each function the output of the other, multiplied by a line-space to simulate a layer.
Then I plotted the final output, to get a good interpretation of the data, we can treat each net as an information channel, then Shannon's theory applies to it.
”Sigmoid → LeakyRealu → Sigmoid”: is equiprobable which means it has a uniform entropy, but it doesn't have information capacity for negative values, which we deal with in our input.
”Tanh → LeakyRealu → Tanh”: is inequiprobable, and given that input data is generated from the x-axis according to a random distribution, this means the combination has non-uniform entropy and is unstable.
”Sigmoid → LeakyRealu → Tanh”: is equiprobable and has information capacity that covers the negative range of the input.

In conclusion, the research presented in this thesis shows that neural nets can learn to minimize loss of information integrity while trying to keep it confidential, it shows that with more research there is the potential of neural cryptography being on par and even surpassing traditional methods, it also shows the feasibility of implementing such a neural crypto-system on modern devices.