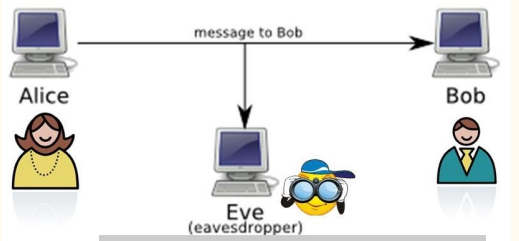
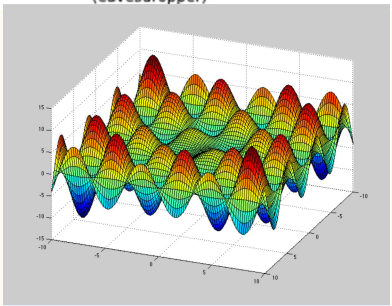
Adversarial Neural Cryptography

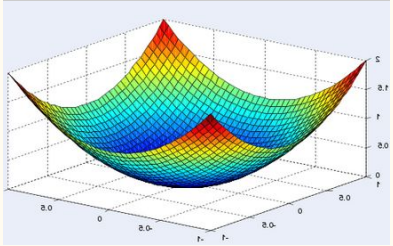
—

Matt Sokoloff

Background



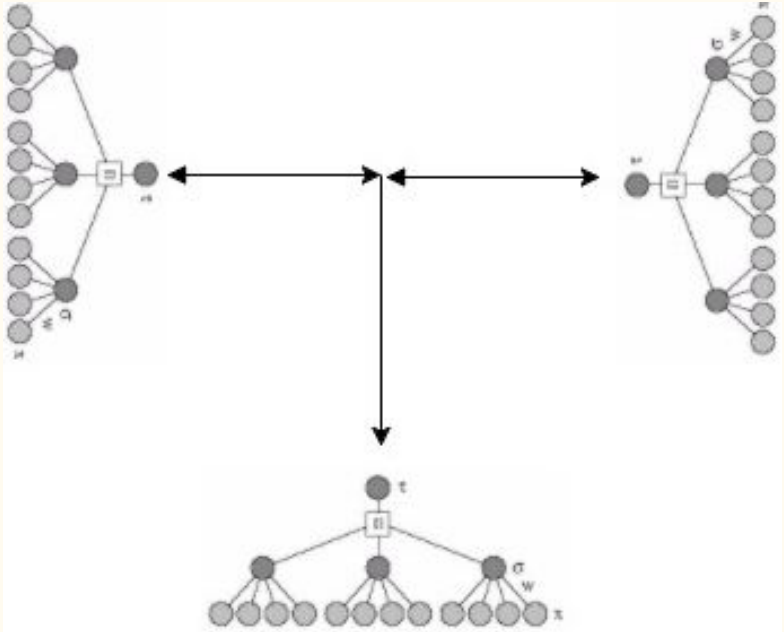




- Based on a Google Brain paper
  - Used neural networks to learn how to protect communications.
    - There is a separate neural network for each of the following tasks:
      - Encoding the data
      - Decoding the data
      - Simulate an unauthorized party attempting to decode the data
  - Concept derived from generative adversarial networks (GANs).
    - All three parties improve at the same time. Otherwise convergence is impossible
    - These models are notorious for unstable convergence properties (non-convex loss).
- The goal was to recreate the paper and search for improvements

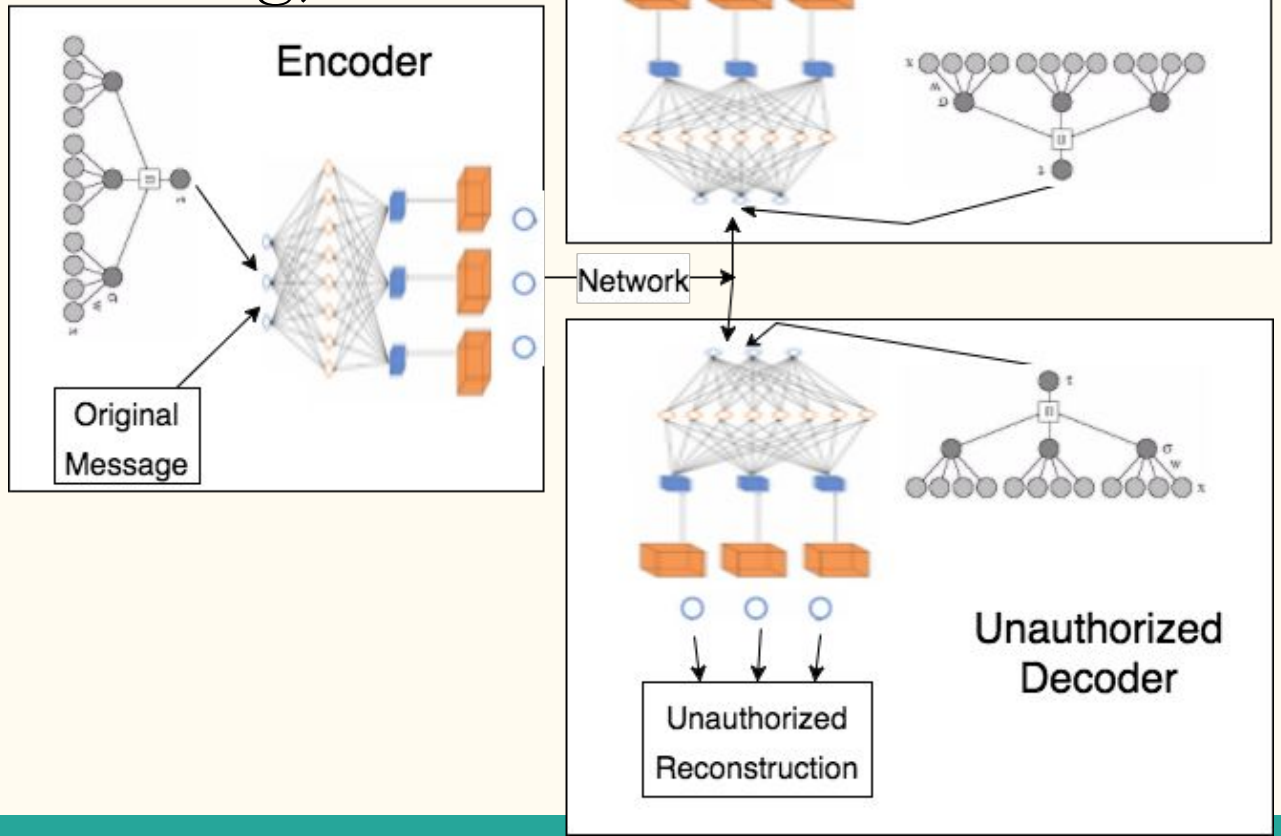
Architecture Overview (Initial key exchange)

- Tree parity machine to exchange key over public channel. Discrete neural networks.
- Communicating parties have an advantage because they can influence each other (random walk). Third party can only listen.
- Man in the middle will set off alarm. Convergence expected after ~500 iterations.
- Inputs and outputs are transmitted over the network.



Architecture Overview (Adversarial Training)

- Encoder and Decoder attempt to maximize reconstruction between each other and minimize the unauthorized reconstruction.
- After training for about 5000 iterations, the unauthorized decoder can no longer break the encryption. This is true even when the encoder and decoder models stop training.
- Input seed is public knowledge.



Network Architectures

- All Networks had the same configuration except for the input.
- Tree parity machines had 100 branches (K) each with 32 nodes (N) and an absolute maximum value of 10.

Layer1	Fully connected, same number of input nodes as output nodes (message length + tree parity key length) , sigmoid activation
Layer2	convolutional, 2 output channels, filter width of 2, stride of 1, sigmoid activation
Layer3	convolutional, 4 output channels, filter width of 2, stride of 2, sigmoid activation
Layer4	convolutional, 4 output channels, filter width of 1, stride of 1, sigmoid activation
Layer5	convolutional, 1 output channel, filter width of 1, stride of 1, hyperbolic tangent activation
MappingLayer	Normalizes output between 0 and 1

Loss Function/Update

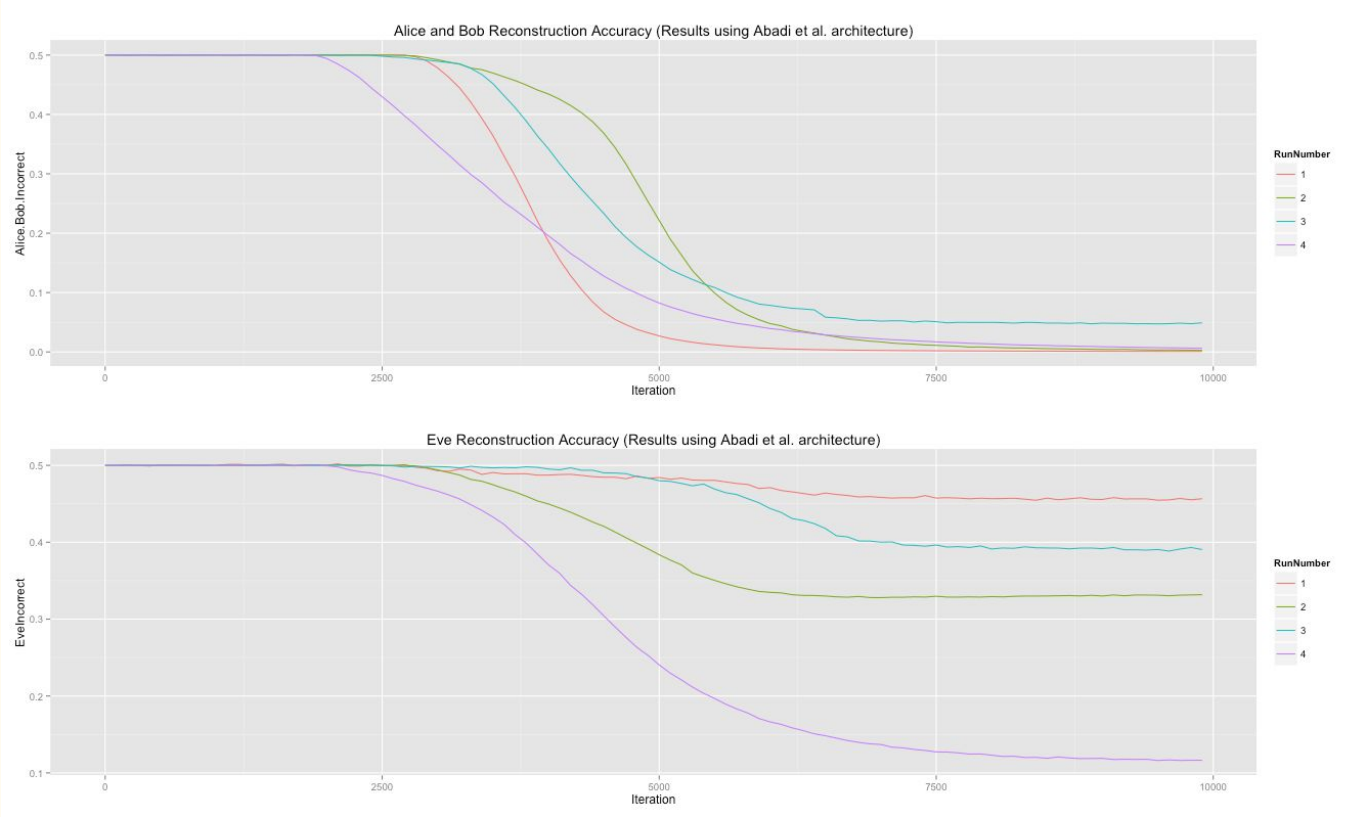
- Tree Parity Machine
  - Hebbian Update
    - $\vec{w}_i = \vec{w}_i + \vec{x}_i \tau_a \sigma_i \tau_a \tau_b$
- Neural Networks
  - Loss
    - Encoder (a)/Decoder(b).
      - $Loss_b = Loss_a = \frac{\sum_{n=1}^{N} |output_b - correct|}{N} + \frac{(\frac{messageLen}{2} - Loss_e)^2}{(\frac{messageLen}{2})^2}$
    - Unauthorized Decoder(e):
      - $Loss_e = \frac{\sum_{n=1}^{N} |output_e - correct|}{N}$
  - Minimize Loss using gradient descent.
    - Adam (Adaptive Moment Estimation) optimizer used to improve convergence properties.
      - Essentially adjusts momentum and learning rate for each weight's gradient.

Training (continued)

- The procedure for applying the updates previously applied are as follows:
  - First we completely train the tree parity machines. We restart if convergence is not achieved within 500 iterations to prevent man in the middle attacks.
  - To challenge the encoder/decoder networks, we train the unauthorized decoder twice for every update of the encoder/decoder.
  - The key changes for every single example and is a random sample of the tree parity machine weights.
  - After about 10000 iterations, we only train the unauthorized decoder to see if it can break the encryption.

Results

- Convergence occurred majority of the time.
- Whenever convergence occurred eve never broke the encryption regardless of how much time was given.
- Architecture alternations had a minor effect on performance.



Vulnerabilities

- Tree parity machine can be compromised by an adversary with enough computing power and memory. Though this requires exponentially more for each node. We tested a geometric attack (most effective single node attack) and it fails to get enough of the key to be significant. Swarm attacks work but they require a close enough random initialization to work (enough guesses will work).
- The idea is similar to the random walk of the tree parity machines. Since the attacker occasionally breaks the encryption, it is entirely possible that a swarm of adversaries (with enough computing resources) could break the encryption.

Conclusion

- We have created the first method that uses purely neural networks to securely communicate over a public channel.
- Training time is approximately 30 minutes when training occurs within one machine. Our expectation is that this would be significantly longer of a network.
- Once the networks have been trained, encryption/decryption time is short enough for practical applications ($<<$1 second).
- Going forward, applying newly discovered methods for stabilizing GANs is probably the best way to guarantee convergence.
- Repositories:
  - https://github.com/msokoloff1/Adaptive-Communication : Final code
  - https://github.com/msokoloff1/Adversarial-Neural-Cryptography : Exploring the original paper
  - https://github.com/msokoloff1/ANN-Public-Key-Cryptography : Tree parity machine tests