Matt Sokoloff
MSCS 630
2 April 2017

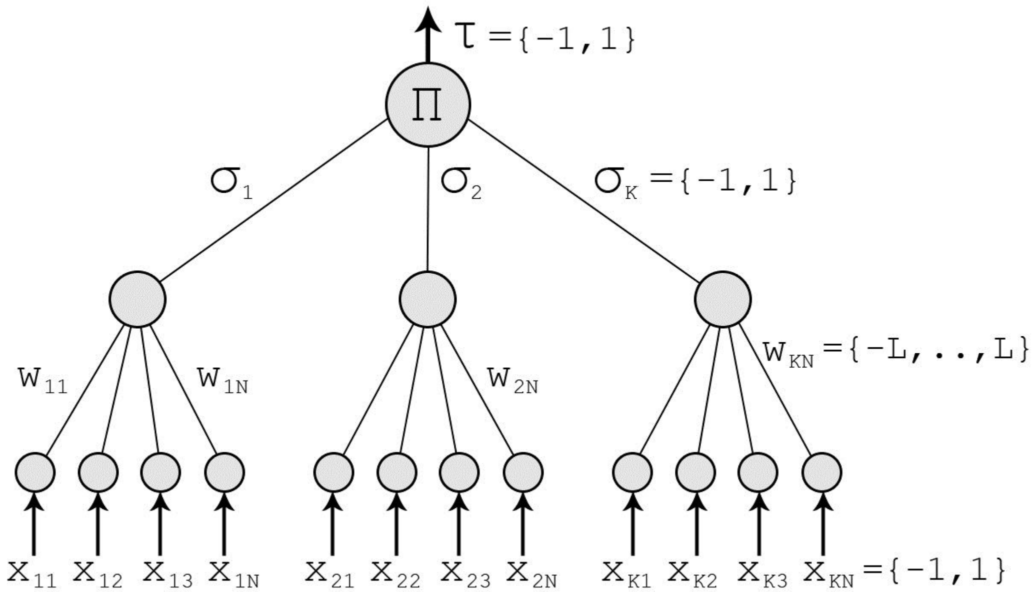
Milestone

The overall purpose of this project is to explore cryptographic applications of neural networks. Specifically, improvements to the google brain paper, "Learning to Protect Communications with Adversarial Neural Cryptography".

Due to the impending obsolescence of current cryptographic methods due to quantum computing, alternative methods must be developed. Unfortunately, there is little successful research in non-conventional cryptographic methods. One technology that has shown a little bit of promise is the tree parity machine. The idea behind this is that two parties can communicate over a public channel securely by working together. Since the two parties can influence each other, they can synchronize faster than a third party which has no influence. While there are possible attacks on tree parity machines, the concept of mutual learning appears promising.

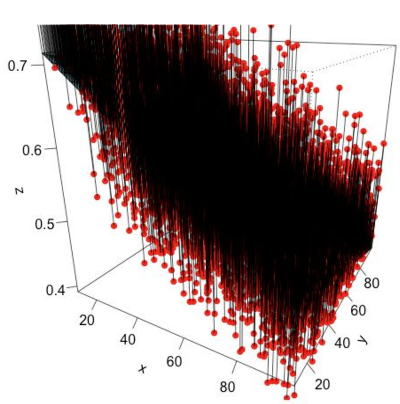
In this project, our focus has been on improving the architecture of the original adversarial paper in order to either reduce the computation necessary to achieve private communications and increase the reliability of converge. Additionally, the Google Brain paper currently requires a private key to be known by both parties at each time step which makes it dependant on cryptographic methods that do not rely on neural networks. Our experiments involve adjusting the original architecture and entering a neural public key exchange method for generating keys. We removed the dependency on a shared private key for each iteration.

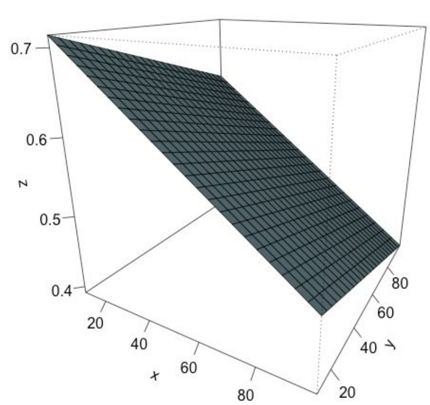
The first experiments were based around tree parity machines. The goal was to find out the optimal method for exchanging a key over public channels. The overall architecture simulates a key exchange over a public network. There are two tree parity machines with publicly predetermined architectures. This includes the number of inputs per hidden node (N), the number of hidden nodes (K), and the max absolute value for the discrete weights. In our experiments, we introduce a third network which simulates an attack on our two synchronizing networks. We use the hebbian update rule to synchronize the weights ($\vec{w}_i = \vec{w}_i + \vec{x}_i \tau_a \sigma_i \tau_a \tau_b$), where $\vec{w}_i$ is the weight vector connecting to the ith hidden unit, $\vec{x}_i$ is the binary vector input to the ith hidden unit, τ_a is the output of the network that we are updating, σ_i is the activation of the ith hidden unit, and τ_b is the output of the network that we are synchronizing with. In this case, the network architectures, the input vectors, and the output values are known publicly. The state of the weights is the private information that each network keeps to themselves. Over time, these networks will synchronize.



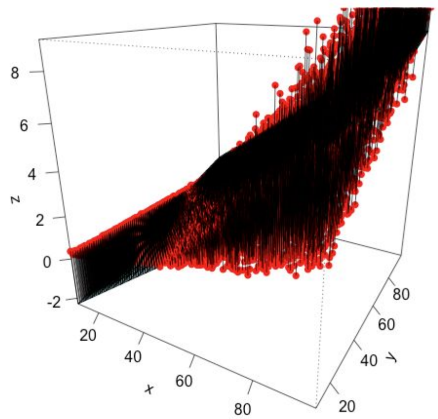
$$\sigma_i = sign(\vec{w}_i \cdot \vec{x}_i)$$
$$\tau = \Pi \sigma_i$$

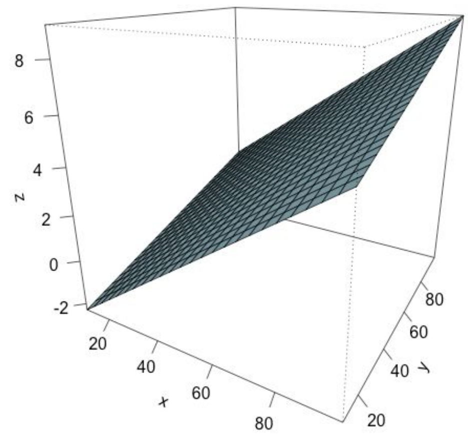
We found that the number of iterations required remains relatively constant regardless of the N chosen. However, increasing N added significant computational overhead. On the other hand, increasing the number of hidden nodes, K, made the model significantly more robust. While, additional input nodes has minimal effect on performance, if the number of hidden nodes exceeds the number of input nodes per hidden node by a factor of 50 then performance begins to suffer. We were able to find a balance of settings to prevent a geometric attack from being effective. The first two charts below show the relationship between number of hidden nodes (x), the number of leaves per hidden node(y), and their relationship to the adversary's synchronization percentage (z). The chart on the left contains the original points and shows that there is significant variation in the level of synchronization. The chart on the right contains the best fit hyperplane. This shows that the number of nodes is more significant in thwarting the adversaries efforts than the number of leaves per node. However, the number of leaves still has a meaningful impact.





The charts shown here demonstrate the time required for synchronization (z) as a function of the number of hidden nodes(x) and the number of leaves (y). While we are using a hyperplane to make a clear image, the relationship is nonlinear. The run-time grows exponentially with the number of nodes.



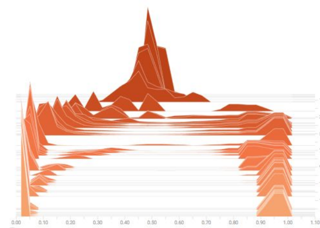
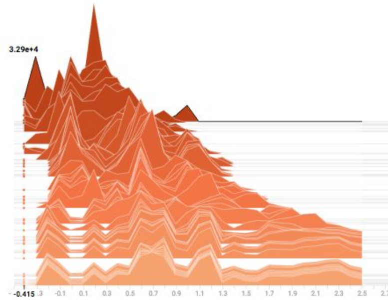
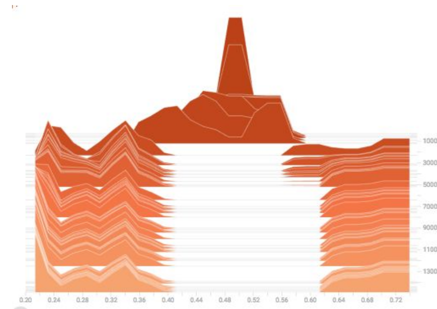
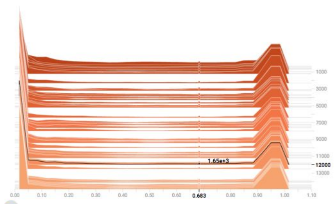
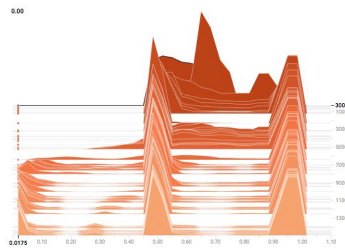
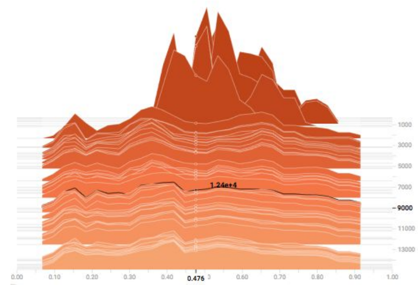
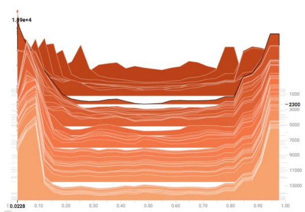
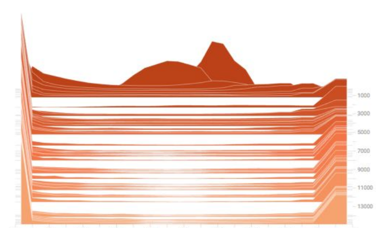
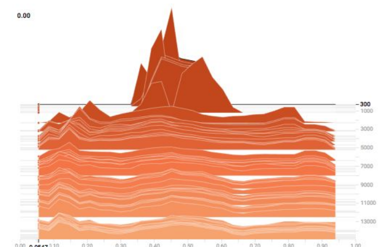
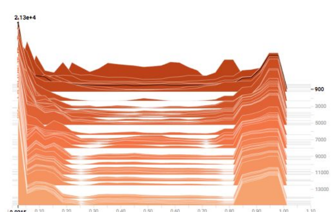
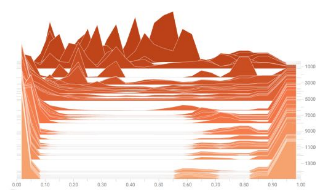
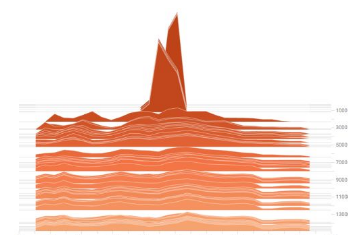
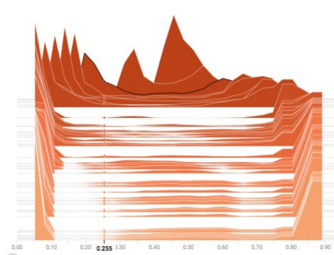
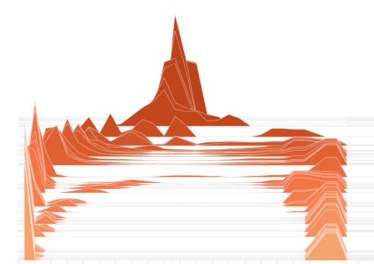
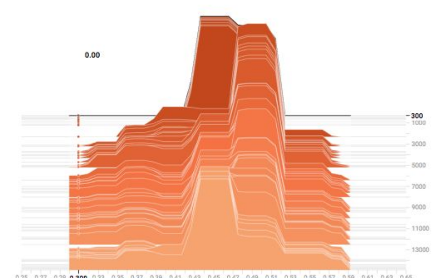


After obtaining an understanding of the tree parity machine dynamics, we set up experiments related to the Google Brain paper. First we reconstructed the architecture. Three neural networks which we refer to as Alice, Bob, and Eve. Alice is tasked with adjusting her weights such that Bob can interpret the message while Eve cannot. Eve and Bob's goal is to be able to read Alice's message. The original version of this model cannot be trained over a public channel because it requires a new private key for each iteration. The paper states that a fully connected layer followed by four subsequent convolutional layers is the most effective architecture. They then use gradient descent applied via the adam optimizer in order to minimize each agent's loss. Since the key and inputs are binary values, Alice and Bob seek to make Eve's reconstruction as close to random as possible. For every three weight updates, Eve's weights are updated twice and Alice and Bob's weights are updated once. This ensures that Alice and Bob are truly encrypting the message and not just staying one step ahead of Eve.

$$Loss_e = \frac{\sum_{n=1}^{N} |output_e - correct|}{N}$$

$$Loss_b = Loss_a = \frac{\sum_{n=1}^{N} |output_b - correct|}{N} + \frac{(\frac{messageLen}{2} - Loss_e)^2}{(\frac{messageLen}{2})^2}$$

We were able to achieve comparable results to the paper. While we are still analyzing our results, we have a few preliminary findings. First, the architecture is more flexible than the paper made it appear. We were able to make a fully convolutional network perform at a reasonable level. We also found that if we changed the location of the fully connected layer there was no effect on performance. Improvements to the model were achieved by increasing the weight applied to the second term of Alice and Bob's Loss (Eve's reconstruction). After 20 test runs, we were able to achieve what the original paper described as acceptable consistently. Our original implementation and Google's paper did have success every time. Furthermore, we were able to achieve this in a truly public environment. Our model combined the tree parity machine with the generative adversarial network. The trees would synchronize before the adversarial network began training. The weights of the tree would be converted to a five bit binary number and would be combined to create the key for each round. Then the seed for randomly selecting the weights would be made public. Our last difference from the paper is that we are using Tensorboard to analyze the weight distributions. Our initial analysis has found that the network creates uniform distributions of activations in order hide the message. When comparing the activations of the Bob and Eve network we see that the reconstruction process differs significantly. It is evident that Eve is uncertain in her reconstruction when looking at her output activations which are frequently around 0.5.

Layer	Alice	Bob	Eve
Fully Connected			
Conv 1			
Conv 2			
Conv 3			
Conv 4 (Output)			

Overall, we have created an improved model over the google brain paper. We have solved the problem of needing a private key to train over a public channel. Unfortunately, there has been no improvement to the convergence properties of the algorithm. Recent advances in methods for training generative adversarial networks makes this an interesting future research project.

Companion Code:

Code For Tree Parity Experiment:
https://github.com/msokoloff1/ANN-Public-Key-Cryptography

Our copy of the original Google Brain implementation:
https://github.com/msokoloff1/Adversarial-Neural-Cryptography

Our adjustments
https://github.com/msokoloff1/Adaptive-Communication

References:

Image: https://en.wikipedia.org/wiki/Neural_cryptography
https://openreview.net/pdf?id=S1HEBe_Jl
https://arxiv.org/pdf/cond-mat/0612537.pdf
https://pdfs.semanticscholar.org/6637/78bfab4317fd6d257e0640b30000f3c7d7b9.pdf
http://www.cs.bilkent.edu.tr/~guvenir/courses/CS550/Seminar/neuralcrypt.pdf