# Applications of Adversarial Neural Cryptography

Jason Liang, Evan Tey, Brandon Zeng

## 1 Introduction

Neural networks, especially in the form of deep neural nets, have been widely applied to many areas of traditional machine learning, from image recognition to natural language processing. As the number of applications of neural networks continues to grow, research has shown that systems of neural networks are fully capable of protecting their communications in order to maintain secrecy. We replicate and extend the results of "Learning to Protect Communications with Adversarial Neural Cryptography" by Martin Abadi and David G. Andersen [1]. In particular, we explore different configurations and structures which are modifications of Abadi and Andersen's original architecture with the aim of achieving a more reliable cryptosystem.

Cryptography, in general, is concerned with methods of encrypting information to ensure its confidentiality from a foreign party. So, a method is described as successful if an adversary has a low probability of successfully gaining information in polynomial time. Here we use neural network representations of all parties – the encrypter, decrypter, and attacker – to show that neural networks are reliably capable of learning how to encrypt and decrypt information without any preconceived notions about successful mechanisms.

Abadi and Andersen provide a working example of such a neural network setup that can successfully learn some encryption and decryption scheme. This configuration and structure, however is only successful $\frac{1}{3}$ of the time at best. We push this boundary by varying configuration variables as well as neural network structure in order to create a concrete implementation that is at least as sucessful. Abadi and Andersen mention multiple points of further research (e.g. experimenting with various minibatch sizes and learning rates) which we also explore.

It should be noted that neural networks are not made to perform well in cryptography. Specifically, neural networks are unable to compute XOR, a very common cryptographic method. Furthermore, many cryptographic functions are not differentiable so they cannot be optimized via stochastic gradient descent (as is the case in much of mahine learning). Despite this, we discover that neural networks are still capable of discovering encryption and decryption schemes without being taught specific algorithms to use.

In Section 2, we explain the neural network configurations and structure and how it allows for encryption and decryption schemes. We then identify our implementation and present performance results. Finally, in Section 3, we consider various extensions to the paper such as alternative architectures, smaller secret keys, and a threshold cryposystem intended to improve security reliability and computational feasibility.

## 2 Replication of Results

We first replicate of Abadi and Andersen's paper by adhering to their original neural network structure.

## 2.1 Network Construction

### 2.1.1 Objectives

As is standard in the cryptographic literature, we assume that there are three participants in this system: Alice, Bob, and Eve. Alice would like to send a message to Bob, but Eve could possibly intercept this message. So Alice and Bob will use a shared key to encrypt and decrypt the message which Eve does not have access to. Using this key, Alice and Bob's goal is to design an encryption / decryption system so a secret message can be sent without Eve capturing any information.

Figure 1 shows this more formally. Starting with plaintext $P$, Alice encrypts $P$ using key $K$ to ciphertext $C$. Bob decrypts $C$ using $K$ to $P_{Bob}$ in an attempt to reconstruct $P$. Meanwhile, Eve intercepts $C$ and decrypts it to $P_{Eve}$ with hopes to gain some information about $P$.

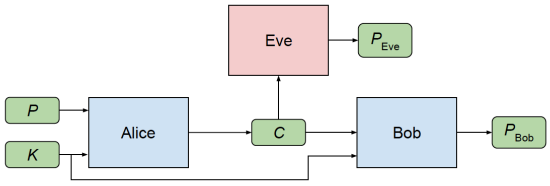


Figure 1: High-level representation of the architecture for a cryptographic neural network.

### 2.1.2 Network Configuration

Here, each of Alice, Bob, and Eve will be represented by neural networks (Figure 1).

Both the plaintext $P$ and ciphertext $C$ will be $n$ bits long. Additionally, we maintain Abadi and Andersen's choice to use an $n$-bit key $K$. This allows the possibility of learning a one time pad as there is a key bit for each message bit.

This means Alice will have an $2n$-bit input layer consisting of $P$ and $K$, and output a $n$-bit output layer of $C$. Bob will receive $C$ along with $K$ making a $2n$-bit input layer and will output his $n$-bit guess, $P_{Bob}$. Eve doesn't have access to $K$, so Eve only has an $n$-bit input layer (C) and an $n$-bit output layer with her guess $P_{Eve}$.

Rather than representing bits as 0's and 1's, we represent them as -1's and 1's so there is no multiplicative reduction from 0's. For hidden layer units, we use sigmoid functions, though for the output layer we use hyperbolic tangent functions – this is so the outputs of the neural nets are in fact in the range (-1,1). Note that this means $C$ will be $n$ floating point numbers between -1 and 1, where in reality Alice would be passing a message of actual bits.

### 2.1.3 Network Structure

The first layer of each neural network is chosen to be a fully connected layer. This is to allow the mixing of characters as is standard for many classic encryption methods. Specifically, this allows Bob and Alice to

mix or permute plaintext bits with key bits like many basic cryptographic methods, such as a Caesaer cipher or again one time pad. Following this fully connected layer is a series of convolutional layers. These allow further mixing as well as the reduction of $2n$ neurons to just $n$ in the output layer.

These layers can each be characterized by various filter sizes, strides, and output depths, all of which are labelled in Table 4. Since the fully connected hidden layer is at the beginning, it's mixing and permutations are propagated globally – any change in the input bit can affect any output bit.

The final output layers of Alice and Bob have $n$ neurons, which are transformed to the interval $(-1, 1)$ using the tanh function. Eve's neural network is similar in structure, except that the input layer only has $n$ neurons (where the neurons represent the $n$ bits of the intercepted encrypted message). We use sigmoids as the activation function for the hidden layers, and the tanh function as the activation for the output layers (in order to map the outputs to the interval (-1,1)).

| Type | Input dimension | Kernel | Stride | Outputs |
|---|---|---|---|---|
| FC | 32 | | | 32 |
| conv | 32x1 | 4 | 1 | 32x2 |
| conv | 32x2 | 2 | 2 | 16x4 |
| conv | 16x4 | 1 | 1 | 16x4 |
| conv | 16x4 | 1 | 1 | 16x1 |

Table 1: Hyperparameters of colorization network

## 2.2 Training

### 2.2.1 Training Structure

In order to train these neural nets, we separate the training out into two steps. First we train Alice and Bob so that they can learn some successful communication method. Then, we train Eve to try and learn Alice and Bob's communication method. Afterwards, we train Alice and Bob again to improve their encryption system, and again train Eve to try and crack their system. So if, for example, Alice and Bob learned a simple encryption technique like $rot13$, Eve would respond by learning it too and Alice and Bob would have to continue improving their system to keep Eve in the dark.

Note that this is a common setup in the field of generative adversarial networks. Eve is an adversarial neural network working against Alice and Bob. We continue to train Eve so that Alice and Bob have to face the "best" version of Eve.

We do this training in batches – 20,000 batches total with a batch size of 4096. We also use a learning rate of 0.0008. These values are taken from Abadi and Andersen.

Batch training is separated so Alice and Bob are trained for one minibatch. Eve is then trained for two minibatches because Abadi and Andersen report that this ratio allows Eve to be a strong adversary for Alice and Bob.

### 2.2.2 Loss Function

In order to conduct this training, we need the appropriate loss functions to ensure that the three networks learn towards their objectives.

Alice and Bob share an objective: to encrypt $P$ as $C$ and have Bob decrypt it successfully without Eve also decrypting it. Eve's objective is to decrypt Alice's message $C$ into its original form $P$.

Concretely, assuming $\theta_A$, $\theta_B$, and $\theta_E$ are the parameters of Alice, Bob, and Eve respectively, we let $C = A(\theta_A, p, k)$ be Alice's output, $P_B = B(\theta_B, c, k)$ be Bob's output, and $P_E = E(\theta_E, c)$ be Eve's output. To measure the difference between the plaintexts $p, p_B$, and $p_E$, we introduce the distance function $d$ which measures the L1 distance. In other words,

$$d(p, p') = \sum_{i=1}^{n} |p_i - p'_i|$$

where there are $n$ floating point numbers in each plaintext which range from -1 to 1.

The loss function for Eve is

$$L_E(\theta_A, \theta_E, p, k) = d(p, p_E)$$

We choose the parameters $\theta_E$ which minimize the expected loss, so the optimal Eve is found by

$$O_E(\theta_A) = \text{argmin}_{\theta_E} E_{p,k}[L_E(\theta_A, \theta_E, p, k)]$$

Similarly, we can define a loss function for Bob

$$L_B(\theta_A, \theta_B, p, k) = d(p, p_B)$$

And go on to define a loss function for Alice and Bob's communication

$$L_{AB}(\theta_A, \theta_B, \theta_E, p, k) = d(p, p_B) - d(p, p_E)$$

Intuitively, Alice and Bob want $p$ and $p_B$ to be as close to each other as possible so that Bob guesses the original plaintext, and they want $p$ and $p_E$ to be as different from each other as possible so that Eve doesn't guess the original plaintext.

Altogether, we can find the optimal Alice and Bob by minimizing $\text{L}_{AB}$

$$(O_A, O_B) = \text{argmin}_{(\Theta_A, \Theta_B)} L_{AB}(\Theta_A, \Theta_B).$$

Note that there is no global optimum and in general, there are many possible optima for Alice and Bob. This can easily be seen as there are many ways Alice and Bob could use the key to encode information in such a way that Eve has no knowledge.

Additionally, we must make a change to Eve's loss function. If Eve were to simply use the L1 distance loss function, Alice and Bob would try to create a communication system in which Eve was wrong at every single bit. This is not ideal because being wrong on every single bit would mean Eve has full knowledge of the message – she just has to flip all the bits to get the message. Thus, we use a quadratic function for Eve's loss in Alice and Bob's training

$$(N/2 - L_E)^2/N/2$$

so Eve's knowledge about the system is minimized, and to allow Eve to correctly guess a few bits at random.

We trained the network with $n$-bit messages, where $n = 16, 32, 64$.

In order to train the network, we train in minibatches of size 4096. Alice and Bob train for one minibatch to learn better parameters, while Eve trains for two minibatches. This is done in order to make Eve a stronger adversary without overfitting.

Alice and Bob's loss function has two components. The first is $d(p, p_B)$, which is the L1 distance between the plaintext and Bob's reconstructed plaintext. The second component is $(N/2 - d(p, p_E))/(N/2))^2$. This essentially says that Eve should not do better than random guessing (if the loss function were $N - d(p, p_E)$, Eve would simply flip her output bits in order to get an accurate decryption.

In order to evaluate the networks, after Alice and Bob achieve reconstruction error of less than 0.05 bits and Eve achieves at most 1-2 bits more than random guessing, we reset the Eve network and train it 5 more times. If Alice and Bob don't achieve the desired reconstruction error after 150,000 steps, the training is a failure. If any of the Eves achieve a good reconstruction error, then the training is also a failure.

## 2.3 Implementation

We use the TensorFlow library, which is a popular machine learning framework for deep learning written in Python. TensorFlow provides abstractions for dealing with the architecture of convolutional neural networks. Beyond that, we did not use any preexisting code.

As is the standard for neural net architectures, we initialize the bias terms to be identically zero. In addition, for each node, we initialize the weights so that they have a standard deviation which is proportional to the square root of the number of input nodes to that node, which is also standard. We use an Adam Optimizer from Tensor Flow and just personal computers for training (though results should not be machine-dependent).

## 2.4 Results

Figure 2 shows an example of a successful training of the neural network. The y-axis represents the average error per bit in the reconstructed message, and the x-axis represents the number of batches that each of the networks has been training for. In this run, both Bob and Eve were unable to learn anything better

than random guessing until batch 4000, after which both their reconstruction errors dramatically decreased. This continues until batch 6000, after which Alice and Bob manage to effectively use the additional shared information provided by their secret key to encode their message. This causes Eve's reconstruction error to increase, at first dramatically and then more slowly, up until training ends, while Bob's reconstruction error slowly decreases over the same time frame.

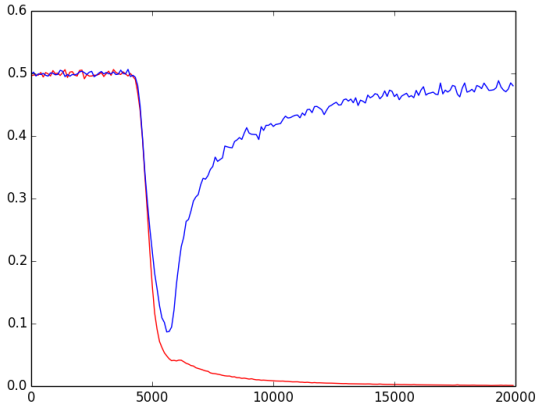


Figure 2: A successful training of the neural network architecture with 20000 batches and batch size 512. The red line represents Bob's average reconstruction error per batch, and the blue line represents Eve's average reconstruction error per batch.

To test the reliability of the neural network, we run 20 trials, where we judge a trial as succeeding if Bob's average reconstruction error is under 0.05 and Eve's average reconstruction error is above $7/16 = 0.4375$ (as discussed in [1]).

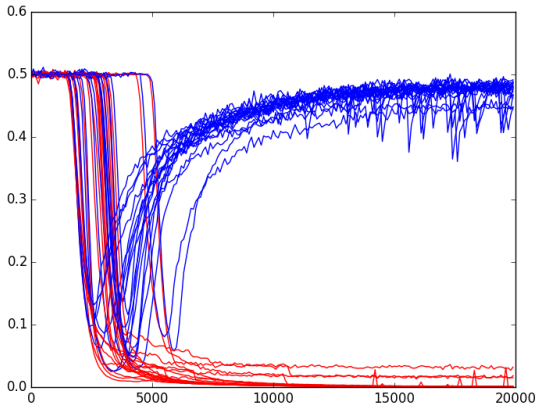


Figure 3: 20 trials of training the neural network architecture

We observe a success rate of 85%, with only slight variance between trials (see Figure 3).

# 3 Extensions

## 3.1 Experimentation with architectures

Now, with this replicated cryptosystem from Abadi and Andersen, we vary the neural network configuration and structure in order to see how robust this setup is.

### 3.1.1 Learning Rates and Batch Sizes

First, we varied the learning rate and minibatch sizes to alter how much Alice, Bob, and Eve are learning.

Table 2 shows the results of various learning rate on a fixed minibatch size. We see that for extremely high learning rates as well as for extremely low learning rates, Alice and Bob are not successful at creating a secure encryption system. This is because either Eve learns so quickly that she cracks Alice and Bob's scheme or because Alice and Bob don't learn fast enough to devise a complex system. Figure 4 shows what these graphs look like with the averages of 20 trials. One specific point of interest is that using a learning rate of .008 does in fact mean that Bob and Alice succeed more quickly – reaching success at 9900 batches while using a learning rate of .0008 doesn't succeed until 12100 batches on average.

| Learning Rate | .00008 | .0008 | .008 | .08 | .8 |
|---|---|---|---|---|---|
| Success Rate | 0 | .90 | .33 | 0 | 0 |
| Avg Success Iter | - | 12100 | 9900 | - | - |

Table 2: Effect of learning rate on success rate and iterations until success

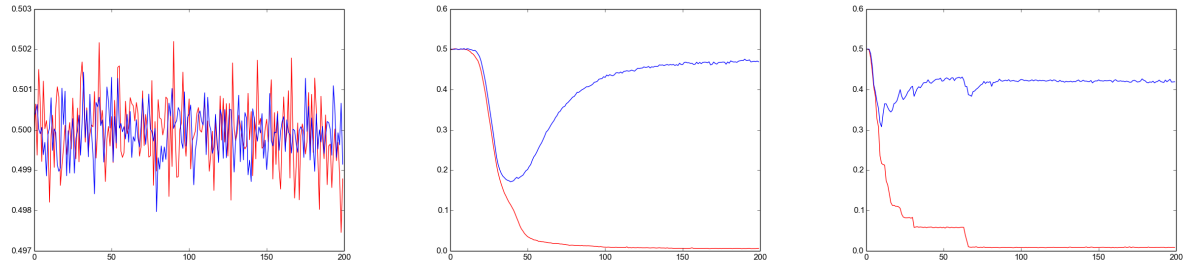

Figure 4: Training minibatch 256 with learning rates of .00008, .0008, .008

We tried all batch sizes in $\{256, 512, 1024, 2048, 4096\}$ with learning rates in $\{.00008, .0008, .008, .08, .8\}$ and found that regardless of minibatch size, Alice and Bob achieved success predominantly for learning rates of .0008 and .008. One point of interest is that with increased minibatch size, all three Alice, Bob and Eve did begin to learn more quickly. For a minibatch size of 4096 and learning rate of .008, Alice and Bob only required 1400 batches to achieve success, however increasing the learning rate slightly to .08, resulted in Eve catching on to Alice and Bob, so they never reached success (see Figure 5).

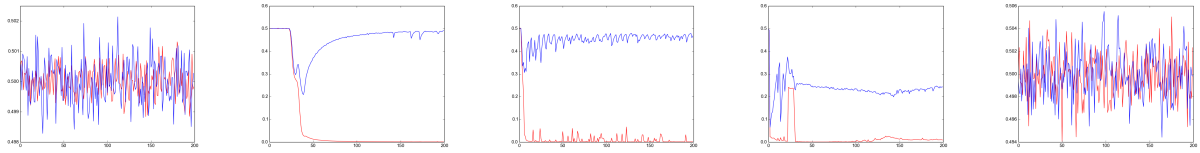

Figure 5: Examples of varying learning rate from .00008, .0008, .008, .08, .8 with minibatch 4096

### 3.1.2 Number of Layers and Stride Length

We next considered alternative neural network architectures with fewer layers and varying stride length. Specifically, we considered the following models:

| Type | Input dimension | Kernel | Stride | Outputs |
|---|---|---|---|---|
| FC | 32 | | | 32 |
| conv | 32x1 | 4 | 1 | 32x2 |
| conv | 32x2 | 2 | 2 | 16x4 |
| conv | 16x4 | 1 | 1 | 16x1 |

Table 3: Model A - using 4 layers

| Type | Input dimension | Kernel | Stride | Outputs |
|---|---|---|---|---|
| FC | 32 | | | 32 |
| conv | 32x1 | 4 | 2 | 16x4 |
| conv | 16x4 | 1 | 1 | 16x1 |

Table 4: Model B - using 3 layers with adjusted stride length

After running several trials, we observed the following results (Figure 3.1.2), with model A reaching a 75% success rate and model B reaching a 85% success rate. Surprisingly, the simpler model with fewer models performs slightly better, but not as well as the original model. We hypothesize that this may be due to variance in the trials, or insufficient training batches in model A. These results may change for larger message or key lengths, but currently, the original model seems to perform the best.

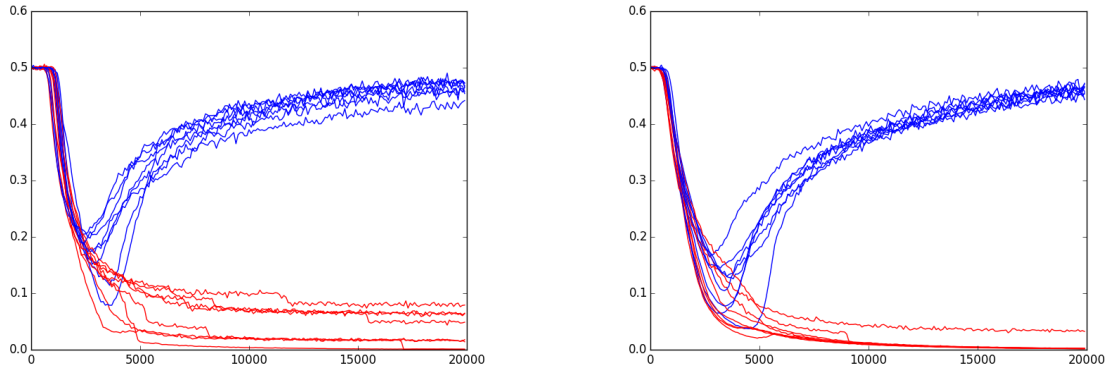


Figure 6: Training with models A and B

### 3.2 Smaller Secret Keys

We also experimented with using secret keys with lengths $m < n$. To accomplish this, we updated the current implementation by including a final fully connected layer with tanh activation and corrected dimensions to each of Alice's, Bob's, and Eve's neural networks, as summarized in Table 5.

| Type | Input dimension | Kernel | Stride | Outputs |
|---|---|---|---|---|
| FC | $(m+n)$ | | | $(m+n)$ |
| conv | $(m+n)$x1 | 4 | 1 | $(m+n)$x2 |
| conv | $(m+n)$x2 | 2 | 2 | $\frac{(m+n)}{2}$x4 |
| conv | $\frac{(m+n)}{2}$x4 | 1 | 1 | $\frac{(m+n)}{2}$x4 |
| conv | $\frac{(m+n)}{2}$x4 | 1 | 1 | $\frac{(m+n)}{2}$x1 |
| FC | $\frac{(m+n)}{2}$x1 | | | $n$x1 |

Table 5: Updated hyperparameters of colorization network

Starting with message length $n = 16$, we trained the model with key lengths $m = 4, 8, 12$, gathering the training results presented in Figure 7. As before, we use 20000 batches and batch size 512, with the red lines representing Bob's average reconstructions error per batch and the blue lines representing Eve's average reconstruction error per batch.

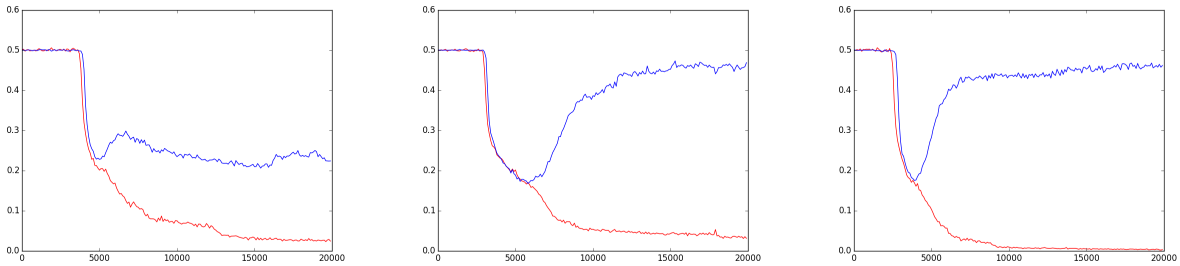


Figure 7: Training with $(m, n) = (4, 16)$, $(8, 16)$, and $(12, 16)$

We note that reducing key length results in Bob's average reconstruction errors increasing slightly, while Eve's average reconstruction errors decrease. It appears $(m, n) = (8, 16)$, $(12, 16)$ still allow for successful encryption and decryption, as Bob's reconstruction error is under 0.05 and Eve's reconstruction error is above $7/16 = 0.4375$, but the reliability is unclear. To test the reliability of the neural network models, we ran twenty trials for each $(m, n)$ combination, gathering the training results presented in Figures 3.2, where we overlay the trials on one graph and batches are labeled by the 100's.

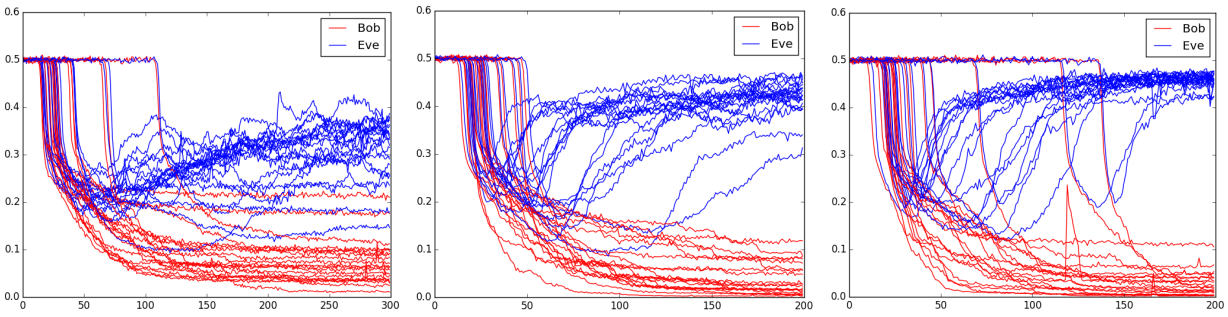


Figure 8: 20 trials of training with $(m, n) = (4, 16)$, $(8, 16)$, and $(12, 16)$

There appears to be much more variance in average construction errors in $(m, n) = (4, 16)$, and $(8, 16)$. Indeed, while our initial trial for $(m, n) = (8, 16)$ succeeded, there appears to be many other failed trials. We confirm this by computing Bob's and Eve's average final reconstruction errors, as well as trial success rate, which are presented in Table 6.

| n | m | Bob | Eve | Success Rate |
|---|---|---|---|---|
| 16 | 4 | 0.074 | 0.308 | 0% |
| 16 | 8 | 0.039 | 0.418 | 55% |
| 16 | 12 | 0.027 | 0.479 | 75% |

Table 6: Results of running 20 trials for each $(m, n)$ combination

As expected, shorter keys lead to decreased performance of our neural network architecture as it reduces the amount of information that can be encoded. However, depending on the key length used and desired secureness, they can still provide for reasonable encryption and decryption between Alice and Bob (e.g., using $(m, n) = (12, 16)$).

### 3.3 Threshold Cryptosystem

At this point, our implementation of a neural cryptography system is simply a technical curiosity. As the original paper claims, and as our experiments also confirm, many of the training runs result in networks where either Bob is unable to consistently reach message reconstruction errors that are below a certain threshold (which means that Bob is unable to decrypt Alice's message), or where Eve is able to consistently achieve message reconstruction at a rate that is measurably better than random guessing (which means that Eve is able to intercept and decrypt Alice's message).

For practical and real-world cryptographic applications, both of these scenarios result in fatal failure of the system. Encoded messages are either unable to be successfully decoded by the receiver, or are able to be successfully decoded by an intercepting party. In this paper, we present extensions to handle these failures.

The modifications that we make draw primarily upon two ideas: ensembles from machine learning and threshold cryptosystems from cryptography.

### 3.3.1 Ensemble Learning

In machine learning terminology, ensemble learning refers to training many different models which when aggregated together in some fashion can achieve better performance than any individual model can by itself.

The classic example of ensemble learning are random forests, which are ensembles of decision trees 9r. The theory behind random forests is that by using bootstrap aggregation (bagging), many different decision trees are generated by repeatedly resampling with replacement from the training data. This decreases the bias of the model, allowing it to draw from a larger space of possible representations, while taking the most common decision as the final output of the model reduces the variance.

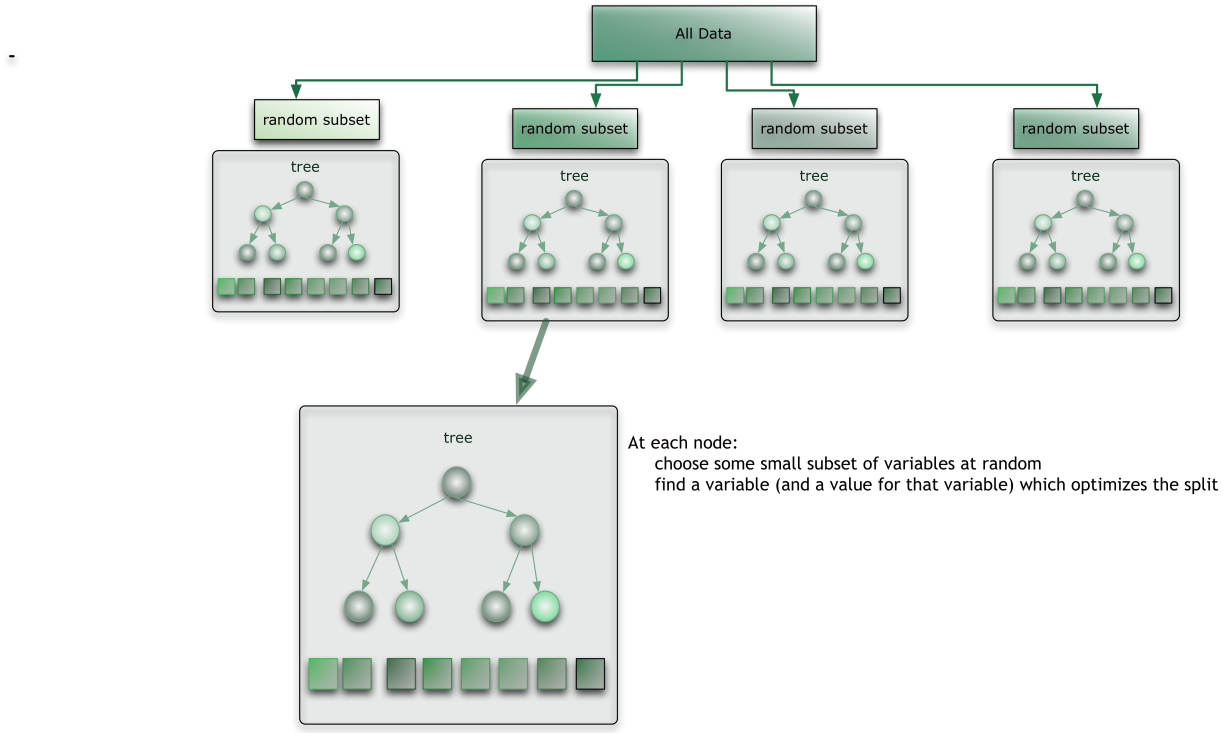


Figure 9: Example of an ensemble model: random forests Taken from `https://citizennet.com/blog/2012/11/10/random-forests-ensembles-and-performance-metrics/`.

This technique is applicable to neural cryptography because each of the randomly initialized neural networks has to be treated as a weak learner, since with random initialization it is impossible to tell with certainty if the network will train successfully (in experiments, failures occur 1/3 of the time) Thus, a possible method to increase the robustness of the network is to train an ensemble of them..

### 3.3.2 Threshold Cryptosystems

To protect some types of extremely sensitive information (nuclear launch codes, bank master passwords, etc.), it may be undesirable to distribute the master key to every participant. Doing so incurs the risk that there are many more avenues through which an attacker can obtain the key, and we would also like to obtain consensus before making such an important decision.

The concept of secret sharing evolved in order to solve this problem. A secret is separated into $n$ different "shares", each of which is distributed to a different "participant". In order to reconstruct the secret, $t$ shares are needed; if even $t-1$ participants come together to attempt to reconstruct the secret, they will be unsuccessful.

One of the earliest and simplest implementations of secret sharing is Shamir's Secret Sharing, which is based upon polynomial interpolation. In this scheme, the secret $p$ is encoded as the constant coefficient of a degree-$t-1$ polynomial, and the other $t-1$ coefficients are chosen randomly. Each participant is then given a distinct ordered pair of points on the polynomial curve (none of which have x-coordinate 0).

Since the polynomial has degree $t-1$, $t$ points (which correspond to $t$ shares) are needed to uniquely define it, and furthermore, if less than $t$ points are used to attempt reconstruction, $p$ is not uniquely defined. One drawback with this variant of Shamir Secret Sharing is that as an attacker gains access to shares, they can narrow down the space of possible secrets.

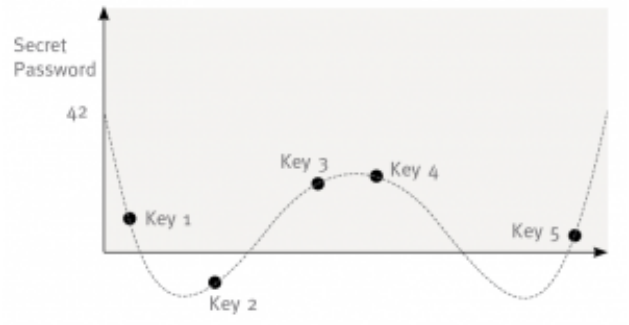


Figure 10: Shamir Secret Sharing

### 3.3.3 A Concrete Implementation

With the previous discussions in mind, we now move on to describing how we would implement a neural cryptography system in practice. In order to train an ensemble of $n$ neural networks, Alice and Bob first use a secure key-exchange protocol, such as Diffie-Hellman, to generate $n$ different symmetric keys. They then set up their neural network architectures to be exactly the same and train the networks as described in the Implementation section.

During this process, it is not necessary for the encrypted messages to be passed over a secure channel, as the previous analysis as assumed that Eve is capable of interception. In addition, Alice passes the correct plaintext messages over the insecure channel in order for Bob to compute his loss. As well, doing so does not compromise the integrity of the protocol.

After some number of batches (20000 seems to be a good number, judging from our previous experiments), Alice and Bob complete their training. At this point, Alice and Bob can tell if Bob has an acceptable reconstruction loss, but they know nothing about Eve's neural network. Eve could have succeeded in training a network which is capable of decoding the majority of Alice's messages, or Alice and Bob could have been successful in (independently) inventing encryption and decryption strategies.

Since Alice and Bob do not know for certain if their efforts have succeeded, they should use a secret sharing protocol. Each of the $n$ different neural networks is treated as a participant, and $n$ and the threshold $t$ can be adjusted in order to achieve the desired levels of accuracy. If $t$ is decreased, Alice and Bob are more

likely to be able to agree on a message, but Eve is also more likely to be able to decrypt a message. If $t$ is increased, Alice and Bob are less likely to agree on a message, but at the same time it will be less likely that Eve performs a successful decryption.

Secret sharing proceeds in a similar manner as in the Shamir Secret Sharing scheme, with the actual message to be passed being encoded as the constant term of the polynomial and the other $t-1$ coefficients being randomly chosen. $n$ random points on the curve are chosen and encoded as plaintext messages, and each message is passed to a separate network in the ensemble. Alice encrypts each message, sends it to Bob (along with the label of the corresponding neural network), and BoB decrypts all $n$ messages.

Bob is now left with $n$ points which approximately fit a polynomial of degree $t-1$. Bob can now approximately determine the correct polynomial using any least squares method, which will allow him to reconstruct the original message with high accuracy.

Meanwhile, Eve also uses her trained neural networks to attempt decryption, but many of her decrypted points lie very far from the correct polynomial. Thus, she will not be able to correctly reconstruct the message.

#### 3.3.4 Advantages of Neural Cryptography

Perhaps the greatest advantage of using neural networks to perform cryptography is that the exact algorithms used to perform encryption and decryption are opaque, even to users who are training the neural networks. Even if the neural network architectures are known, the weights change every time the system is trained.

On the other hand, attacks involving quantum computers and efficient factoring algorithms (if any are ever discovered) are capable of cracking many of the most popular protocols, including RSA.

An argument against using neural cryptography is that it will likely be more computationally intensive than today's techniques, especially when training and computing backpropagated gradients. However, this will be less of a problem when actually performing encryption and decryption, since addition, multiplication, and applying the activation functions are the only operations involved.

## 4 Conclusion

We conclude that neural networks can be used for securing communications, even though the learning does not require implementing specific cryptographic algorithms. The extensions we explored suggest further effectiveness of neural networks in cryptography through architecture modifications and a threshold cryptosystem.

## References

[1] "Learning to Protect Communications with Adversarial Neural Cryptography". Abadi and Anderson, October 2016. `https://arxiv.org/pdf/1610.06918v1.pdf`