Life of Alice, Bob and Eve with Neural Net

Learning to protect communications with adversarial neural cryptography

Vamshik Shetty Dec 10, 2018 · 14 min read

As a “human”, if there is one thing I know for sure about our species is that, when we discover or invent something, we try integrating it with technology that already exists, however bizarre it may be. Well, I guess that’s just part of being human. This paper Learning to protect communications with adversarial neural cryptography by Martin Abadi and David G. Andersen was the result of such curiosity.

The article is going to be a long read with a complete walk-through of the theory and side by side system’s implementation in python using TensorFlow. I advice you get a hot cup of earl gray before we begin.

What do we need to know ?

There are at least two topics that everyone here needs to have a clear understanding of:

1. Symmetric Cryptosystem

2. How are GANs Trained

They are both insanely broad topics. In the interest of time and attention span, I will give a brief introduction to both, which will be sufficient to follow through and if you are already familiar with them, feel free to skip right on ahead.

Symmetric Cryptosystem

Basic idea of symmetric cryptosystem can be easily explained by an age old example of three people named Alice, Bob and Eve. Before we start, we assume that Alice and Bob have shared a secret key. One day Alice wants to send a secret message to Bob which she encrypts using their shared secret key to create a ciphertext. After which, Alice sends this ciphertext to Bob through a channel, presumably insecure. Later when Bob receives this ciphertext, he uses the same shared secret key to decrypt it and reads the message. On the other hand when ciphertext was passed through the insecure channel it was intercepted by Eve but she wouldn't have been able to decrypt it because she didn't have the shared secret key. The key-point to note here is that even though the message was passed through an insecure channel it wasn't compromised. A question may now arise, regarding the origin of this shared secret key, which is an entirely different story which would take us into the world of asymmetric cryptosystem and we won't be looking into it for now.

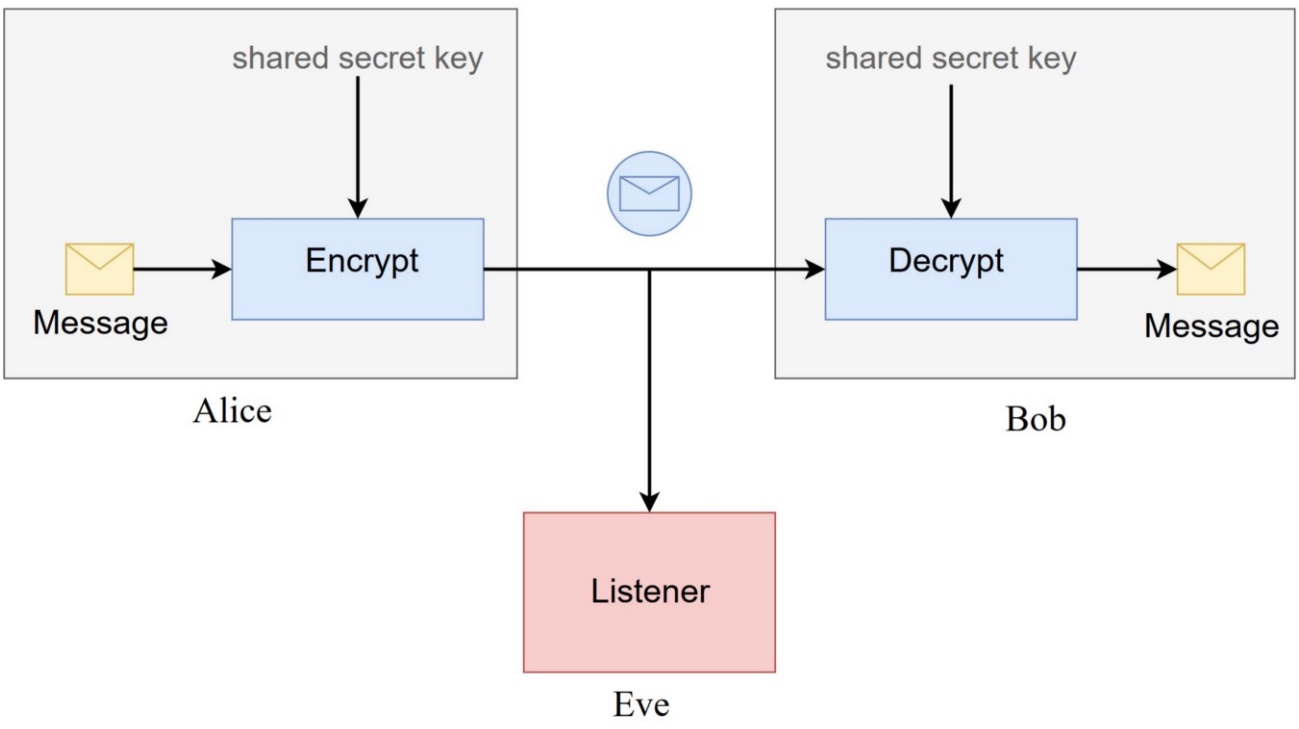

Figure 1 — Symmetric Cryptosystem

Above diagram not only gives a clear picture of what was just explained, but also visually explains the basic foundation of symmetric cryptosystem in one go.

How are GANs Trained

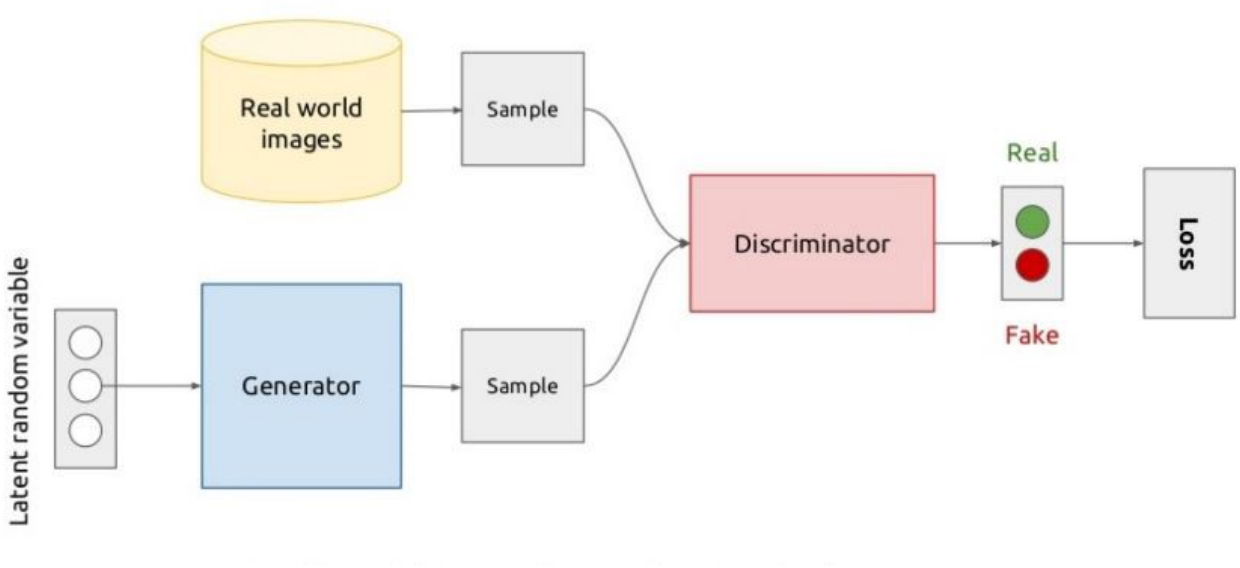

Figure 2 — ref

> *Adversarial: characterized by conflict or opposition*

Unlike common uses of neural network as a supervised learning algorithm where it learns to classify or predict something, in a system of Generative Adversarial Networks or GANs for short, there are usually two networks one named *generator* and another named *discriminator.* The job of generator model is to create its own fake sample. For example, an image; while the job of discriminator is to classify if the image it is seeing is a fake (i.e created by generator) or real (i.e its taken from the real world). In training, with each iteration generator tries to fool discriminator, while the discriminator tries not to get fooled by generator. Hence the name **adversarial.** In the event of generator fooling the discriminator, the discriminator is optimized by the loss to work better at distinguishing real vs fake samples. But if the generator is not able to fool the discriminator then the generator is optimized by the loss to create better samples.

If you are fascinated by how GAN works I would suggest you to read this article or if you want to see all the applications which are currently out there then you can check out this article.

Note: We are not going to use GAN but I used it in order to explain a different type of training where neural nets can be trained by competing them against each other which

is an important part of this paper.

Moving forward

Now that all of us have a basic idea of symmetric cryptosystems and adversarial networks, it's time to mix them together. In the paper authors have tried to use neural networks instead of using rigid symmetric-key algorithms such as AES, Triple DES etc. to encrypt or decrypt messages and soon we will talk about the adversarial part of it.

Where do we integrate Neural Network into this Cryptosystem ?

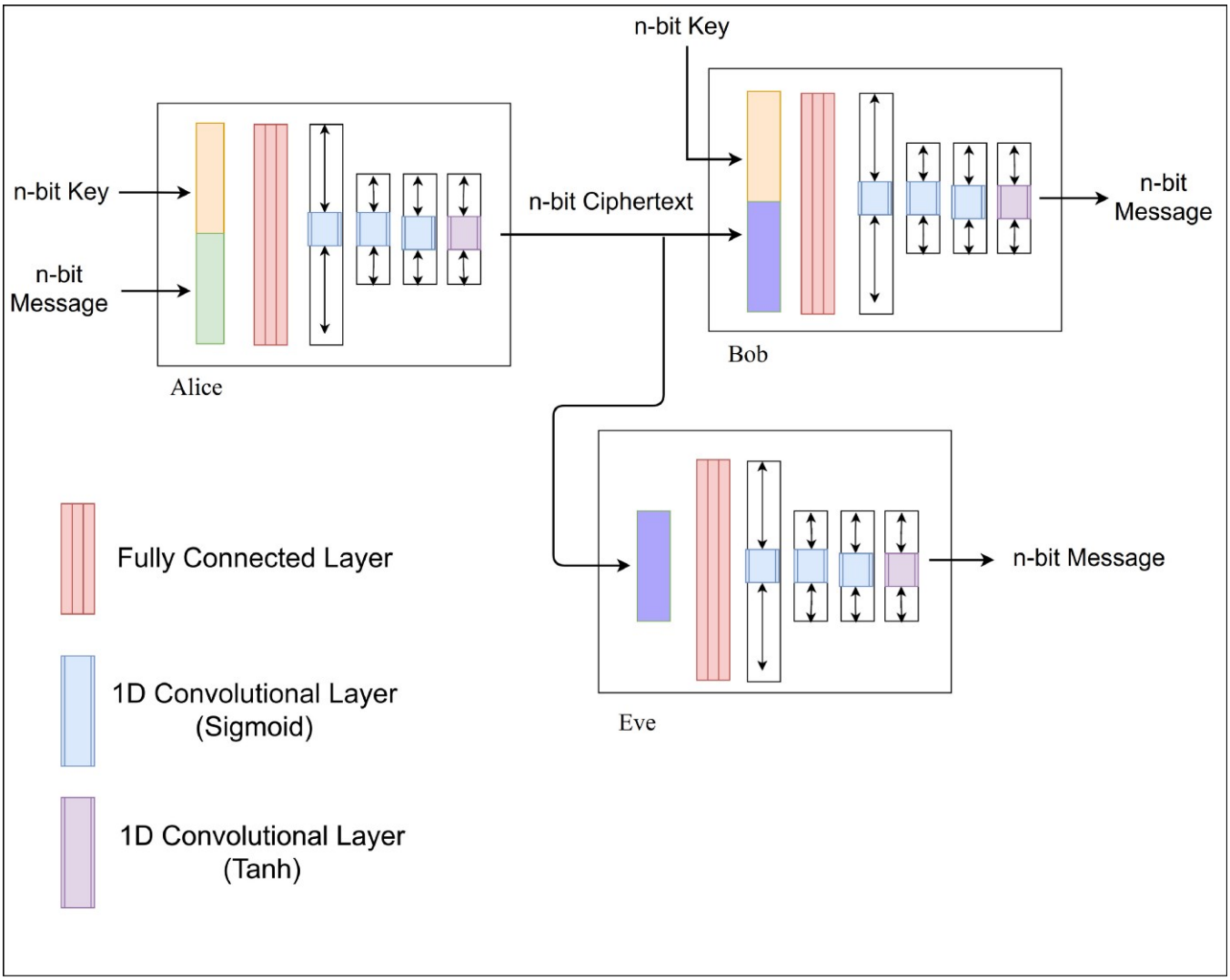

Figure 3

We train three neural networks namely Alice, Bob and Eve whose jobs are as follows:

- **Alice's job** is to take in n-bit message (encoded as a vector of -1 and 1 to represent 0 and 1 respectively) and n-bit key as input to output a n-bit cipher-text.
- **Bob's job** is to take the n-bit cipher-text created by Alice and use the n-bit key as input to reconstruct the original n-bit message.

- **Eve's job** is to take only the n-bit cipher-text and try to recreate original n-bit message.

Nothing much has changed compared to the original symmetric cryptosystem model except that the rigid encryption and decryption algorithm was replaced by neural nets. Here the main point to note is that we are creating adversarial neural nets in a sense where Alice trains to encrypt better than ever, using a shared key to create ciphertext, Bob trains to decrypt the ciphertext using a shared key and Eve trains to reconstruct the original message from ciphertext without knowing the shared key.

Why do we need Eve ?
As we are trying to make Alice and Bob learn a way to securely communicate data on their own, we can't feed them a data-set of message and cipher pairs created from available symmetric encryption algorithm. This would defeat its entire purpose of learning encryption and decryption on its own. Also, only having Alice and Bob in the model might result in Alice learning to pass the message as is, to Bob, without any form of encryption or with some form of encryption which is easily breakable hence we need Eve to act as an adversary against secure communication. A loop is created where Alice learns better form of encryption to communicate securely with Bob and Eve becomes better at breaking them until a point is reached where Eve is no longer able to keep up and isn't able to make sense of anything without the key. Now, we actually implement it and figure the math behind it.

Hopefully by now everyone here has **some** form of idea about our discussion and so, from this point on, we are going to continue the rest of the theory while also constructing its corresponding code and that's because at least for me, I have always felt more comfortable understanding things through their code, well owing to the fact that nature of words sometimes can be interpreted in a different way but code… code is absolute.

Let's try this for ourselves using TensorFlow

For people who are confident enough to directly dive into its code:

VamshikShetty/adversarial-neural-cryptography-tensorflow

Mimicking symmetric cryptosystem using adversarial networks

github.com

Creating Individual Models

There are few things that we need before we can start building individual models. For sake of simplicity we will be considering text & key to be of same size, let's say 16 bits which is the smallest size considered by the authors in the paper. For training, authors used a mini-batch of size ranging from 256 to 4096.

```
text_size   = 16
key_size    = 16
batch_size  = 4096
```

Given that, these are all the required parameters for now, we create two tensors to hold randomly generated messages and keys. We want to hold the entire batch size of 4096 at a time and since each text or key is of size 16, the shape of each tensor will be (4096, 16).

In the above snippet, Function named `model` is used to create the required neural nets. This function takes three input parameters where `collection` is the name of the model ('Alice', 'Bob' or 'Eve') so all the layers created for a model are under one scope. `message` is the tensor which holds the text while `key` is the tensor which holds shared secret key. One thing to note here, is that in the event of a key being given, the `if` condition evaluates to `True` and both message and key is concatenated to form a single tensor of size 32 bits (text_size + key_size) this scenario is valid for Alice and Bob. While for Eve, `if` will evaluate to `False` and only ciphertext will be given as a input.

In every model, other then the input tensor, the rest of the model is similar to each other. Each network has a Fully Connected (fc) layer of size 2n (n = number of message/key bits) which is then followed by four 1-dimensional convolutional layers.

The role of fully connected layers is pretty intuitive. Every cryptography method has some form of mix and transform, this layer enables message and key bits to affect the

output of every independent neuron in fc layer because each output bit can be a linear combination of all of the input bits.

FC layer is followed by a sequence of convolutional layers, the last of which produces an output of a size suitable for a plaintext or ciphertext. These convolutional layers learn to apply some function to groups of the bits mixed by the previous layer to take advantage of spatial locality. As the authors wanted locality i.e. the bits to combine, to be a learned property, instead of a pre-specified one like in rigid symmetric key algorithms.

Now comes the tricky part of getting the n-bit output from 2n-bit FC layer given that it is followed by 4 convolutional layers and also we don't have any FC layer of size n-bit in the end to produce n-bit output. To answer that question let's shift our focus to the 2nd Conv layer where we have set stride value to 2 and padding is set to 'valid'. Now if you are familiar with convolution then you must have already understood that at this point in model 2N units get halved and we get a tensor of n-bits.

Note: All the parameters such as filters, kernel size and strides are taken according to the original paper.

Why do we have Tanh as activation function in the last layer?
As we used -1 & 1 to represent 0 & 1 bit respectively, it would be better if our output also had similar representation and tanh function produces output in range of -1 and 1. Simple as that.

For the curious readers who are wondering what role are `Expand_dims` and `Squeeze` functions are playing; initially the input is of size (4096,32), when doing convolution there comes a concept called channels, similar to how we have 3 channels in an image i.e RGB. Given our data is only one channel we need the data to show similar characteristics. hence `Expand_dim` converts (4096,32) to data having one channel i.e (4096, 32, 1) and finally `Squeeze` function does the opposite.

Mimicking Symmetric Cryptosystem

Alice_out_ciphertext indicates the output of Alice model i.e n-bit cipher-text which Bob and Eve will try to decipher. You can see that this ciphertext is later given as input to Bob and Eve's model in line 8 and 11 respectively.

Now, to train Alice and Bob in such a way that they are able to communicate data without losing any information and also learn to encrypt and decrypt from scratch on their own such that Eve is unable to understand what Alice is sending to Bob. The answer is hidden in the formation of their loss function.

Loss Functions

Eve's loss is simple, Just try to minimize the error in difference of bits between Eve's output and original message using L1 norm/distance. We consider loss function to be average L1 norm loss of the given batch.

The real trick of introducing the **adversarial nature** in the system falls onto Alice and Bob's Loss, which is divided into two parts. First part is similar to Eve's Loss i.e try to minimize the error in number of bits, where Bob learns to reconstruct original message using the ciphertext and key. Like we already discussed, only trying to minimize the error in bits might lead to Alice passing the original message itself as ciphertext to Bob because it is easier not to learn some fancy encryption method to encrypt the original message and to reconstruct the same from the ciphertext and key.
How do we make sure that this doesn't happen, We introduce a loss replicating Eve as an eavesdropper. In short, better the Eve gets at breaking their encryption bigger Alice-Bob's loss gets. To formulate it:

This loss makes sure that Eve should not do better than random guessing i.e. this component is minimized when half of the message bits are wrong and half are right. In the training of Alice and Bob, we should not attempt to maximize Eve's reconstruction error because if we did and made Eve completely wrong, then Eve could be completely right in the next iteration by simply flipping all output bits. Authors choose a quadratic formula in order to place more emphasis on making Eve have a large error. Hence we

will need Eve's L1 loss to compute the 2nd part of Alice and Bob's loss function. We can see this written in TensorFlow in below mentioned code snippet.

let's create some random dataset

The data-set here is a pair of vectors to represent message and key having random bit values i.e 0 and 1, But 0 is never a good option to be given as a input to a neural network because it represents absence of a signal , so lets just encode 0 to -1 and 1 to 1.

Training begins

In below snippet, we use the culmination of all the functions that we have discussed by far. It starts from creation of a symmetric cryptosystem model, then we move on to create Alice-Bob and Eve Loss functions for Adversarial training of networks. Later to denote a optimizer used to minimize Alice -Bob communication loss by training tensors that belongs to Alice and Bob's model we create a variable named bob_opt. While on the other hand eve_opt does the same for Eve's model.

Note: Eve's model is optimized twice as compared to Alice-Bob communication model has mentioned in the original paper. (line 60 in the below code snippet)

Experiments & Results

ref — Evolution of Bob's and Eve's reconstruction errors during training. Lines represent the mean error across a mini-batch size of 4096

Author trained with text and key size of 16 bits for mini-batch of 4096. An ideal result would have Bob's reconstruction error drop to zero and Eve's reconstruction error reach 8 (half the bits wrong). In this experiment, both reconstruction errors start high. After a period of time, Alice and Bob start to communicate quite effectively, but in a way that allows Eve to improve its understanding as well. Then, around step 10,000 Alice and Bob counter
Eve's progress. By about step 15,000 the training goals are effectively achieved. The remaining steps merely increase Eve's reconstruction error slightly.

Note: At step 0, average error across batch of 4096 samples is 8 or close to 8. It can be thought of as, given the binary output we have 50% chance of being right hence 50% of 16 bits are right.

Some observations made by author:

1. key-dependent: changing the key and holding the plaintext constant results in different ciphertext output.
2. Single bit flip in the key typically induces significant changes in three to six of the 16 elements in the ciphertext.

Learning to selectively protect

By far we have seen a simple scenario of trying to protect the entire message from an adversary by encrypting it but what if message was made up of many parts which can be used to publish some correlated value related to this message and make it available to right individual but protect the remaining parts of the message from being calculated by an adversary.

Theory:

To test these ideas, Author constructed a data-set consisting of tuples of four values (A, B, C, D). Now a system was created and trained to use first three input (A,B,C) to output two values of D namely:

1. **D-public** which is the best possible estimate of D that **does not reveal** any information about the value of C. (produced by Alice)
2. **D-predict** ("true" prediction of D) which is the most accurate possible estimate of D. (produced by Bob)

The known assumption is already present i.e Alice and Bob share a secret key. Here Alice receives (A,B,C) as input to produces D-public in addition to a ciphertext. Bob uses D-public and ciphertext with shared secret key to producing an improved estimate of D i.e D-predict. While Eve takes in D-public and ciphertext to recover C acting as an **adversary** to the system.

Goal: Is to demonstrate that the adversarial training permits approximating D without revealing C.

> ***How to generate such a data-set i.e. (A, B, C, D) ?***
>
> *" A, B, C, and D are normally distributed floating-point values with a correlation matrix imposed upon them. The specifics of how we generate A, B, C, and D are standard: we compute the Cholesky decomposition LL of a 4x4 correlation matrix that specifies the pairwise correlations, we generate vectors of 4 normally distributed floating-point values, and then we multiply these vectors by L in order to generate values of A, B, C, and D."*

Training process:

Eve is Trained with the goal of minimizing the squared error of estimating C. Alice and Bob are trained not only with the goal of minimizing the squared error of estimating D-public and D-predict w.r.t D, but also by making sure that Eve isn't able to reconstruct C using the D-public & ciphertext which is achieved by minimizing their covariance value.

Results:

To check if the system was working, authors created a new model called "Blind Eve" which knows the distribution of C. Blind Eve predicts the value of C based on baseline information of its distribution. If error of Eve who has access to D-public and ciphertext becomes equal to error of Blind Eve then we can conclude that Eve is unable to extract meaningful information from D-public and ciphertext, same is show in the chart below:

ref — Training to estimate D while hiding C

- `Public Error` is error between D-public and D.
- `Bob Error` is error between D-predict and D.
- `Blue Dots` show Eve's average error improvement over Blind Eve but as training proceeds the average error becomes zero denoting that Eve isn't able to extract any useful information to recreate C.

What was the point of doing all this ?

> *"We do not prescribe specific cryptographic algorithms to these neural networks; instead, we train end-to-end, adversarially. We demonstrate that the neural networks can learn how to perform forms of encryption and decryption, and also how to apply these operations selectively in order to meet confidentiality goals"*
>
> **-** *Martin Abadi and David G. Andersen*

This quote from the ending of abstract, sums up everything from how neural nets can be trained to communicate securely with each other mimicking the system of symmetric cryptosystem to how they can hide selective information from an adversary while making sure right individual is able to extract the hidden information using a shared secret. All this without any prior domain knowledge or help of already existing rigid cryptographic systems.

Even though it might not be revolutionary enough to change the world of cryptography it is something which gives us new perspective of looking at deep-learning and cryptosystem together.

Thank you for reading this article, Happy learning !

References:

[1610.06918] Learning to Protect Communications with Adversarial Neural Cryptography

Abstract: We ask whether neural networks can learn to use secret keys to protect information from other neural...

arxiv.org

tensorflow/models

Models and examples built with TensorFlow. Contribute to tensorflow/models development by creating an account on...

github.com

If you believe that we are like minded people and should connect then you can find me on LinkedIn or you can email me at vamshikdshetty@gmail.com. If you have any thoughts, questions or feedback feel free to comment below I would love to hear from you.

Thanks to Ludovic Benistant.

Machine Learning Cryptography Deep Learning Adversarial Network Neural Networks