An Exposition of Neural Cryptanalysis of Classical Ciphers

David Z.

February 24, 2021

In this paper, we further explain how to leverage machine learning techniques described in `Neural Crytanalysis of Classical Ciphers` to crack classical ciphers. More precisely, we show how to use neural networks to predict the keys of ciphertexts encrypted by the Shift, Affine, and General Substitution Ciphers. We provide insight on the algorithms described in [1] by ...

Contents

§1 Terminology

A **corpus** is a collection of written texts. These will be a useful source of data for creating our own sets of data to train and test the models we create.

A **n-gram** is a continuous sequence of n items from a given sample of text or speech. In this context, these "items" will be letters from the alphabet. If n = 2, then the n-gram is a pair of letters. For $n > 1$, analyzing n-gram frequencies of texts will allow our models to get higher accuracies. The number of possible n-grams is 26^{n}.

§1.1 Ciphers

A **cipher** is an algorithm for performing encryption and decryption. The **keyspace** of a cipher is the set of all possible keys. Consider encrypting English plaintext. We will associate the letters a with 0, b with 1, c with 2, and so on.
is English

§1.1.1 The Shift Cipher

The **Shift Cipher** with key $s \in \{0, 1, 2, \cdots, 25\}$ has the encryption function, $E_s(x) = (x + s) \mod 26$ and decryption function, $D_s(x) = (x - s) \mod 26$, where x is a letter shifted by s characters in the alphabet. To encrypt a plaintext T, we apply $E_s(x)$ to each letter in T to receive a ciphertext C. Likewise, to decrypt a ciphertext C, we apply $D_s(x)$ to each letter in C to receive a plaintext T. The keyspace of the Shift Cipher is $\{0, 1, 2, \cdots, 25\}$, which has size 26.

§1.1.2 The Affine Cipher

The **Affine Cipher** with key a, b has the encryption function $E_{a,b}(x) = (ax + b) \mod 26$ and decryption function $D_{a,b}(x) = a^{-1}(x - b) \mod 26$, where a is relatively prime with 26 and a^{-1} is the modular multiplicative inverse of a modulo 26. To encrypt a plaintext T, we apply $E_{a,b}(x)$ to each letter in T to receive a ciphertext C. Likewise, to decrypt a ciphertext C, we apply $D_{a,b}(x)$ to each letter in C to receive a plaintext T. The keyspace of the Affine Cipher is $\{1, 3, \cdots, 25\} \times \{0, 1, 2, \cdots, 25\}$, which has size $12 \times 26 = 312$.

§1.1.3 The General Substitution Cipher

The **General Substitution Cipher** with key k, a permutation of $\{0, 1, \cdots, 25\}$, has the encryption function $E_k(x) = k(x)$ and decryption function $D_k(x) = k^{-1}(x)$, where $k(i)$ is the i^{th} element of k and k^{-1} is the inverse of k. The keyspace of the General Substitution Cipher is every permutation of the set $\{0, 1, \cdots, 25\}$, which has size $26! \approx 4 \times 10^{26}$.

§1.2 Neural Networks

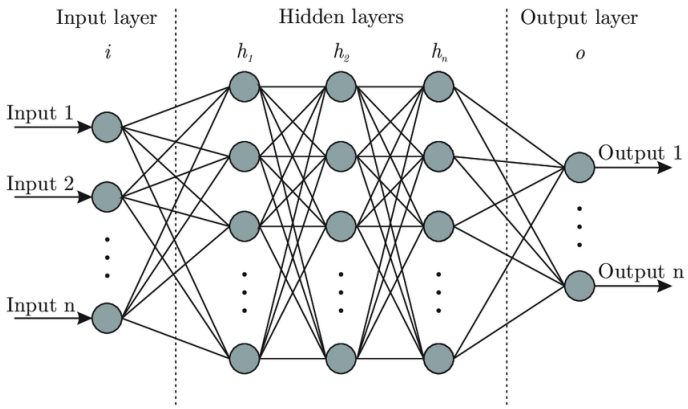

Figure 1: Labeled diagram of a basic neural network [2]

A **neural network** is a network of layers consisting of **neurons**, a node containing a number between 0 and 1 (or **activation**). The first layer of the neural network is called the **input layer**, which takes in input data. The last layer of the neural network is called the **output layer**, which outputs a value(s). Any layers in between the input and output layers are called **hidden layers**, which act as "breathing room" for the neural network to recognize patterns in input data.

Every neuron in a layer is connected to every neuron in the next layer by an edge, which has an associated value or **weight**. There is a **bias** value associated with each neuron, which represents the tendency for the node to be activated. Each neuron has an associated **activation function**, which applies a non-linear mapping to a real number; they are designed to map real numbers to a value between 0 and 1 to create an effective choice operator—the output value of the activation function becomes the activation associated with each neuron. The activation value in each non-input layer neuron is computed as a weighted sum of the weights and biases of the neurons and edges of the previous layers, then rescaled using the activation function.

Each neuron in the output layer can be associated with a certain class (e.g. cat or dog). The "goal" of classification for a neural network is to recognize patterns in input data and associate the entry of input data with a particular class. This is achieved by changing the weights and biases of each neuron and edge in the neural network. The larger the activation is in a neuron, or "brighter" the neuron is, the more the neural network thinks that a given set of input data is associated with that particular "class". A prediction by the neural network given some input data is essentially choosing the class associated with the neuron containing the highest activation value.

A **loss function** is a a function of activation values from neurons in the output layer and actual known computed values that outputs a number or loss. The **loss** indicates how well the neural network is performing based on a set of input data—the higher the loss, the worse the neural network is performing. **Backpropagation** is an algorithm ...

Training is the process of modifying the weights and biases of each neuron and edge in a neural network, which are initialized to random values at the start, based on a set of

input data.

An **epoch** is a run through a given training set.

One-hot encoding is...

A **sigmoid** function is . We will be using this as the activation function for nodes in the output layer.

A **rectified linear unit** (ReLU) is . We will be using this as the activation function for the nodes in the input and hidden layers.

In this paper, we will be using **feedforward neural networks**, a class of neural networks where connections between nodes do not form a cycle.

§2 Cracking the Shift Cipher

In this section, we will show two different ways to use a feedforward neural networks to crack the Shift Cipher. The Shift Cipher only has 26 possible keys — making it an easy task to find the key using brute force since only one key will "make sense." The purpose of using a neural network for this cipher is solely to demonstrate how to leverage machine learning techniques for more complex ciphers and by no means is this a better way for cracking the Shift Cipher.

§2.1 Method 1

To use a neural network, we need to train it to recognize patterns in a dataset. However, the dataset is not exactly clear, so we will have to generate it from a corpus of English plaintext. For this method, we will train a neural network to recognize what a plaintext is shifted by, given the letter frequencies of the ciphertext.

§2.1.1 Generating Data

a freq.	**b freq.**	$\cdots$	**z freq.**	**key 0**	**key 1**	$\cdots$	**key 14**	$\cdots$	**key 25**
0.50	0.25	$\cdots$	0.25	0	0	$\cdots$	1	$\cdots$	0

Figure 2: Entry of data for the string "aaaabbzz" and a key of 14

The first step is to get data to train and test the neural network. The input will be letter frequencies of a plaintext and the output will be the key. The key will be a vector of length 26 to perform one-hot encoding for each of the shifts. More precisely, the i^{th} entry of the key vector is equal to 1 if $i = s$ and 0 otherwise (using zero-based indexing).

Then, we can generate a dataset using a corpus as follows:

1. Take a plaintext, T, from the corpus of sufficiently many characters — [1] takes blocks of 100 words from the British national corpus
2. Pick a random integer $s \in [0, 25]$ and encrypt T by shifting each character in T by s many characters to generate a ciphertext, C
3. Compute the letter frequencies of C. An entry in the dataset will be an input/output pair of vectors of length 26: the input will be the frequency vector and the output vector will be the key vector
4. Repeat steps 1-3 until we have sufficiently many entries in the dataset

§2.1.2 Creating the Neural Network

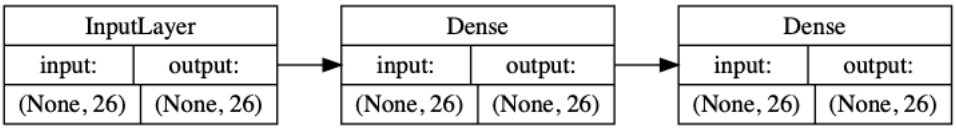

Figure 3: Model used to crack the Shift Cipher

Now, we will discuss the architecture of the feedforward neural network. We use a simple feedforward neural network consisting of 3 layers: an input layer, a hidden layer, and an output layer. Each of the layers consist of 26 neurons. For activation functions, we choose a sigmoid function for the nodes in the output layer and a rectified linear unit (ReLU) for the nodes in the other layers. Figure 3 shows the model we described in Keras, an open-source neural network library written in Python.

After defining the model, we can train the model through backpropagation. The model takes only a few epochs to train. This step only takes a few seconds, depending on the size of the dataset and computing power of your machine. Once trained, the model can be saved and used later.

§2.1.3 Applying the Model

Now that we have a fully-trained model, we may use the model to actually crack the Shift Cipher:

1. Given a ciphertext C, compute the letter frequencies of C, denote it as $\vec{f}$
2. Using the trained neural network, input the frequency vector f to get a key vector k.
3. The key s is the index of the max element in k. Namely, we choose the node that is most "brightly" lit or has the highest activation.

Upon successfully training the model, we can store the model locally to remove the necessity of training the model each time we want to crack the Shift Cipher. Cracking the Shift Cipher sing a pretrained neural network model is near instantaneous. When tested on an independent dataset, we get 100% accuracy. By no means is this an advancement on existing methods and anything less than 100% is not worth looking into. In the next section, we will look into a different method using neural networks, similar to the "is English" algorithm.

§2.2 Method 2

Now, we introduce a second neural network model to crack the Shift Cipher that will be useful later on for more complex ciphers. We will introduce the notion of a "goodness" value that measures how close a text is to English based on its frequencies. The primary differences between the first and second method are that in the second method, the output is a scalar value, rather than a key vector, and we will be using the neural network indirectly to crack the Shift Cipher.

§2.2.1 Generating Data

The first step is to get data to train and test the neural network. The input will be letter frequencies of a plaintext and the output will be a scalar value between 0 and 1, the goodness value. The closer the "goodness" value is to 1, the closer the plaintext is to English, in terms of letter frequencies.

Then, we can generate a dataset using a corpus as follows:

1. Take a plaintext, T, from the corpus of sufficiently many characters
2. Randomly pick 0 or 1. Half of the dataset will be English plaintexts and half of the dataset will be ciphertexts encrypted by the Shift Cipher
   a) If 1, then do not apply the Shift Cipher to T (shift of 0), $T = C$
   b) If 0, then pick a random integer $s \in [0, 25]$ and encrypt T by shifting each character in T by s many characters to generate a ciphertext, C
3. Compute the letter frequencies of C. An entry in the dataset will be a vector of length 26 as input and a scalar value as output: the input will be the frequency vector and the output will be 1 if the entry is an English plaintext and 0 if the entry is a ciphertext
4. Repeat steps 1-3 until we have sufficiently many entries in the dataset

§2.2.2 Creating the Neural Network

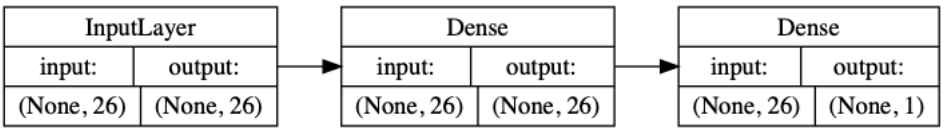

Figure 4: Model used to identify ciphertexts encrypted by the Shift cipher

Now, we will discuss the architecture of the feedforward neural network. We use a simple feedforward neural network consisting of 3 layers: an input layer, a hidden layer, and an output layer. The input layer and hidden layer consists of 26 neurons, but the output layer only consists of a single neuron. For activation functions, we choose a sigmoid function for the nodes in the output layer and a rectified linear unit (ReLU) for the nodes in the other layers. Figure 4 shows the model we described in Keras.

The model takes only a few epochs to train. This step only takes a few seconds and the model can easily identify whether a text is English or not, given the letter frequencies of the text. It is important to note that we cannot directly use this neural network to find the key since it outputs a scalar value.

§2.2.3 Applying the Model

We will now propose an algorithm to crack the Shift Cipher (by finding the key) using the neural network .

First, we will define auxiliary functions to be used in the algorithm:

- `FREQ`(T) is a function that takes in a plaintext, T, and returns the letter frequencies of T, a vector of length 26
- `DECRYPT`(C, n) is a function that takes in ciphertext C and an integer n and returns a decrypted plaintext by the Shift Cipher with key n
- `NN`($\vec{v}$) is a function that takes in a letter frequency vector, $\vec{v}$, and returns a scalar value representing the goodness of the text with letter frequencies $\vec{v}$—essentially a call to the pretrained neural network

Then, we can crack the Shift Cipher using the following algorithm:

Algorithm Crack Shift Cipher

```
let C be the given ciphertext
f⃗ ← FREQ(C)
max ← NN(f⃗)                          ▷ Store current max goodness value
key ← 0                               ▷ Store key associated with max goodness value

for i = 1 to 25 do
    C′ ← DECRYPT(C, i)
    f′ ← FREQ(C′)
    goodness ← NN(f′)
    if goodness > max then
        max ← goodness
        key ← i
    end if
end for

return key
```

The algorithm runs in $O(1)$ time since it is guaranteed to crack within 26 iterations. In practice, cracking the Shift Cipher using this method is near instantaneous as well. While this method may seem trivial, it solves the issue of having too many output nodes by introducing a "goodness" value. We will see in the next few sections that this architecture will be useful when attempting to crack more complex ciphers—those with larger keyspaces.

§3 Cracking the Affine Cipher

The Affine Cipher is a more complex cipher than the Shift Cipher as it has a larger keyspace. Namely, the keyspace is $\{1, 3, 5, ..., 25\} \times \{0, \cdots, 25\}$, which contains 312 elements. However, we may still follow a similar process used in cracking the Shift Cipher. The only difference is accounting for a slightly larger keyspace when generating the dataset.

§3.1 Generating Data

a freq.	**b freq.**	$\cdots$	**z freq.**	**key 1-1**	**key 1-2**	$\cdots$	**key 3-1**	$\cdots$	**key 25-25**
0.50	0.25	$\cdots$	0.25	0	0	$\cdots$	1	$\cdots$	0

Figure 5: Entry of data for the string "aaaabbzz" and a key of $3 - 1$

First, we will generate data to train and test the neural network. Again, the input will be letter frequencies of a long text and the output will be the key. However, the length of the key vector will be 312 instead—one entry for each possible key. Let a key for the Affine Cipher be of the form: $a - b$. Then, the possible keys are

$$\{1-1, 1-2, \cdots, 3-1, 3-2, \cdots, 25-1, 25-1, \cdots, 25-25\}$$

Using zero-based indexing, the zeroth index is equal to 1 if the key is $1-1$ and 0 elsewhere, the first index is equal to 1 if the key is $1-2$ and 0 elsewhere, the twenty-sixth indedx is equal to 1 if the key is $3-1$ and 0 elsewhere.

Then, we can generate a dataset using a corpus as follows:

1. Take a long text, T, from the corpus
2. Pick random integers $a \in \{1, 3, \cdots, 25\}$ and $b \in [0, 25]$. Encrypt T using the Affine Cipher with key a, b to generate a ciphertext, C
3. Compute the letter frequency of C. An entry in the dataset will be an input/output pair of vectors of lengths 26 and 312, respectively: the input is the frequency vector and the output is the key vector.
4. Repeat steps 1-3 until we have sufficiently many entries in the dataset

§3.2 Creating and Applying the Neural Network

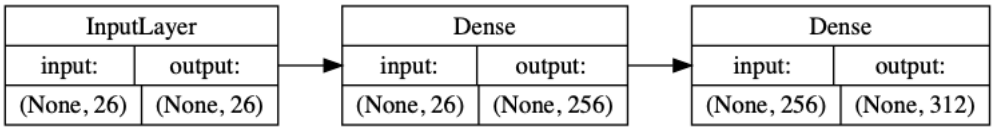

Figure 6: Model used to crack the Affine Cipher

The model we will use to crack the Affine Cipher will be similar to the one used in the first method to crack the Shift Cipher. The model will consist of 3 layers: an input layer, hidden layer, and an output layer. The input layer consists of 26 neurons, the hidden layer consists of 256 neurons, and the output layer consists of 312 neurons. For activation functions, we choose a sigmoid function for the neurons in the output layer and a rectified linear unit (ReLU) for the neurons in the other layers. Figure 6 shows the model we described in Keras.

It only takes a few seconds to both train the model and crack the Affine Cipher itself. To actually use the model, a similar procedure can be follow to that of the first method used to crack the Shift Cipher. Briefly, we input letter frequencies of a ciphertext, C, and use the model to get a vector of length 312. The highest activation corresponds to the key that the model predicts. When testing the model on an independent dataset, we were able to achieve 100% accuracy.

§4 Cracking the General Substitution Cipher

In this section. we will show how to use a feedforward neural network to crack the General Substitution Cipher. Compared to the Shift and Affine Ciphers, the General Substitution Cipher is a substantially more complex cipher as its keyspace is several orders of magnitude larger. The keyspace of the General Substitution Cipher is every permutation of the alphabet, which has length $26! \approx 4 \times 10^{26}$. It is impractical to have a neural network with that many nodes in a layer since you would quickly run out of space. We will proceed with a method similar to the second method used to crack the Shift Cipher.

§4.1 Generating Data

aaa freq.	**aab freq.**	$\cdots$	**azz freq.**	$\cdots$	**zzz freq.**	**is English**
0.50	0.50	$\cdots$	0	$\cdots$	0	1
0	0	$\cdots$	0.5	$\cdots$	0.5	0

Figure 7: Entries of data for the strings “aaab” and “azzz”

First, we will generate a dataset to train and test the neural network. Rather than use letter frequencies of a long text, we will use 3-gram frequencies to get a better sense of how close a text is to English. So, the input will be 3-gram frequencies of a long text and the output will be a scalar value between 0 and 1, the goodness value. The closer the “goodness” value is to 1, the closer the long text is to English, in terms of 3-gram frequencies.

The 3-gram frequencies will be a vector of length $26^3 = 17,576$—one for each possible sequence of 3 letters from the alphabet. The zeroth index of the vector will be the frequency of ”aaa,” the first index will be the frequency of ”aab,” the twenty-sixth index will be the frequency of ”aba.” Let $a = 0$, $b = 1$, $\cdots$, $z = 25$. Then, the frequency of the 3-gram $\sigma_1\sigma_2\sigma_3$ lies in index $26^2\sigma_1 + 26^1\sigma_2 + 26^0\sigma_3$, where $\sigma_1, \sigma_2, \sigma_3$ are letters in the alphabet.

We can generate a dataset using a corpus as follows:

1. Take a long text, T, from the corpus of sufficiently many characters
2. Randomly pick 0 or 1. Half of the dataset will be English texts and half of the dataset will be ciphertexts encrypted by the General Substitution Cipher
   a) If 1, then do not apply the General Substitution Cipher to T. The key is the alphabet (or $\{0, 1, 2, \cdots, 25\}$), so $T = C$
   b) If 0, then take a random permutation of the alphabet to be the key, $\vec{k}$, and encrypt T by using the General Substitution Cipher with key $\vec{k}$ to generate a ciphertext, C
3. Compute the 3-gram frequencies of C. An entry in the dataset will be a vector of length $17,576$ as input and a scalar value as output: the input is the 3-gram

frequency vector and the output is 1 if the entry is an English text and 0 if the entry is a ciphertext

4. Repeat steps 1-3 until we have sufficiently many entries in the dataset

§4.2 Creating the Neural Network

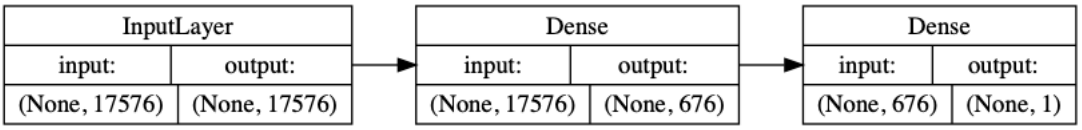

Figure 8: Model used to identify ciphertexts encrypted by the substitution cipher

We will use a model similar to the neural network used in the second method to crack the Shift Cipher. The model consists of 3 layers: an input layer, a hidden layer, and an output layer, which have 17.576, 676, and 1 node(s), respectively. For activation functions, we choose a sigmoid function for the nodes in the output layer and a rectified linear unit (ReLU) for the nodes in the other layers. Figure 8 shows the model we described in Keras.

The neural network takes a few epochs to train, which is a matter of seconds. Upon completing its training, the model is able to identify whether a text is English or not, given the 3-gram frequencies of a text, with 100% accuracy.

§4.3 Applying the Model

We will now propose an algorithm to crack the General Substitution Cipher using the neural network.

First, we will define auxiliary functions to be used in the algorithm:

- `FREQ3`(T) be a function that takes in a text, T, and returns the 3-gram frequencies of T, a vector of length $26^3 = 17,576$
- `DECRYPT`$(C, \vec{v})$ is a function that takes in ciphertext C and a key vector $\vec{v}$ and returns a decrypted text by the General Substitution Cipher with key $\vec{v}$
- `SHUFFLE`$(\vec{v})$ is a function that returns a random permutation of the vector $\vec{v}$
- `NN`$(\vec{v})$ is a function that takes in a 3-gram frequency vector $\vec{v}$ of length $17,576$ and returns a scalar value representing the goodness of the text with 3-gram frequencies $\vec{v}$—we feed $\vec{v}$ through the pretrained neural network

Then, we can crack the General Substitution Cipher using the following algorithm:

Algorithm Crack General Substitution Cipher

```
let C be the given ciphertext
let guess be an initial key vector
let keys[] and goodness[] be a lists of length MAXITERATIONS

for i = 0 to MAXITERATIONS - 1 do
    keys[i] ← guess
    C' ← DECRYPT(C, keys[i])
    goodness[i] ← NN(FREQ3(C'))

    for j = 0 to MAXSWAPS - 1 do
        new_key ← keys[i]
        Swap two random values in new_key
        C' ← DECRYPT(C, new_key)
        new_goodness ← NN(FREQ3(C'))
        if new_goodness > goodness[i] then
            keys[i] ← new_key
            goodness[i] ← new_goodness
        end if
    end for
end for

return key[k] where k is the index of the max element in goodness[]
```

The algorithm as as follows: given a ciphertext C, the algorithm begins with an initial key or "guess." We swap a random value in the current key to create a new key, then decrypts C using the new key. If the goodness value of the decrypted text using the new key is higher than the goodness value of the decrypted text using the previous key, we update the current key with the new key. In other words, if the text using the new key is closer to English than the text using the old key, we update the key. We use the neural network to decide if a text is close to English or not. This swapping routine is repeated for MAXSWAPS many times.

What exactly do we use for the initial "guess?" [1] discusses picking a random permutation of the alphabet as the initial guess. However, we found that it is not feasible at all to use, contrary to their results. So, we use their second proposed option: a key that matches the letter frequencies of English. Namely, we compute the letter frequencies of ciphertext C and map the highest frequency to the letter e, the next highest frequency to the letter t, and so on.

One problem that occurs when swapping random values in the key is that we can end up updating the current key with a new key that is actually further away from the actual key. This "false swap" results in the algorithm getting stuck and may not increase in "goodness" for subsequent swaps. In other words, the algorithm gets stuck at a "local maximum." This occurs because of letters that have similar frequencies in English, and it happens more often for texts of shorter length. To prevent this issue, we perform

the swapping routine more than once. More precisely, we perform the swapping routine MAXITERATIONS many times, storing the keys and max goodness values from each of the routines, then picking the key with the maximum goodness value out of all of the iterations. [1] finds that MAXITERATIONS = 10 and MAXSWAPS = 400 yields sufficient results.

We were able to reproduce similar results to [1] with a slightly slower runtime, most likely due to differences in computer performance. Although the full key is not always recovered, the predicted key is only off by one or two mappings, which can easily be corrected by inspection. Though this method does not completely crack the General Substitution Cipher (recover the correct key 100% of the time), we demonstrate a way to automate cracking the General Substitution Cipher.

References

[1] Focardi, R. & Luccio, F. (2018). Neural Cryptanalysis of Classical Ciphers. http://ceur-ws.org/Vol-2243/paper10.pdf

[2] Portwood, Gavin & Ragusa, Jean & Tano Retamales, Mauricio. (2020). Accelerating Training in Artificial Neural Networks with Dynamic Mode Decomposition.