LSTM Ciphertext Decryption

Josiah Wallis
University of California, Riverside
Group 1
jwall014@ucr.edu

Abstract

I present a proof of concept that supervised LSTM recurrent neural networks are more efficient at deciphering plaintext ciphers than typical cryptographic and cryptanalysis techniques. This is in the context of deciphering plaintext words without knowing neither the encryption algorithm nor its key. Given that keys come in the form of many variations, data types, and forms, the method proves powerful as modern decryption techniques require a myriad of bruteforce tests to crack an encryption algorithm. I present results of both validation accuracy during training of the LSTM, as well as test accuracy on unseen encrypted words. Additionally I present an LSTM's performance on a complex encryption algorithm conjectured in 2014 with fruitful results.

1 Introduction

In the field of cryptography, encryption algorithms are used to transform plaintext strings into strings of variable size. This process increases the security of information being transmitted across any number of communication channels. Encryption algorithms require not only an input, but a key to determine the nature of the encryption process. In other words, if an adversary knew both the encryption as well as its key, they could decipher the ciphertext an obtain potentially sensitive information. Usual decryption techniques involve noticing patterns in the ciphertext that are similar to well-known named ciphers, then it is necessary to search through a possibly infinite number of keys that fit the encryption algorithm as the keys could be numbers, words, even matrices. This process requires a great deal of overhead, testing, heuristics, and is overall a taxing method to approach more complicated ciphers that may not even be similar to currently known encryption algorithms. LSTMs, a type of neural network that works well with both sequential and temporal data, may be able to solve this problem. These networks are very popular in the field of natural language processing, time series analysis, and in sequential classification because of their ability to learn time-based dependencies, like if a word in a sentence is a name given both past and future words. Thus, LSTMs have the potential to detect character dependencies in permutation ciphers, character-ascii shifting with permutation ciphers, and potentially many other dependencies we may not be aware of in the space of ciphers.

2 Related work

After looking at recent literature within the last 10 or so years, I found that researchers did start using neural networks to decrypt cipher systems (5), but they were inefficient and required more training data and more computational power the more complex the ciphers got. As seen in the upcoming results section, many basic ciphers can be broken with a few number of epochs. But as past researchers found, composing ciphers together made the training process harder. Fully-connected networks worked for

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

some time, but as computational resources and neural network methodology improved, recurrent neural networks began to show up in research to decrypt more complex algorithms as they could handle the temporal nature of sequence data. Thus, given input data and a key, researchers developed RNNs and extension models that could break much more complicated encryption algorithms (6, 7). The fallback to these approaches, to my understanding of the papers, is that the key was required when training the models. I hope to improve upon their ideas in a supervised setting where the key is not known to the RNN.

3 Problem formulation

The goal is to be able to find a decryption mapping from a given English ciphertext, or encrypted word, to its correct plaintext English word. My approach uses a supervised method to implicitly learn the decryption scheme by looking at a large sample of encrypted English words as well as their correct decryptions as labels. In other words, given a set of ciphertext English words as inputs with their correct deciphered words as outputs, I want a neural network to learn the decryption algorithm without needing the encryption key. The network will be fed each word to the network character by character. Thus the predicted output labels to compare against are the characters of its corresponding deciphered word. I use validation accuracy as my initial benchmark, where the accuracy is based on character-by-character accuracy. For example, if "tstl" is mapped to "text," but the correct output was "test," the accuracy here would be 75%. The graphs shown in the experimental results section will be using character-by-character accuracy. For test results, I will be judging the final performance of the decryption scheme the model learned through word-by-word accuracy. If a single character is deciphered incorrectly, the entire decryption of the input word would count as a miss. Therefore an incorrect decryption is defined as a deciphered word where one or more characters is incorrect with respect to the actual desired word. This methodology gives rise to future directions.

4 Experimental results

For the dataset, I used one-hot-encoded vector representations of 150,000 plaintext English words. This means each each word would be of shape *(number of characters in word, one-hot-encoding)*. The training set is 120,000 words while the test set is 30,000 words. The input to each LSTM would be the 120,000 words enciphered with the labels being the original 120,000 words pre-enciphering.

I trained an LSTM for each encryption algorithm. That means that each LSTM was fed a list of words that were encrypted by a single encryption algorithm. For the 18 encryption algorithms I used, I produced 18 different LSTMs with slightly varying hyperparameters which can be found in the ipython notebook attached. For the model architecture, I used a stacked bidirectional LSTM with a dense layer as the final layer. Only two bidirectional LSTMs were stacked together, and I used a softmax activation for the dense layer.

The first main result was a proof of concept. I tested how well the LSTM performed on a composition of ciphers. In other words, one encryption was performed on the input set of words then another encryption was performed on its output. In Figure 2, the encryption in the algorithms was performed first followed by the encryption on the outside of the parentheses. In the legend, the keys are read from left to right. I trained an LSTM for 5 different polyciphers. We see that the LSTM performs well in this supervised setting, and the validation accuracy for the other 3 models not shown are also all over 95% training under 10 epochs.

The primary test I ran was on a set of encrypted words using an encryption that touts its strength and security (Rajput et al., 2014). It was a custom designed cipher that involved permuting characters, substituting characters, then finally unstructuring the characters to make it difficult to reverse engineer the encryption. As we see in Figure 2, the LSTM had essentially no problem breaking this encryption

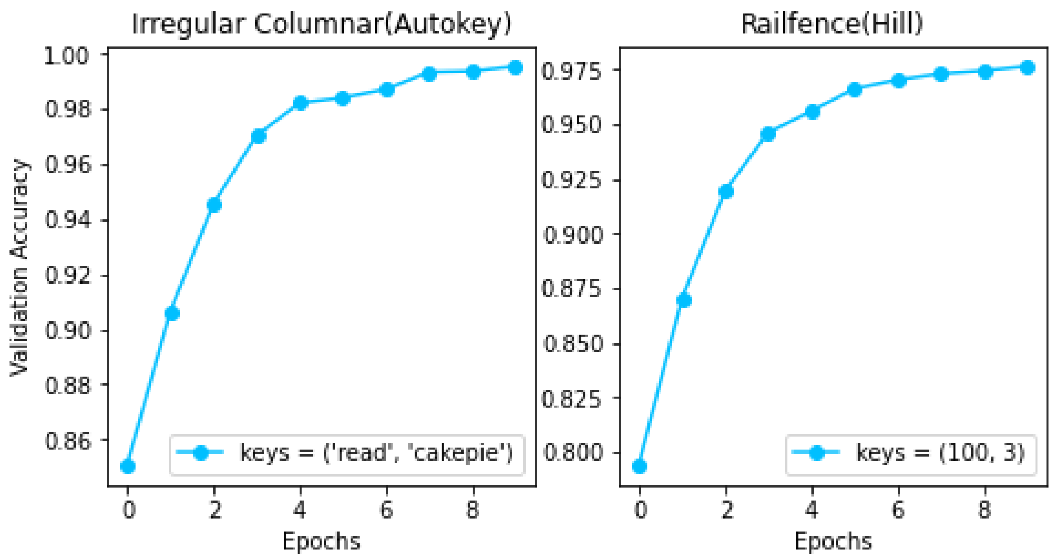

Figure 1: Proof of concept for compositions of ciphers

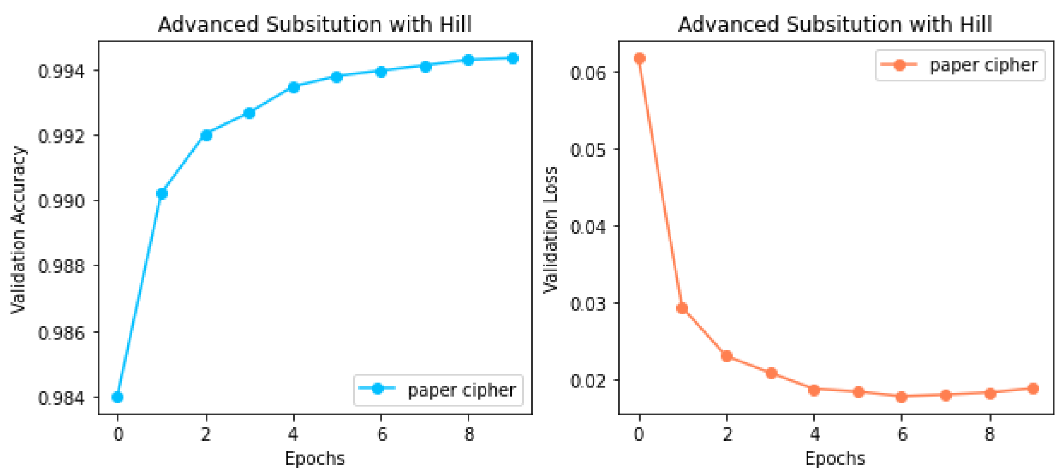

Figure 2: Main Result

and could almost score 100% validation accuracy.

Details on extraneous tests and performance on test sets can be found in the ipython notebook.

5 Contributions

- I read multiple papers to facilitate this idea and project
- I developed the architecture and did hyperparameter tuning
- I wrote all the code
- I designed the experiments

6 Acknowledgements

I did not develop or design any of the following libraries I used in my project: matplotlib.pyplot, numpy, tensorflow_text, or tensorflow. The dataset I used is from a github collection linked in the references (2) and is not of my own design. I gained my knowledge of cryptology and the encryption algorithms I used in my tests from practicalcryptography.com (1) linked in the references. I did not develop the concept behind any of the encryption algorithms I used. I simply learned about them and implemented them myself.

7 References

- Practicalcryptography.com. 2022. Practical Cryptography. [online] Available at: <http://practicalcryptography.com/cryptanalysis/text-characterisation/identifying-unknown-ciphers/>.
- Focardi, R., and Luccio, F., "Neural Cryptanalysis of Classifcal Ciphers." ICTCS, 2018
- GitHub. 2022. GitHub - dwyl/english-words: A text file containing 479k English words for all your dictionary/word-based projects e.g: auto-completion / autosuggestion. [online] Available at: <https://github.com/dwyl/english-words>.
- Rajput, Y., Naik, D. and Mane, C., 2014. An Improved Cryptographic Technique to Encrypt Text using Double Encryption. International Journal of Computer Applications, 86(6), pp.24-28.
- Khaled M. Alallayah, M. Amin, W. A. El-Wahed, and Alaa H. Alhamami. 2010. Attack and construction of simulator for some cipher systems using neuro-Identifier. Int. Arab J. Inf. Technol. (2010). Retrieved April 26, 2022 from https://www.semanticscholar.org/paper/c079b0a5589d56597838b6dfd5dd7033634b627c
- Nada Aldarrab and Jonathan May. 2020. Can sequence-to-sequence models crack substitution ciphers? arXiv [cs.CL] (2020). DOI:https://doi.org/10.48550/ARXIV.2012.15229
- Sam Greydanus. 2017. Learning the Enigma with recurrent neural networks. arXiv [cs.NE] (2017). DOI:https://doi.org/10.48550/ARXIV.1708.07576