Detection of Classical Cipher Types with Feature-Learning Approaches

Ernst Leierzopf*[1], Vasily Mikhalev[2], Nils Kopal[2], Bernhard Esslinger[2], Harald Lampesberger[1], and Eckehard Hermann[1]

[1] University of Applied Sciences Upper Austria, Hagenberg, Austria
*ernst.leierzopf@ins.jku.at
{Harald.Lampesberger,Eckehard.Hermann}@fh-hagenberg.at
[2] University of Siegen, Germany
{Vasily.Mikhalev,Nils.Kopal,Bernhard.Esslinger}@uni-siegen.de

Abstract. To break a ciphertext, as a first step, it is essential to identify the cipher used to produce the ciphertext. Cryptanalysis has acquired deep knowledge on cryptographic weaknesses of classical ciphers, and modern ciphers have been designed to circumvent these weaknesses. The American Cryptogram Association (ACA) standardized so-called classical ciphers, which had historical relevance up to World War II. Identifying these cipher types using machine learning has shown promising results, but the state of the art relies on engineered features based on cryptanalysis. To overcome this dependency on domain knowledge, we explore in this paper the applicability of the two feature-learning algorithms long short-term memory (LSTM) and Transformer, for 55 classical cipher types from ACA. To lower the necessary data and the training time, various transfer-learning scenarios are investigated. Over a dataset of 10 million ciphertexts with a text length of 100 characters, Transformer correctly identified 72.33% of the ciphers, which is a slightly worse result than the best feature-engineering approach. Furthermore, with an ensemble model of feature-engineering and feature-learning neural network types, 82.78% accuracy over the same dataset has been achieved, which is the best known result for this significant problem in the field of cryptanalysis.

Keywords: cipher-type detection · classical ciphers · neural networks · transfer learning · ensemble learning.

1 Introduction

In machine learning, classification is a problem of assigning data objects to the predefined categories by recognizing category-specific regularities in the data. There are various approaches of how neural networks can be applied to solve this problem. One way is to first manually preprocess the raw data and to calculate features which are then used by neural network for classification. In this paper, the term feature engineering (FE) is used when referring to this approach. It can be very efficient in many cases as it allows to use the domain knowledge when constructing the features. Domain knowledge in the field of cipher type detection is based on statistical measures. For example it is known that transposition ciphers have similar frequency distributions as the English language, but with different letters. Substitution ciphers are also distinguishable, because of the different algorithms used for encryption. However, the feature-engineering approach struggles from two main drawbacks: i) It requires a long and costly process of defining and testing different features. ii) It only works for problems where domain knowledge is available.

Another approach is to use neural networks which have a built-in mechanism for detecting regularities in data. This approach is often referred to as feature learning (FL). Examples of feature-learning architectures are recurrent neural networks (RNN) like long short-term memory (LSTM) [6], and Transformer-based neural networks [19].

This paper compares FE and FL approaches applied to the problem of classification of (a large synthetic set of) classical ciphers, which were historically relevant up to World War II. Current research focuses on the automated digitization, analysis, and decryption of encrypted historical documents [15]. Ciphers (or encryption algorithms) are a method to protect information (plaintext) by converting it to ciphertext. Ideally, only someone who knows the secret (key) is able to decipher a ciphertext. Therefore, strong encryption algorithms are used to achieve a high level of security and information privacy. However, if a cipher has weaknesses, a cryptanalyst (or attacker) could break it and recover the original information from the ciphertext without knowledge of the key. If the attacker has access only to the ciphertext (ciphertext-only attack), the first step is to determine which cipher was used to produce it. Then further techniques are applied for recovering the information. For this reason, cipher-type classification is a crucial part of cryptanalysis. Note, that modern ciphers are designed in such a way that it is extremely difficult to distinguish their ciphertexts from randomly generated data [7]. Therefore, classification of modern ciphers is usually impossible with the FE approach. On the other hand, a FL approach can potentially detect regularities in ciphertexts which are not detected by common methods yet. As classical ciphers usually have many weaknesses, FE approaches can be very effective for their classification as we demonstrated in [13].

This work investigates how well FL approaches perform when applied to this cipher-type-detection problem. For this research, a large amount of ciphertexts was generated using 56 classes – 55 different classical ciphers standardized by the American Cryptogram Association (ACA)[2] and one plaintext class. Without any feature pre-computation, these ciphertexts were provided as raw data to the LSTM and Transformer neural networks to detect the cipher type used to produce them. Also ensemble models with FE-only approaches, FL-only approaches and combined FE and FL approaches, were evaluated.

We also evaluate whether transfer learning is applicable to this kind of classification problem. Our results show: If a feed-forward neural network (FFNN) is trained to classify a certain number of ciphers and if at a later step previously not included ciphers are provided to it, then the learning process becomes much faster compared to training the complete model again. In other words, transfer learning can be successfully applied to this problem and catastrophic forgetting does not happen.

To summarize, our key contributions are:

– We demonstrate the feasibility of the FL approaches to the problem of classical cipher-type detection.
– By using the combination of various techniques in an ensemble model, we achieve the accuracy of 82.78% which is the best state-of-the art-result for this problem.
– We show that transfer learning is applicable here, which means that the knowledge gained through our experiments can also be helpful for the different related problems.

2 Related Work

We described in [13] the state-of-the-art feature map for all ACA ciphers and the according hyperparameters for the FE approaches FFNN, random forests (RF) and naïve Bayes networks (NBN). As a result it is shown that RF needs the smallest amount of training data in order to achieve acceptable results (about 1.5% data of the FFNN). Different optimizers and activation functions were tested with the baseline FFNN. The best and most comparable results were achieved with the Adam optimizer and the ReLU activation function. Out of the total 27 different features, which were tested one by one, 20 were used for the final feature map [13].

All relevant related work in the field of classical cipher-type detection is listed in Table 1. All results in Table 1 are based on FE to classify cipher types. They are, however, not directly comparable, because every listed work uses their own set of cipher types, features and datasets. For example, in [1] a very high accuracy of 99.60% is shown. However, this result is achieved by using ciphertexts which are 1 million characters long. Such a length is unrealistic for any practical scenario of classical ciphers usage. In other works [9, 10, 18], the classification is done among a very small number of cipher types – from 3 to 6, as compared to 56 types considered in this research. The most notable related works are from Nuhn and Knight [17] and from the authors [13] who solve the classification problem for the comparable number of ciphers – 50 and 56 respectively. Nuhn and Knight achieved an accuracy of 58.50% for 50 ACA ciphers using a neural network with a linear classifier and a Stochastic Gradient Descendent (SGD) optimizer with default parameters. To accomplish this, they used 58 features and 1 million ciphertexts with random lengths in 20 epochs for training [17]. Besides neural networks, other technologies like Support Vector Machines (SVM), Hidden Markov Models (HMM), Decision Trees (DT) and multi-layer classifiers were used in the listed work. Multi-class classification in SVMs can only be achieved by using many one-vs-one binary classifiers. This is the reason for the long training times of the SVM and the reason why it is not implemented in our current solution.

Results from the work of Sivagurunathan et al. [18], where the three classical ciphers Playfair, Hill and Vigenère were analyzed with a simple neural network, coincide with the results of Kopal [9]. Both discovered the difficulty of classifying (distinguishing) the Hill and Vigenère ciphers, because of their similar statistical values.

A multi-layer classifier has been introduced by Abd and Al-Janabi [1] to classify plaintexts and ten different cipher types. The impressive results of over 99% accuracy are lessened by the enormous ciphertext length of about one million characters, which is the equivalent of an average book with 500 pages. Ciphertexts with these lengths are seldom. Based on the DECODE database, described in [14, 15], the majority of encrypted historic manuscripts is only between a few lines and some pages of ciphertext long. As of the end of May 2021, the DECODE database contains more than 2,600 records of encrypted historic manuscripts and keys.

N. Krishna [10] developed approaches that have not yet been used by other authors for the four ciphers Simple Substitution, Vigenère, Transposition and Playfair. An important point for comparison is that the Hill cipher was not used here. The first approach, a support vector machine (SVM), uses the ciphertexts of length 10 to 10,000 characters, which are mapped in a number range, as training data. The SVM uses the implementation of the Sklearn Library[3] with a 10-fold-stratified-cross-validation and a one-vs-rest classifier for the calculation of the confusion matrix. This means that 9 out of 10 datasets were used for training and 1 dataset for testing in order to find the most suitable class. In the second and third approach, a Hidden Markov Model (HMM) was trained for 1,000 ciphertexts per class. In the second approach, this was used as input for a convolutional neural network, and in the third approach it was used as input for an SVM. The first approach achieves an accuracy of 100% with a text length of 200 characters, the second 71% with a text length of 155 characters and the third 100% with a text length of 155 characters.

Author	Accuracy in %	Ciphertext Length	Dataset Size	#Cipher Types	Cipher Category	Technology
Abd [1]	99.60	1M	N/A	11	classical	3-level-classifier
Kopal [9]	90	100	4,500	5	classical	FFNN
Krishna [10]	100	155	4,000	4	classical	SVM, HMM
Leierzopf [13]	80.24	100	200M	56	classical	FFNN, RF, NBN
Nuhn [17]	58.50	random	1M	50	classical	Vowpal Wabbit
Sivagurunathan [18]	84.75	1,000	900	3	classical	FFNN

Table 1. Summarized results and attributes of related work in the field of cipher-type detection (M = million)

[3] Sklearn Library: https://scikit-learn.org/stable/

3 Neural Cipher Identifier

During this research, the software suite “Neural Cipher Identifier (NCID)” which trains and evaluates different FE-neural-network based classifiers like FFNN, DT, RF and NBN, was extended by two FL approaches. Further details are described in [13]. The NCID software [12] is available as open-source and can be productively run for free to determine the cipher type of a given ciphertext.[4]

This section contains a short summary and the advances with respect to the previous work [13]. Basically, with the NCID software suite different types of neural networks can be trained to classify ciphertexts produced by various ciphers. The cipher types and the corresponding key lengths can be configured as well as the type of neural network to be used. The trained model gets multiple ciphertexts and predicts their cipher type. The data is generated with an extended data loader and 14 GB of English texts from the Gutenberg project[5], which can be used for research purposes free of charge. The data loader loads plaintexts with the specified length range, converts all letters to lowercase and filters all characters not included in the alphabet consisting of 26 lowercase characters. The produced ciphertext is encoded as an array of indices of the alphabet and used to calculate all features in FE approaches and directly in FL approaches. In general, neural networks need a fixed input length. The length of the ciphertext is not relevant in the FE approach, because the number of features and their lengths are always the same. In case of FL neural networks, an input is padded by repeating the ciphertext if it is not long enough. The FL neural networks should be trained with the maximal length of the expected ciphertexts, as the text easily can be extended by itself, however information is lost when the text needs to be shortened. Multiple neural network types based on Keras[6] components (FFNN) and Sklearn[7] components (DT, RF, NBN) are implemented. Also a 1-dimensional convolutional neural network has been implemented and tested, however, no notable results have been achieved with it.

3.1 Long Short-Term Memory

Hochreiter and Schmidhuber [6] describe that RNN struggle to detect long-range dependencies in sequential data, because of their short-term memory structure. Therefore, important information can be lost during processing long sequences. To overcome this problem, the long short-term memory (LSTM) was introduced [6], which uses cell states and various gates to keep or forget information. New data is filtered with the forget gate. The old hidden state together with the current input form the new hidden state by calculating several mathematical operations. The last hidden state is also the prediction of the LSTM. The forget gate uses the sigmoid activation function to decide which information should be

[4] https://www.cryptool.org/ncid
[5] https://www.gutenberg.org/
[6] Keras: https://keras.io/
[7] Sklearn: https://scikit-learn.org/stable/

kept and which should be forgotten. Intuitively, with the sigmoid functions values near 0 are not important, but values near 1 are kept. The input gate updates the cell state by passing the hidden state and current input into a sigmoid function. The sigmoid output then is multiplied with the output of the tanh function output of the hidden state and current input. The cell state is updated by pointwise multiplication by the forget factor and a pointwise addition with outputs of the input gate. The output gate decides what the new hidden state should be. [6]

The implementation of LSTM is fully based on Keras components and uses an embedding layer with the number of classes, which is 56, as input and an output of 64 data points to create a dense representation of the ciphertext. The LSTM layer uses 500 hidden units. The output is flattened and densed into the number of classes. The softmax function finally decides on the prediction by converting inputs into a probability distribution (all positive values from 0 to 1 which add up to 1) and sorting the indices of the classes by the resulting probability.

Hyperparameter	Grid	Standard	Best Result	Accuracy in %
units	[50, 100, 150, 200, 500]	-	500	72.84 (68.50)
dropout	[0, 0.2]	0	0	68.50

Table 2. Iterative grid search for LSTM

Table 2 shows the results of the iterative hyperparameter optimization for the LSTM neural network. The grid search showed that a higher number of units leads to better results. That's about 2% per 50 units. Due to the high computing time, however, the dropout and features were tested and compared with only 200 units, which results are shown in the brackets. It could be shown that the architecture leads to better results with a higher number of units, but the computing time increases in direct proportion to it.

3.2 Transformer

RNN process a text sequentially and therefore require relatively high computational costs to capture relationships between pieces of information located far away from each other in the sequence. The Transformer model [19] tackles this problem by using a so-called self-attention mechanism which allows to measure the influence of each word to all the other ones in the same sequence. A nice property of self-attention is that each of the scores can be computed in parallel and hence GPUs can be efficiently utilized for the training process.

There are various ways of how Transformer neural networks can be used to solve the classification problem. The actual architecture of the Transformer-based neural network is based on a Keras example [16]. This actual architecture used for cipher-type classification works like described in the following:

As the first step, all of the words are converted into 128-dimensional vectors using an embedding algorithm. Here, the vocabulary size is set to $20,000$ words. The resulted vectors are passed to the self-attention layer that is used to find relations between one word vector (query) and all the other word vectors (key tensors or just keys) in the same ciphertext. Multi-headed attention with 8 heads is used, which means that 8 sets of query/keys/values weight matrices are utilized. For each of these sets, the dot product is calculated between the given query and each of the key tensors, which results in the attention scores. These scores are normalized by the softmax function to obtain attention probabilities (all positive values from 0 to 1 which add up to 1). The value vectors are multiplied by these probabilities and summed up together to produce the output of the self-attention layer for the given query. These outputs are then fed to a feed-forward neural network, with the hidden layer of size $1,024$ and a ReLu activation function. This FFNN produces the output of the Transformer layer. Therefore, for each time step one vector is received. The mean across all time steps is calculated which is provided to another feed forward network on top (with the softmax activation function) for ciphertext classification.

Hyperparameter	Grid	Standard	Best Result	Accuracy in %
pooling	[Average, Max]	-	Average	58.73
ff hidden layer size	[1, 5]	1	1	58.73
vocab size	[20,000, 50,000]	20,000	20,000	58.73
ff dim	[32, 64, 128, 256, 512, 1024]	32	1.024	61.96
embed dim	[32, 64, 128, 256]	32	128	67.96
num heads	[2, 4, 8, 16]	2	8	73.71

Table 3. Iterative grid search for Transformer neural networks

Table 3 shows the results of the iterative hyperparameter optimization for the Transformer neural network. Due to computational limitations the ff dim and embed dim parameters could not be further raised.

3.3 Learning-Rate Schedulers

The learning rate of a neural network optimizer can be described as the most important hyperparameter. Setting the learning rate too high means that a model is delivering poor overall performance. A learning rate that is too low means that a model has to train for a very long time before it converges. The Adam optimizer has an internal mechanism for adapting the learning rate, but it works stochastically and calculates a learning rate between 0 and the maximal value which can be adjusted [11]. Different algorithms for adapting the learning rate are discussed in this chapter. These make it possible to use a higher initial learning rate and thus significantly reduce the time required for a model to converge. There are learning-rate schedulers that start with a very small learning rate and search for the optimal learning rate in small steps in order to reduce it later on, but these are not considered due to the increase in complexity.

The time-based decay learning rate scheduler (LRS) uses a small fixed factor (decay) to adapt the learning rate after every batch (t) by multiplying it with the number of ciphertexts trained (iter). The following equation shows a calculation step after every batch [11].

$$lr_{t+1} = lr_t \cdot \frac{1}{1 + decay \cdot iter_{t+1}} \tag{1}$$

The step-/drop-based decay LRS reduces the learning rate after a fixed number of epochs (drop), for example after every 10 epochs, by using a small decay. The following equation shows a calculation step after every epoch (t) [11].

$$lr_t = lr_0 \cdot decay^{\lfloor \frac{t}{drop} \rfloor} \tag{2}$$

Just like with the time-based decay LRS, the exponential decay lowers the learning rate after every epoch by multiplying the initial learning rate with the exponential of a small factor (k) multiplied with the number of epochs trained (t). The main difference is that the exponential LRS reduces the learning rate rapidly in the beginning and slowly in the end. In theory that should allow the network to generalize better. The following equation shows a calculation step after every epoch [11].

$$lr_t = lr_0 \cdot e^{-kt} \tag{3}$$

The custom step-based decay LRS uses an early stopping mechanism, which stops training as soon as no progress can be determined after 250 mini-batches. The idea is to adjust the learning rate ahead of time in order to achieve further progress. To do this, the learning rate is gradually reduced by a percentage as soon as no progress is detected after 100 mini-batches.

In NCID, the time-based decay LRS (equation 1) and a custom step-based decay LRS are implemented. The step-based decay LRS (equaton 2) and the exponential LRS (equation 3) are not applicable in NCID, because they are based on the number of epochs trained, but training examples are generated on-the-fly and used only once, which means the epoch stays the same. The custom step-based decay LRS is used by default. It is the safest option to use, because it is only applied when no progress is made for a long time.

3.4 Transfer Learning

Transfer learning is the process of using obtained knowledge from solving one problem and transferring this knowledge to a different (but related) one. For the case of classical cipher type detection, this approach can be advantageous due to the high diversity of the historical encrypted manuscripts and ciphers which vary based on the age, language, length, presence of typos, etc. In this work, such factors are out of the scope. However, we would like to understand if the knowledge gained through our experiments can also be helpful for classifying other, but similar, ciphers which slightly differ from the ones that we used for training of the models. Transfer learning sometimes can be threatened by the so-called catastrophic forgetting, where the additional training (retraining) with a new set of classes causes known features (learned weights) to be forgotten [8]. As a first step, to determine whether catastrophic forgetting takes place in the cipher-classification problem, we evaluated whether the knowledge obtained by training the models to classify the ciphers from the smaller set, can be reused when applied to the extended sets of encryption algorithms. In order to achieve this, multiple FFNN base models were trained from scratch with 30 (B30), 35 (B35), 40 (B40) and 5 (B5) ciphers. The ciphers were selected and added alphabetically, whereas B5 only used those that were not used in B30. This results in the following scenarios:

- Extension of B30 by 5 new ciphers (T35)
- Extension of B30 by 10 new ciphers (T40)
- Extension of B30 by 5 new ciphers, however, the ciphers are added incrementally in 5 steps, whereas the newest model becomes the new basis (T35incremental)
- Training of a new model with 5 new ciphers, which are not included in B30, which is used as a basis (T5)

Scenario	Accuracy in %	Converges after iterations in million
B30	94.44	261
B35	92.26	189
B40	85.24	166
B5	99.96	138
T35	92.28	76
T40	86.29	75
T35incremental	92.70	50-106
T5	99.98	91

Table 4. Results from the transfer learning tests

Table 4 shows that transfer learning leads to substantially less training data needed for convergence and at least the same or better accuracy in all scenarios. Especially when expanding models with a large number of classes, the amount of data required drops to less than half. When comparing B35 to the scenarios T35 and T35incremental, a better accuracy and lower amount of data needed for convergence was achieved. Similarly B40 is compared to T40 and B5 is compared to T5.

3.5 Ensemble Learning

An ensemble is a model that combines the predictions of two or more models. These models can be trained in advance with the same or different datasets. The prediction can be calculated using simple procedures such as a mean voting or more complex procedures such as weighting the individual votes [4].

In NCID, mean voting and weighted voting are used as voting methods in the ensemble models. The final two ensemble models combine three FE architectures (FFNN, NB and RF) and two FL architectures (LSTM and Transformer). With weighted voting, the probabilities with the respective statistics, i.e. Precision, Recall, Accuracy, F1-Score and Matheus Correlation Coefficient, are calculated in one voting. This is intended to ensure that the architectures that are better for the respective classes are preferred. Based on statistical findings, FL architectures produce about 10-20 % better results, for example, for the Myszkowski cipher, and FE architectures produce 10-20% better results, for example, for the Beaufort cipher.

Multiple models were trained with different datasets and each one was evaluated with 10 million records of the same test dataset. Table 5 shows that the FE&FL ensemble of models results in an improvement of 4% against the FE ensemble (82.78-78.67%). Compared to FFNN (the model with the best single performance), the FE-only ensemble does not improve the accuracy significantly (78.67-78.31%). However, the ensemble of FL-only approaches improves the recognition rate with about 4% (76.10-72.33%) compared to the best single FL approach, which is the Transformer neural network.

Model	Approach	Accuracy in %
Weighted-Voting FE & FL	Ensemble	82.78
Mean-Voting FE & FL	Ensemble	82.67
Weighted-Voting FE-only	Ensemble	78.67
FFNN	FE	78.31
Weighted-Voting FL-only	Ensemble	76.10
RF	FE	73.50
Transformer	FL	72.33
LSTM	FL	72.16
NBN	FE	52.79

Table 5. Comparison of different models analyzing ciphertexts of 100 characters length

4 Conclusion

The current state-of-the-art classifiers approach the cipher-type detection problem by engineering features based on domain knowledge (FE approach). In this paper, we use an FL approach exploring the two neural networks, LSTM and Transformer, without the reliance on domain knowledge. In terms of the number and type of ciphers used, the results of this paper are mostly comparable with the classifiers by Nuhn [17] and by Leierzopf et al.[13], while the other related work use much smaller sets of ciphers or consider unrealistically long ciphertexts.

The best results for FL-only approaches were achieved with the Transformer model, which shows an accuracy of 72.33%. Applied to the same test data, this result is worse than the best FFNN FE approach [13], which achieved 78.31% accuracy. Using only a single model, FFNN outperforms the Transformer and LSTM neural networks in our particular application. However, an ensemble model, where 5 different neural network models are put together, improved the result and achieved an accuracy of 82.78%. This improvement is an indication that the FL approach detected some regularities in the data that were not explored by the FE approaches, which is an interesting result for the cryptanalysts. The improvement is also validated by the results of the ensembles of FE-only approaches with 78.67% accuracy and FL-only approaches with 76.10% accuracy. These results are summarized in Table 5.

The conclusion of this work is that the solely FE approach can be superior in fields with domain knowledge, like the classical cipher type detection. However, FL neural networks have the ability to extract unknown feature-types and can thus be a valuable additional part improving the best known FE results.

The combined FE&FL ensemble models delivered better results than any single neural network and the combination of only FE or only FL approaches. The ensemble models use the strengths of the individual models without having to spend additional computing time on training. Although the evaluation time increases significantly due to the number of models used, the combined FE&FL ensemble models lead to an improved recognition rate by another 4% which is a strong result.

The experiments with different scenarios also showed that catastrophic forgetting did not occur. This knowledge allows us to train a basic model, e.g. with all ACA ciphers, and to use it as a basis for new models. Therefore, only the last layer is exchanged with a layer of the needed output size and the pre-computed weights are further trained. As a result, we achieved the same or better accuracy with 40-60% less data and thus a much shorter training time.

Future work in this field should include the training of models with texts from different languages or texts including errors. Another interesting further research direction is to apply the FL approach to classification of modern ciphers which are designed to be resistant against the FE approach. First successes in this direction are achieved for instance by Gohr [5] against round-reduced versions of the Speck32/64 block cipher [3]. One more approach for future research is to combine different one-vs-one classifiers and add these to the decision making process.

Acknowledgements

This work has been supported by the Swedish Research Council (grant 2018-06074, DECRYPT – Decryption of historical manuscripts) and the University of Sciences Upper Austria for providing access to the Nvidia DGX-1 deep learning machine.

References

1. Abd, A., Al-Janabi, S.: Classification and Identification of Classical Cipher Type using Artificial Neural Networks. In: Journal of Engineering and Applied Sciences. vol. 14 (2019)
2. American Cryptogram Association: Cryptogram (2005), https://www.cryptogram.org/ [visited 2021-04-14]
3. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The SIMON and SPECK lightweight block ciphers. In: Proceedings of the 52nd Annual Design Automation Conference. pp. 1–6. Association for Computing Machinery, San Francisco California (June 2015)
4. Brownlee, J.: Why use ensemble learning? (October 2020), https://machinelearningmastery.com/why-use-ensemble-learning/ [visited 2021-04-14]
5. Gohr, A.: Improving attacks on round-reduced speck32/64 using deep learning. In: Annual International Cryptology Conference. pp. 150–179. Springer (2019)
6. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Computation **9**, 1735–1780 (1997)
7. Katz, J., Lindell, Y.: Introduction to Modern Cryptography. CRC press (2020)
8. Kemker, R., McClure, M., Abitino, A., Hayes, T., Kanan, C.: Measuring catastrophic forgetting in neural networks. 54 Lomb Memorial Drive, Rochester NY 14623 (November 2017), arXiv:1708.02072
9. Kopal, N.: Of ciphers and neurons–detecting the type of ciphers using artificial neural networks. In: Proceedings of the 3rd International Conference on Historical Cryptology HistoCrypt 2020. pp. 77–86. No. 171, Linköping University Electronic Press (2020)
10. Krishna, N.: Classifying Classic Ciphers using Machine Learning. Master's thesis, San Jose State University, California, USA (May 2019)
11. Lau, S.: Learning rate schedules and adaptive learning rate methods for deep learning (December 2020), https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1 [visited 2021-04-14]
12. Leierzopf, E.: NCID - Neural Cipher Identifier (2021), https://github.com/dITySoftware/ncid
13. Leierzopf, E., Kopal, N., Esslinger, B., Lampesberger, H., Hermann, E.: A massive machine-learning approach for classical cipher type detection using feature engineering. In: HistoCrypt (accepted) (2021)
14. Megyesi, B., Blomqvist, N., Pettersson, E.: The DECODE Database: Collection of Historical Ciphers and Keys. In: In Proceedings of the 2nd International Conference on Historical Cryptology, HistoCrypt 2019. pp. 69–78. Linköping Electronic Press, Mons, Belgium (June 2019)
15. Megyesi, B., Esslinger, B., Fornés, A., Kopal, N., Láng, B., Lasry, G., Leeuw, K.d., Pettersson, E., Wacker, A., Waldispühl, M.: Decryption of historical manuscripts: the DECRYPT project. Cryptologia pp. 1–15 (2020), dOI:10.1080/01611194.2020.1716410
16. Nandan, A.: Text classification with transformer (May 2020), https://keras.io/examples/nlp/text_classification_with_transformer/
17. Nuhn, M., Knight, K.: Cipher Type Detection. In: Conference on Empirical Methods In Natural Language Processing. pp. 1769–1773. Doha, Quatar (October 2014)

18. Sivagurunathan, G., Rajendran, V., Purusothaman, T.: Classification of Substitution Ciphers using Neural Networks. In: IJCSNS International Journal of Computer Science and Network Security. vol. 10, pp. 274–279 (March 2010)
19. Vaswani, A., Shazeer, N., Parmer, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. In: 31st Conference on Neural Information Processing Systems. Long Beach, CA, USA (2017)