Deciphering Simple Ciphers With Reinforcement Learning

Peiyu Liao
Email: u3512485@connect.hku.hk

1. Introduction

AlphaGo Zero has amazed the world not only by its unbeatable performance in playing Go, but also by the fact that it learns Go entirely without human knowledge using reinforcement learning [1]. Reinforcement learning seems to be a promising model to approach complicated tasks by providing a formalization of the interaction environment, but how hard is it to achieve a performance as outstanding as AlphaGo Zero.

Zaremba et al. [2] has worked on learning simple algorithms with reinforcement learning, including copying text, reversing text, adding multi-digit numbers, etc. Despite the tasks being seemingly more straightforward than playing Go, the experiments were not successful until several advanced generalization techniques were incorporated. Moreover, the output prediction of their model were trained with supervision over the input-target pairs and not entirely with reinforcement learning. This has motivated us to apply a pure reinforcement learning technique in solving similar simple algorithms, as well as to demonstrate the complications in formalizing tasks with a time-variant environmental setting.

Deciphering simple ciphers is chosen as the task in this project. The agent reads through the cipher text character by character, receives rewards upon action, and eventually outputs the correct deciphered text, also called plain text. Substitution cipher is among the simplest types of ciphers, and due to time and resource limitation, Caesar cipher will be the focus in our experiments. Other variations such as affine cipher and polyalphabetic cipher can be easily substituted into the learning environment.

The objectives of this project include:

1) Demonstration of the feasibility and difficulty of reinforcement learning in learning simple algorithms in a time-variant environmental setting, without any supervision.
2) Observations of the effects of different hyper-parameters in the decipherment process.
3) Construction of a decipherment environment that can be used for further research into the problem with reinforcement learning.

The full code of the project is on github: https://github.com/pyliaorachel/reinforcement-learning-decipher.

2. Related Work

The work from Zaremba et al. [2] has the closest nature to this project. In their work, six tasks were approached, including copy, reverse, walk, addition, 3 number addition, and single digit multiplication. The environment for observation was either a 1-D input tape or a 2-D input grid, with the agent reading one character at a time, hence it was a time-variant system as the next action depended on the history of the read inputs instead of the current one only. A Recurrent Neural Network (RNN)-based controller encoded the input history into a hidden state for action decision. Upon observation, the direction to move over the input, whether to output or not, and which symbol to output were decided. They came up with two settings to approach the problem, one of which was based on Q-learning. However, only

the moving direction and whether to output were learned by Q-learning, while the output prediction was trained with input-output pairs in a supervised manner. On the contrary, the output prediction in our task was formulated as an action to be trained merely by rewards in Q-learning, which removes the need to provide the exact target, making the learning more general yet more challenging.

For the task of deciphering classical ciphers, seldom works were done using machine learning approaches, as methods with human knowledge involved may be the most efficient. Brown [3] has approached the decipherment of substitution ciphers and transpositional ciphers with several machine learning algorithms, including hill climbing algorithm, genetic algorithm, and simulated annealing. A more effective model by Wu [4] was based on Hidden Markov Model (HMM), in which the objective was to find out the conditional probability distribution of the plain text given the cipher text, i.e. $P(\text{plain text}|\text{cipher text})$. Our project is, to the best of our knowledge, the first attempt to explore the use of reinforcement learning in deciphering simple ciphers.

3. Task and Environment

3.1. Substitution Cipher

In substitution cipher, each character in the alphabet was substituted with another character, i.e. the cipher text uses a different alphabet from the plain text. Caesar cipher is a type of substitution cipher, in which the alphabet of the cipher text is a linear "shift" from the original alphabet. For example, if the plain text is "machine", and the shift amount is +5, then the cipher text becomes "rfhmnsj".

Another kind of substitution cipher is called affine cipher, which maps the letters with a simple function:

$$c = (ap + b) \bmod m \tag{1}$$

where c and p are the cipher and plain letters respectively, m is the size of alphabet, and a and b are the parameters of the mapping function. In fact, Caesar cipher is a particular case of affine cipher where a is 1.

In this project, Caesar cipher will be the focus, which is the simplest form of substitution cipher.

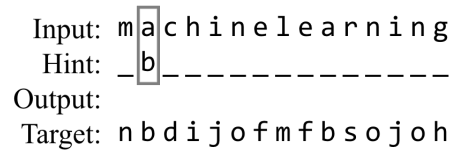


Figure 1: The environment.

3.2. Environment

Similar to [2], the environment contains an input tape with the cipher text and an output tape with the deciphered plain text. This is not enough for the decipherment of Caesar ciphers, as the shift amount can be arbitrary. In a more general setting, a dictionary can be provided to mimic human's language knowledge, and the decipherment would try to maximize the number of matches between the plain text and the dictionary. Alternatively, hints can be provided to indicate the shift amount, such as a pair of correct cipher and plain letters. The experiments will only focus on the hint mode, but the interface can be easily set up in our framework for the general mode.

There is a cursor on the input tape, moving right or left at each step depending on the agent's action. Under the hint mode, the cursor is placed above the input-hint pair, reading the pair at once, as illustrated in Figure 1. Ideally, the model should learn the shift amount present in the useful hint pair as well as learn to ignore the underscores in the hints.

At each timestep, three actions need to be determined: 1) Next cursor movement 2) Output or not 3) Output character. A positive reward is given upon each correct prediction, while negative rewards are given upon wrong prediction, timeout, or not outputing.

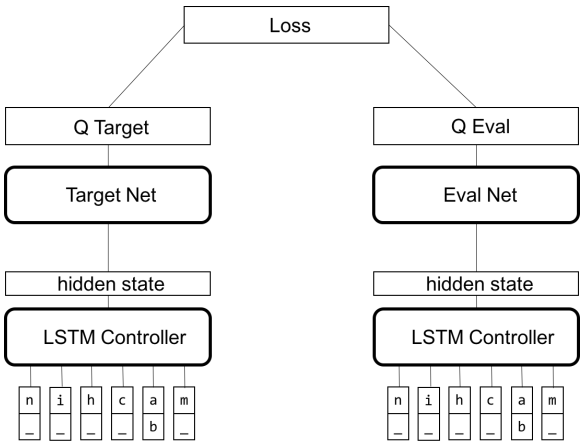


Figure 2: The model consists of an evaluation network and a target network. The networks take input states from the underlying LSTM controllers.

4. Model

The two essentials involved in the task are encoding the history of inputs in a state, and learning the best actions to perform upon observing the state. The former issue is addressed with a LSTM controller, while the latter one is approached with deep Q-learning algorithm. The full architecture is shown in Figure 2.

4.1. LSTM Controller

Upon observing the useful input-hint pair, it is necessary for the shift amount to be remembered in subsequent output predictions. Hence, a function of memory needs to be present in our model.

RNN is a popular neural network model that preserves information in the network by cascading the previous output to the current hidden state, forming an internal information loop. This has granted RNN the ability to retain memory over long sequences [5], making it adaptable to applications concerning sequential data.

Long Short-Term Memory (LSTM) neural network [6] is a variation of RNN that generally has good experimental performance on time-variant tasks. It was proposed to resolve the vanishing gradient problem with simple RNNs [7], which is an issue that prevents the preservation of memories from long-term dependencies in the network.

In our model, a LSTM controller encodes the sequence of observations from the input-hint pairs, and the hidden state is the representation that can be used in choosing the actions.

4.2. Deep Q-Learning

The objective of reinforcement learning is to learn an optimal policy, i.e. knowing the optimal action to choose upon an observed state. To achieve this with Q-learning algorithm, the Q function is approached by value iteration update through the experienced interactions with the environment:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a)+\alpha(r+\gamma \max_{a'} Q(s',a')) \tag{2}$$

where s and a are the current state and action, s' and a' are the next ones, r is the current reward, α is the learning rate, and γ is the discount factor.

The Q function approximation can be done with deep learning, hence deep Q-learning. With deep Q network, hand-crafted features can be avoided and instead be extracted from raw observable data, the effectiveness of which was demonstrated in the Atari game engine built by Mnih et al [8]. In our project, the deep Q network can be directly connected to the LSTM controller, which serves as the feature extractor.

The training of deep Q network was found to be hard to stabilize. In [8], two techniques were proposed to address this problem: 1) Experience replay. Retain a pool of past experiences and randomly select some of them for learning. This removes dependencies arising from the observation sequence. 2) Maintain two networks, an evaluation and a target network. Iteratively update the evaluation network while delaying the update of the target network for a period of time. This reduces the correlation between the Q-values that is being learned with the target ones. Both techniques are present in our implementation.

5. Experiments

The effectiveness of the models are evaluated in terms of total rewards received and decipherment accuracy, i.e. the ratio of successful decipherments over the experiment. Performance is compared between different settings, including different character representations and different rewarding schemes in the environment.

5.1. Character Representation

When the input and hint characters in the environment are read as observations, they need to be transformed into some format other than the original ascii or integer representations for the neural networks to process. Three different kinds of representations are under experiment:

1) One-hot vectors. The character symbols are treated as independent classes and represented as vectors.
2) Ordinal vectors. The character symbols are treated as classes with ordering information and represented as vectors. The methodology is described in [9].
3) Ordinal numbers. The character symbols are treated as classes with ordering information and represented as float numbers in the range of $[0, 1]$.

In this experiment, the rewarding scheme is static.

5.2. Rewarding Scheme

Aside from the architecture and hyperparameters of the model, sometimes the rewarding schemes provided by the environment can affect the training effectiveness significantly. In our environment, rewards are given under four scenarios: 1) Correct output, with positive rewards 2) Wrong output, with negative rewards 3) No output, with negative rewards 4) Timeout, with negative rewards. A straightforward rewarding scheme is to always provide static rewards.

However, due to the nature of the deciphering task, an error-based rewarding scheme can be utilized in the case of wrong output prediction. In this scheme, when a wrong output is predicted, the absolute reward amount is proportional to the difference between the prediction and the target so that if the prediction differs from the target too much, it is penalized more. For example, when the target is "i", predicting an "h" may receive a reward of -0.1 while predicting an "a" may receive a reward of -0.8, due to their respective ordering difference to the target character. Under the error-based scheme, the agent is expected to be trained better as the rewards indicate the significance of the alphabet ordering in the task.

In this experiment, one hot vectors are tested.

6. Result Evaluation

The performance is measured based on total rewards per episode, and the number of successful decipherments, i.e. success rate, per few episodes.

The distribution of rewards is as follows:

1) A correct prediction earns 1.
2) A wrong prediction earns -0.1 if under static rewarding scheme, $-\frac{\text{difference between prediction and target}}{\text{alphabet size}}$ if under error-based rewarding scheme.
3) No output earns -0.05.
4) Timeout earns -1.

The length of the ciphers are 2 to 4, which is small compared to previous works. In fact, in our implementation, all training starts with small cipher lengths, and the length will be ramped up if the rewards in the last few episodes are above threshold. However, the default threshold in our experiments were set too high, and the length cannot be ramped up. Nevertheless, interesting facts are still observable from the performance trends on these short ciphers.

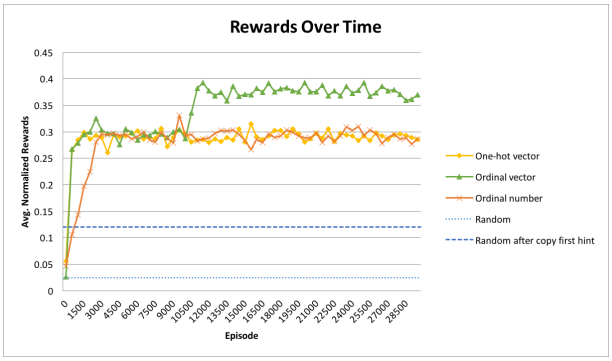
For simplification, the alphabet size is set to 5, and the hint is always the first character.

6.1. Character Representation

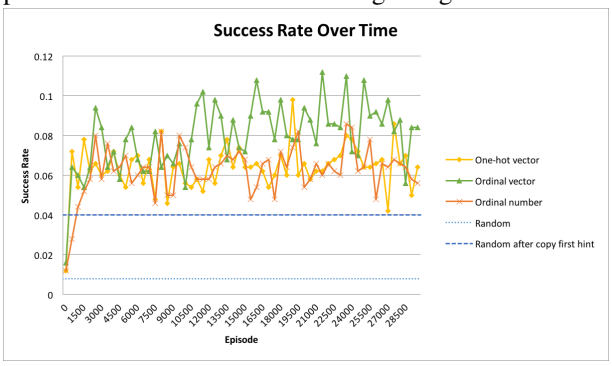
Figure 3 shows the gained reward and success rate over time for the three character representations. The rough theoretical performance of an agent acting randomly, and the performance given that the first character is always perfectly predicted, are also presented as the baselines. The presence of the latter theoretical performance is to prove that the model not only learns to copy the first hint, but also learns to decipher. The estimation is rough in that only successful decipherments with maximum reward are considered, and the average reward of all other conditions is assumed to be zero, i.e. reward $=$ maximum reward $\times$ probability of successful decipherment.

The following four observations have been made:

1) Overall performance better than random. This has demonstrated that decipherment has been learned by the model, despite to an imperfect extent.
2) Performance flattened after a few episodes. A perfect performance in the experiment will be a reward of 3, as is the average cipher length. However, the performance has been stalled at an average reward of 0.3, and there is an observable performance gap to be filled.
3) Ordinal vector has the overall best performance. By taking into account the significance of ordering in the decipherment task, the performance has been improved. However, this technique is based on human knowledge about Caesar cipher, and the performance of one-hot representation more truly reflects how good an agent can perform in the task without any prior knowledge.
4) Vectors stabilize better than scalars. From the performance at the first few episodes, it is observed that ordinal numbers, despite incorporating the ordering information, reaches local maximum more slowly than vector representations. This implies that vector representations are a more suitable format than scalars in neural network training.



(a) Rewards gained over time. The rewards are normalized by the cipher lengths. The theoretical random performance is based on an average length of 3.



(b) Success rate over time.

Figure 3: Performance of different character representations.

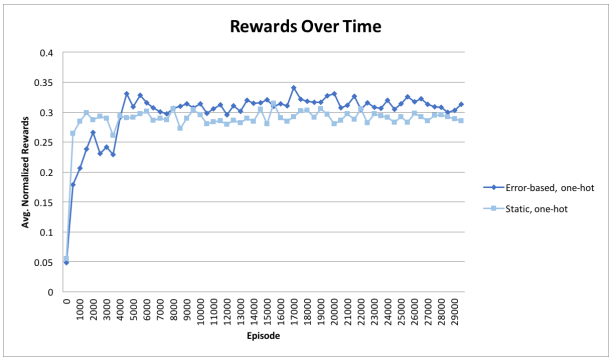
6.2. Rewarding Scheme

Figure 4 shows the gained reward and success rate over time for the two rewarding schemes under one-hot vector character representation.

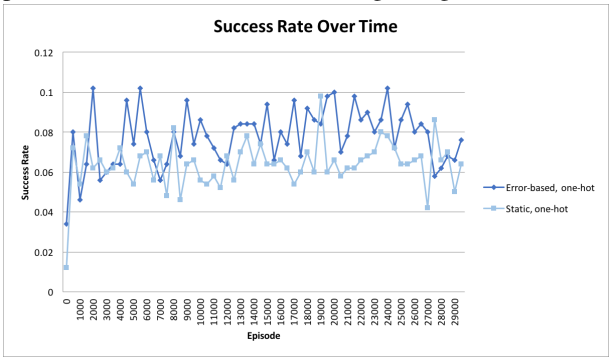
Overall, the performance has been boosted under error-based rewarding scheme. Error-based rewarding is demonstrated to be indicative of useful ordering information to the model, but the difference is not significant.

7. Discussion

Deciphering simple ciphers with reinforcement learning is no trivial task, as demonstrated by the unacceptable performance in the experimental results. In the work of [2], the character-related tasks are mostly trivial copying of charac-



(a) Rewards gained over time. The rewards are normalized by the cipher lengths. The theoretical random performance is based on an average length of 3.



(b) Success rate over time.

Figure 4: Performance of different rewarding schemes. Character representation include one-hot and ordinal vectors.

ters, which is relatively much simpler than our deciphering task that requires the mathematical relationships among the input and target to be captured. In fact, observing from the deciphering process, the models have learned that the first character of the hint should always be copied, and this has demonstrated the different levels of difficulty between deciphering and copying tasks.

Another challenge lies in the encoding of the history of observed inputs into representable states. Due to the nature of neural networks, the hidden state representation is hard to interpret, and it is not trivial to conclude whether the current LSTM controller has been effective, or needs improvement. The same uncertainty of implementational effectiveness applies to the deep Q-network.

8. Conclusions and Future Works

Despite the eye-catching success stories of reinforcement learning in building game playing agents, utilizing reinforcement learning in tasks as simple as deciphering can be non-trivial, which is probably due to the time-variant environmental setting.

While the model performance was below satisfaction, the model has still learned some amount of knowledge in deciphering. Useful observations have also been made from the experimental results. Ordinal vector outperforms other types of character representations in the deciphering task in which the ordering information is critical, and it can be suggested as an option in solving other tasks with similar nature. In addition, vector representations are more suitable for training in neural networks, compared to scalar ones. Furthermore, the rewarding mechanism of the environments can be crucial to the performance and convergence time in reinforcement learning, and it should be carefully determined if possible.

The decipherment environment has been set up in this project, and anyone can take up the challenge if interested. In the future, deciphering different ciphers such as affine cipher or polyalphabetic cipher can be investigated. Deciphering without hints is another possible direction and is closer to the settings in real world decipherment; the use of dictionary or other language knowledge needs to be incorporated.

References

[1] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[2] W. Zaremba, T. Mikolov, A. Joulin, and R. Fergus, "Learning simple algorithms from examples," in *International Conference on Machine Learning*, 2016, pp. 421–429.

[3] R. Brown *et al.*, "breaking and entering: Evaluation of various decryption techniques to decipher a polyalphabetic substitution cipher." Ph.D. dissertation, Cardiff Metropolitan University, 2017.

[4] Z. Wu, "Decipherment of evasive or encrypted offensive text," Ph.D. dissertation, Applied Sciences: School of Computing Science, 2016.

[5] M. Boden, "A guide to recurrent neural networks and back-propagation," *the Dallas project*, 2002.

[6] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[7] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *International Conference on Machine Learning*, 2013, pp. 1310–1318.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[9] J. Cheng, Z. Wang, and G. Pollastri, "A neural network approach to ordinal regression," in *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*. IEEE, 2008, pp. 1279–1284.