Character Based Neural Network Decipherment for Lost Languages

Abhishek Mishra[1]
USC
mishraab@usc.edu

Vansh Dhar[1]
USC
vdhar@usc.edu

Vaibhav Vats[1]
USC
vvats@usc.edu

Shinka Mori[1]
USC
shinkamo@usc.edu

Kushal Agarwal[1]
USC
kushalag@usc.edu

Abstract

Every known case of lost language decipherment has been accomplished by human researchers, often after decades of painstaking efforts. There are still dozens more of archaeological languages that remain undeciphered till date. For some of these archaeological languages, even the basic questions pertaining their origins and their connections to already known languages is unclear. In this work, we tried to utilize Natural Language Processing and Deep Neural Architecture in order to answer these questions and finding connections of unknown symbols in lost language to already known languages. Our proposed model is able to identify 81% of the cognates correctly between Ugaritic(lost) and Hebrew(known) language and beats the state-of-art's performance by 15.13%.The code for this project is available at: https://github.com/ShinkaM2/CSCI544_project, The video presentation for this project is available at: https://youtu.be/Re5VLVPvOno

1 Introduction

Decipherment is a very challenging task in the field of statistical NLP. It takes enormous amount of time of linguists and scholars to understand the mapping between the lost and known languages. Any Machine Learning model requires surplus data to train which makes this task tedious as there is scarcity of ancient texts. Even if a decent model is developed in the process, we cannot reuse the same model to decipher other lost languages. [1]

[1]All the authors have contributed equally. And are currently pursuing their graduate degrees' from University of Southern California.

Our work for computational decipherment task closely follows the setup of Neural decipherments (Luo, Cao, and Barzilay 2019b). The input consists of text from lost language and non-parallel data from known language. To facilitate the process of decipherment a deep learning architecture needs to take the advantage of known patterns in language translation. Some of these properties are: distributional similarity of matching characters, Monotonic character mapping within cognates, Structural sparsity of cognate mapping and Significant Cognate Overlap Within Related Languages.

2 Related Work

Non-parallel Machine Translation: Our work closely relates to non-parallel machine translation on a higher level. As per the research by (Lample et al. 2018), it is possible to make accurate cross-lingual lexical representation without any access to parallel data via character-level information. It was done by aligning monolingual word embedding space in unsupervised way. However, high-quality monolingual word embedding data is required for better results. Thus, these methods can not be applied on lost language translation as data available is not there in rich quantity.

Decoding Ciphered Texts: Most of the early work in decipherments has man-made ciphers as focal points. EM Algorithms tailored towards these man-made ciphers, prominently substitution cipher, are usually deployed in these decipherments (Knight, Nair, et al. 2006). The adjustments in EM algorithm were made according to the assumptions by ciphers on ways to produce data. Later on, research by (Kambhatla, Mansouri Bigvand, and Sarkar 2018) solves this problem by using heuristic search procedure along with a pre-trained language model. However, these methods of tackling man-made ciphers so far has

Dataset	Tokens(lost/known)	Cognates
Ugaritic	7353 / 41263	2214
Linear B	919 / 919	919
Linear B with Greek names	919 / 455	455
Linear A with Greek names	919 / 455	455
Romance	583 / 583	583
Asian	919/919	919

Table 1: Dataset observations

not been successful on archaeological data.

HMM based Method to Decipher Scripts: As per the research by (Knight and Yamada 1999), a HMM based deciphering method can be applied on unknown scripts that represents known spoken language. This method of "make the texts speak" applies character-to-sound mappings from non-parallel characters and sound sequences. However, there has not been any successful attempts of its implementation on archaeological data. Also, this work does not relates words in different languages, hence it is not possible to use it with this data.

Decoding of Ancient Texts: Our work is kindred towards computational decipherment of ancient scripts. Some initial works has already done in this field like (Snyder, Barzilay, and Knight 2010a) where they designed a Bayesian model to integrate linguistic constraints. This includes incorporation of morphological structure, customized prior alphabetic matching, etc. (Berg-Kirkpatrick and Klein 2011) proposed a different model incorporating with an inference algorithm. Their model performed well when matching vocabularies only contain cognates, however it doesn't show success in deciphering full scenarios. Most closely related research for our work is by (Luo, Cao, and Barzilay 2019b) where they use seq2seq model to capture character level correspondences between cognates.

3 Ideology

Our Aim was to build a character-based sequence-to-sequence Deep learning architecture for deciphering of various unknown languages using the following known patterns in language translation:

1. *Distributional Similarity of Matching Characters:* Matching characters that appear in similar places in corresponding cognates, generally have context that also matches one another.

2. *Monotonic Character Mapping within Cognates:* Matching cognates rarely exhibit character reordering, thus their alignment must be order preserving.

3. *Structural Sparsity of Cognate Mapping:* It is well-documented in historical linguistics that cognate matches are mostly one-to-one, since both words are derived from the same proto-origin.

4. *Significant Cognate Overlap Within Related Languages:* We expect that the derived vocabulary mapping will provide sufficient coverage for lost language cognates.

4 Proposed Architecture

4.1 BaseLine Architecture

In this section, we give a brief overview of the Baseline Architecture for our proposed model as developed by (Luo, Cao, and Barzilay 2019b).

4.1.1 Generative Framework

The decipherment principles mentioned earlier are all incorporated into a single generative framework. Specifically, we introduced a latent variable

$$G = f_{i,j}$$

that represents the word-level alignment between the words in the lost language X = $\{x_i\}$ and those in the known language Υ= $\{y_j\}$. More formally, we calculated the joint probability:

$$Pr(X, \Upsilon) = \sum_{\Gamma \epsilon F} Pr(\Gamma) Pr(X|\Gamma) Pr(\Upsilon|\Gamma, X)$$
$$\propto \sum_{\Gamma \epsilon F} Pr(\Upsilon|X, \Gamma)$$

$$= \sum_{\Gamma \epsilon F} \prod_{y_j \epsilon \Upsilon} Pr(y_j|X, \Gamma)$$

by assuming a uniform prior on both Pr(Γ) and Pr(X / Γ), and i.i.d. for every y_j εΓ. We use F to describe the set of valid values for the latent variable Γ.

The probability distribution Pr(y_j / X , Γ) could be further defined as:

$$Pr(y_j|X, \Gamma) = \sum_{x_i \epsilon X} f_{i,j}.Pr_\theta(y_j|x_j)$$

where the conditional probability $Pr_\Theta(y_j/x_i)$ is modeled by a character-based neural network parameterized by Θ, which helps us incorporate Properties 1 and 2 from last section.

4.1.2 Base Neural Decipherment Model

We are using a character based sequence-to-sequence model for decipherment and in order to incorporate Property 1, we're using shared universal character embedding space along with residual connections among weighted sum of embeddings and softmax. Property 2 is realised using monotonic alignment regularization. Further details of these parts are explained below:

Universal character Embedding: We assume that any character embedding is linear combination of universal embedding and embeddings of both lost and known language resides in the same space. Hence, we use an universal embedding matrix $U \in M^{n_u \times d}$, lost language character embedding matrix $W_x \in M^{n_x \times n_u}$ and known language character weight matrix $W_y \in M^{n_y \times n_u}$. Here, n_x, n_y are the number of unique characters in lost and known languages respectively and n_u represents size of universal character inventory. Embedding matrices are computed by:

$$E_x = W_x U$$
$$E_y = W_y U$$

Residual Connection: A residual connection is added between encoder embedding layer to decoder projection layer in order to improve the quality of character alignment. Then the following is computed

$$c = \sum_i \alpha_i E_x(i)$$

$$\hat{h} = c \oplus h'$$

where h' is a context vector, $E_x(i)$ is the encoder character embedding at position i and c is weighted character embedding. $\hat{h}$ is used to predict the next character.

Monotonic Alignment Regularization: To regularize the model a penalty is added whenever insertion or deletion occurs. For each word in the lost language, the alignment probability $Pr(a^t{}_i|x_i)$ is computed over the input sequence at decoder time t. Expected alignment position:

$$p^t{}_i = \sum_k k.Pr((a^t{}_i = k|x_i))$$

where k is any potential aligned position. The regularized term is:

$$\Omega_1(\{p_i^t\}) = \sum_t (p_i^t - p_i^{t-1} - 1)^2$$

Furthermore, quadratic loss functions are used to avoid expensive multi-character insertions or deletions.

4.1.3 Minimum-Cost Flow

The latent variable F should identify a reasonable number of cognate pairs between the lost and known languages, while meeting the requirement that word level alignment is one-to-one. In order to identify the cognate pairs, we use minimum-cost flow methodology. The edges in flow setup are:

- $f_{s,i}$: edges from source node to the word x_i in the lost language.
- $f_{j,t}$: edges from word y_j in the known language to the sink node.
- $f_{i,j}$: edges from x_i to y_j.

Each edge has a capacity of 1 and only the edges $f_{i,j}$ has associated costs. Cost is defined as follows:

$$\bar{d_{i,j}} = E_{y\ Pr(y|x_i)} d(y, y_i)$$

where $d(.|.)$ is the edit distance function, and $Pr(y|x_i)$ is given by the neural decipherment model.

4.2 Proposed Embedding Improvements

In this section, we discuss the changes we made in the embedding part of the baseline architecture to improve the performance of the model

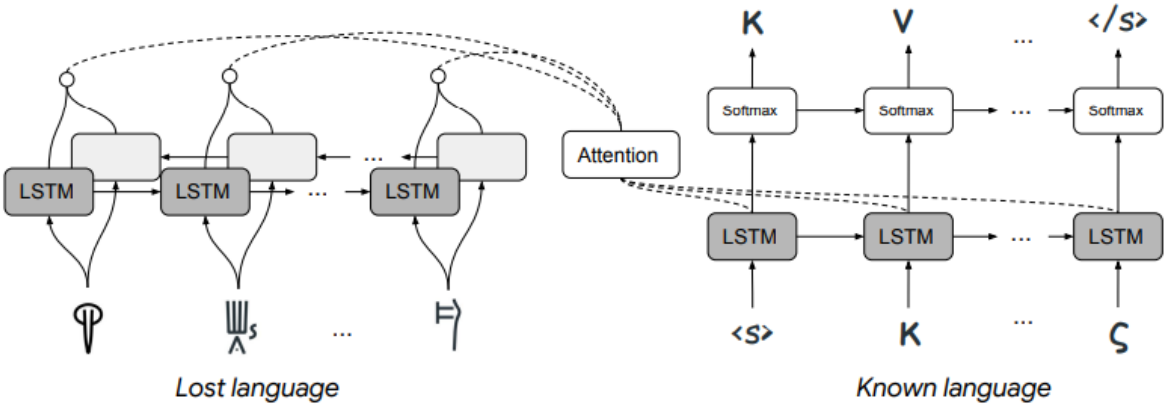

Figure 1: Seq2seq model architecture with lost language as input to encoder and known language as input to decoder. Residual connections linking weighted sum of embeddings and softmax not represented in the figure.

4.2.1 Normalization

We implemented normalization on the embeddings of lost language characters prior to them being feed to the generative framework. This helped in the alignment stage according to our analysis. We hypothesized that normalization will help adjust for the differences in the character embeddings caused by the structural difference between the pairs.

And thus improved the performance of our architecture over the baseline model. The performance of the model architecture with normalization can be seen in table 2.

4.2.2 Probabilistic Weight Initialization

We created a probabilistic dictionary of each character of the lost language being translated to a character of the known language. This dictionary was used during weight initialization for the characters, to multiply the weight created by embeddings of a character to the probability value of that character. Although on its own it didn't outperform the baseline architecture but in combination with the proposed architectural changes in the model its significantly improved the baseline models performance as can be seen in table 2.

4.3 Stacked Neural Architecture

Although the authors of the Baseline Architecture experimented with using multiple stacked BiLSTM layers with residual connection to improve their models performance they were unable to achieve the desired results and hence ended up using only one BiLSTM layer in their architecture, we were able to rectify the issues with their implementation and were able to extract better performance from the model by using multiple stacked BiLSTM layer in the proposed Architecture.

Model	Mode	Edit	Score	Accuracy
NP1	MLE	None	0.710	66.7
	Flow	False	0.710	
	Flow	True	0.787	
P1	MLE	None	0.715	51.1
	Flow	False	0.715	
	Flow	False	0.801	
N1	MLE	None	**0.738**	**69.7**
	Flow	False	**0.738**	
	Flow	True	**0.828**	
Baseline	MLE	None	0.692	65.9
	Flow	True	0.692	
	Flow	False	0.760	

Table 2: N: Baseline Architecture with Normalized Embedding, P: Baseline Architecture with Probabilistic Weight Initialization, #Number: Number of BiLSTM layers used in stacked architecture

5 Experiments

In this section, we discuss in detail all the experiments we conducted to develop our proposed architecture from the baseline model.

5.1 Experiment Settings and Dataset

In this Subsection, we give a brief overview of the experiment setup and datasets use to train and test our model. We also discuss about the extra dataset we added for training and testing, and the idea behind using that dataset.

5.1.1 Training configuration

Our Neural Architecture comprises of Bidirectional LSTM as the encoder and multiple layers of LSTM as the decoder. For all experiments we decided to fix the dimensionality of character embeddings and hidden size to 250. Here multiple layers are 1, 3 and 5. The size of the universal character inventory is 50 for all datasets except Linear B for which we use 100. The hyperparameter for alignment regularization is set to 0.5, and the ratio r to control the norm of the context vector is set to 0.2. We use ADAM to optimize the neural model. To speed up the process of solving the minimum-cost flow problem, we sparsify the flow graph by only considering the top 5 candidates for every xi. γ = 0.9 is used for the flow decay on all datasets except on UGARITIC for which we use γ = 0.25. We use the OR-Tools optimization toolkit4 as the flow solver. We ran all the experiments on a local machine which has 16GB of RAM and uses Nvidia RTX 3070 GPU with 8GB memory. The time required to train the model has come down to about 4hrs, with about 50% GPU usage.

5.1.2 Datasets

Given the nature of the task it is preferred that the low resource language is paired with a language that is most related to the language. We used the original Linear B and Ugaritic, from the data collected by (Luo, Cao, and Barzilay 2019a). While we attempted to create our own datasets using other lost languages or low-resource languages, we struggled to find any parallel corpus that was with a closely related family that was also large enough for training.

Linear B: Linear B is a language formerly used by the Minoans after being borrowed from the Myceanean Greeks in 1600 BC. In (Luo, Cao, and Barzilay 2019a) the authors also created an additional corpus using Linear B syllabograms and Greek location names to simulate a situation similar to an actual decipherment task.

Ugaritic: Ugaritic is a cuneiform writing system used around 15th to 13th century BC and is a member of the Northern Semitic languages. In (Luo, Cao, and Barzilay 2019a) the authors paired it with Hebrew, as it is also in the same Semitic language family. The Ugaritic - Hebrew language pair has been used for previous works on lost language decipherment (Snyder, Barzilay, and Knight 2010b).

Linear A: We extracted words from a github repository containing Linear A corpus [2]and manually searched and mapped Linear A characters to its speculated counterparts based on *Linear A Texts Inscriptions in phonetic transcription* n.d. We then paired these with the character for unpaired tokens '_' and created a cognate file with the Greek names dataset.

Asian: We also developed a corpus similar to **Romance** by combining similar Asian languages such as Japanese and Chinese, taken from WikiMatrix(Schwenk et al. 2019). This dataset was created by extracting Chinese and Japanese entries from Wikimatrix, using `fasttext` to only include entries classified as Chinese or Japanese, eventually we shuffled and sampled 919 items from the filtered dataset.

5.2 Normalization Enquires

We experimented with the effects of, normalizing the embeddings on the decipherment capabilities of the model. This was achieved by normalizing the embedded input prior to the LSTM step. We hypothesized that normalization helped with the alignment step since the performance of the baseline model with edit distance and min cost flow improved. This model was trained on the Ugaritic-Hebrew dataset, and we found that although the initial loss was notably higher than the baseline, the performance increased for the setting with min cost flow and editdistance. We also trained the model on the Linear A - Greek dataset. Unfortunately the model did not seem to pick up any notable patterns between the languages. Considering the lack of transparency in the linguistics of Linear A, and the fact that it is not certain how much similarity Linear A shares with Linear B except for its shared characters, this result may suggest that there is no significant overlap in the linguistic features of Linear B and Linear A. We also trained the model on the Chinese-Japanese dataset on the baseline model with normalization. The model performed somewhat well, scoring the highest on the setting with min cost flow and editdistance. The results are shown in Table 4.

[2]https://github.com/mwenge/lineara.xyz

5.3 Probabilistic Weight tests

We experimented with a function that maps characters/symbols from cognates of the lost language to the known language, forming a probability matrix of a particular character/symbol of the lost language to be translated to the characters/symbols in the known language, in order to initialize embeddings to be passed to the LSTM. But this approach did not yield good results, probably because the assumption of the characters' translation being consistent was too strong to form a sound embedding.

So, we created a probability dictionary of the certainty of each character of the lost language being translated to a specific character of the known language, and used this probability to initialise the weights of the character embedding. This led to an improvement of the results(see Table 2 and 3)

5.4 Stacked Architecture trials

The changes in the Neural Network was done only in the decoder section of BiLSTM. We reversed the flow of the network to get results from known-language to lost language. The reason we account for minute changes was because the Neural Network already is very complex and to make any substantial change will require to change whole network.

These changes failed to beat the state of the art. The cause for this result was due to the lack of cognates for lost-language which was hitting the performance of the model. Secondly, the baseline architecture performed with only single layer of LSTM. To see the performance of the model on different number of layers we chose 3 and 5 layers. The results were quite promising as now the LSTM cell can capture more features and have substantial history of words which increased the performance of the model(Table 2)

6 Results & Discussion

In this section we present the results of all the experiments we have done and discuss briefly about the outcome of those results.

6.1 Normalized Embedding Performance

The performance improvement gained by using embedding normalization, prior to being provided to the neural architecture, so as to generate better alignment mapping can be clearly seen in both Table 2 and Table 3 (by the performance of N and

Model	Mode	Edit	Score	Accuracy
NP1	MLE	None	0.710	66.7
	Flow	False	0.710	
	Flow	True	0.787	
NP3	MLE	None	0.724	68.0
	Flow	False	0.724	
	Flow	True	0.824	
NP5	MLE	None	0.710	75.3
	Flow	False	0.710	
	Flow	True	0.824	
N1	MLE	None	0.738	69.7
	Flow	False	0.738	
	Flow	False	0.828	
N3	MLE	None	0.729	75.3
	Flow	False	0.729	
	Flow	True	0.856	
N5	MLE	None	0.606	68.0
	Flow	False	0.606	
	Flow	True	0.656	
P1	MLE	None	0.715	51.1
	Flow	False	0.715	
	Flow	True	0.801	
P3	MLE	None	0.733	69.3
	Flow	False	0.733	
	Flow	True	0.828	
P5	MLE	None	0.715	**81.3**
	Flow	False	0.715	
	Flow	True	0.802	
Baseline 1	MLE	None	0.692	65.9
	Flow	False	0.692	
	Flow	True	0.760	
Baseline 3	MLE	None	0.729	70.7
	Flow	False	0.729	
	Flow	True	0.851	
Baseline 5	MLE	None	0.733	67.9
	Flow	False	0.733	
	Flow	True	0.828	

Table 3: Ablation Study Table-N: Baseline Architecture with Normalized Embedding, P: Baseline Architecture with Probabilistic Weight Initialization, #Number: Number of BiLSTM layers used in stacked architecture

Model	Mode	Edit	Score
N1	MLE	None	0.498
	Flow	False	0.493
	Flow	True	0.524

Table 4: Result of our proposed architecture on Asian dataset.

NP models).

6.2 Probabilistic Weight Initialization Performance

Furthermore, we also used a probability dictionary which mapped the probability of each character in the unknown language being translated to one certain character in the known language, and used this along with the preexisting inbuilt character embedding to initialize the weights of the character matrix. This led to an improvement in the results, because it gave higher weights to the characters in the lost language which were more certain to be translated to a specific character in the known language. As can be seen in table 2 (by the performance of model P and NP models).

6.3 Ablation Studies

We experimented with different configuration of Normalized Embedding, Probabilistic Weight Initialization and Stacked BiLSTM layers to see which configuration of the proposed architecture leads to best performance on decipherment of lost languages. Table 3. Highlights the results of different configurations of our proposed architecture as well as provides an idea of the performance of our model as compared to the different versions of the baseline architecture.

7 Conclusion and Future Work

While the model structure is meant to be language-agnostic, it will perform better if some engineering is done to better fit the language pair. For example, LinearB-Greek pair benefits from shifting the regularization terms by two to account for Linear B's syllabic structures. Similarly, we believe the Chinese-Japanese pair will benefit from customizing the regularization. In a hypothetical setting where Japanese is considered to be the known language, one may observe that the Japanese particles have a distinctly different shape than that of Chinese and Japanese Kanji since they are written in Hiragana. Using these differences they can adjust the alignment prediction step to shift it accordingly when it encounters a particle.

Overall, We were able to produce a better character based neural decipherment approach to find connection between unknown symbols of various lost language to symbols in already known languages than the current state-of-the art approaches. This was achieved by using a neural sequence-to-sequence model to captures character level cognate generation process.

In future work, we would like to use machine translation from one known language to another to be used as a base to develop frameworks for deciphering of unknown languages.

References

Guillaume Lample, Myle Ott, Alexis Conneau, Ludovic Denoyer, Marc'Aurelio Ranzato. 2018. *Phrase-Based & Neural Unsupervised Machine Translation*.

Kevin Knight, Anish Nair, Nishit Rathod, and Kenji Yamada. 2006. *Unsupervised analysis for decipherment problems*. In Proceedings of the COLING/ACL, pages 499–506

Nishant Kambhatla, Anahita Mansouri Bigvand, and Anoop Sarkar. 2018. *Decipherment of substitution ciphers with neural language models*. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, pages 869–874. Association for Computational Linguistics

Knight and K. Yamada. 1999. *A computational approach to deciphering unknown scripts*. In ACL Workshop on Unsupervised Learning in Natural Language Processing.

Snyder, Benjamin and Barzilay, Regina and Knight, Kevin. 2010. *A statistical model for lost language decipherment*. Association for Computational Linguistics

Taylor Berg-Kirkpatrick and Dan Klein. 2013 *Decipherment with a million random restarts*. In Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, pages 874–878. Association for Computational Linguistics.

Jiaming Luo and Yuan Cao and Regina Barzilay. 2019 *Neural Decipherment via Minimum-Cost Flow: from Ugaritic to Linear B*.