Morphological Analysis on Jeju and Korean Using Transformer Models

Hyunjoo Cho
(hyunjoo.cho@student.uni-tuebingen.de)

Eberhard Karls University of Tübingen, Germany

This study explores the morphological differences between the critically endangered Jeju dialect and Standard Korean through transformer-based modeling. Parallel corpora were collected, cleaned, and morphologically tokenised using the KoNLPy toolkit before training two models: an autoencoder for Jeju-to-Jeju reconstruction and an encoder-decoder for Jeju-to-Korean translation. Experimental results demonstrated that the Jeju-to-Jeju model achieved notably lower losses compared to the Jeju-to-Korean model, indicating the significant linguistic and morphological divergence between the two languages despite their syntactic similarity. These findings provide computational evidence of Jeju's unique linguistic status and highlight challenges in modeling Jeju under-resourced languages. The approach offers potential applications in digital preservation and revitalization of Jeju to non-native speakers such as a translator or an online learning platform, and suggests directions for future research, including the development of dedicated Jeju morphological parsers, multilingual transfer learning, and expanded datasets.

0 INTRODUCTION

Jeju is the largest island in South Korea with the size of 1,833km^2 [1]. It's a volcanic island where it has a resting volcano called Halla-mountain in the center. There are four most popular places in Jeju which are: Gimnyeong Beach on the northern end, Songak Mountain on the southern end, Suwol Peak on the western end, and Seongsan Ilchulbong on the eastern end [2].

However, due to the increase usage of Seoul Korean in media and online, Jeju is losing native speakers slowly over the time. The fluent speaker of Jeju are now about 1% [3], and among those people most people are over the age of 75 [4], and many fluent speakers remaining in Jeju now speak a variety of Korean with a Jeju substratum. According to UNESCO in 2010, Jeju was classified as critically endangered [5]. Critically endangered is the notable category before the level of Extinct [6]. Therefore, quite some effort from the local people and the government are ongoing. With my special interest in Jeju, it lead me to run on an experiment on Jeju and the main language, Korean. The goal of the project is to train models for jje-jje and jje-kor to test similarity.

1 BACKGROUND

Jeju, which is also referred as Jejueo or Jejuan in English language scholarship [7], is a Koreanic language from Jeju island, South Korea. Comparing Korean and Jeju, they are not mutually intelligible. As a consequence, there is an on-going argument if Jeju is supposed to be considered as a historical divergent of the Korean language, or as a separate language in its own right [8].

The consonants of Jeju are similar to Korean, but Jeju has a larger vowel inventory. Nouns and verb inflections of Jeju are so different to Korean that many Koreans will never understand Jeju sentences if they do not learn them; above all, Korean and Jeju differ significantly in verbal paradigms [9].

2 DATA

To compare the morphological differences between Jeju and standard korean, I looked for data on Hugging Face. Among all data set, `jeju_potato_datasets`, `jeju-dialect-to-standard`, and `jejueo_interview_transcripts` were chosen because these had well prepared parallel data for both Jeju and Korean. Many of the other data sets had speech recording data but not transcriptions. After downloading three datasets, I processed them through to collect only Jeju

transcriptions and Korean transcriptions. As all files have different structures of json, parquet and csv, three different processing scripts were necessary. The json file had data of speakers information, audio information, speech transcription information, and extra annotation. As the file was neatly organised, it was easy to extract only transcription from the json file using the key-value pair. For parquet files, it was not complicated to read parquet files, and it only contained Jeju data and Korean data without extra information. These two data were able to be easily extracted. Finally, the csv file had sentences with subject's personal information, with some annotation of how Jeju and Korean differ in the file. As the csv file had the column of Jeju and Korean separate as well, putting it to a separate file was an easy process.

At the end, all downloaded data were combined together in `jje-kor.tsv`. However, as data from Hugging Face were not cleaned, further processing was required to remove anonymisation masks and standardise punctuation. There were special characters such as `@name` or `@address` which were masking the real names of the people. Since the masked sentences were few, deletion did not significantly affect dataset size.

Inspecting the file, there were many duplicated lines which are not meant to be repeated. Hence, another `sort.py` file has been written to sort the sentences in an alphabetical order. Therefore, if there are few repeated sentences, it can be eliminated. Finally, the sorted and cleaned file has been saved to `jje-kor_sorted.tsv`.

3 MORPHOLOGICAL PREPROCESSING

The purpose of the paper is to train a model using morphemes of two languages, Korean and Jeju. Therefore, it was necessary to tokenise all sentences to morphological level. There is a natural language processing package for Korean which is called KoNLPy [10]. In KoNLPy, there are five different sub-packages that can parse phrases into morphemes: Hannanum, Kkma, Komoran, Mecab, Okt. To decide what morphological parser is the best for the project, each of the package has been gone through a given example sentence. Comparing the result of parsings, Kkma was decided to be used for the project because the result seemed to be most reliable among the other results. The decision was based on the visual inspection.

Kkma is the morphological analyser and POS tagger written in Java basing on Korean. Hence, the performance of Kkma on Korean was satisfying; nonetheless, its performance on Jeju was to be determined via a few example sentence processing. To make sure Kkma works to a satisfying extent, a few Jeju example sentences were morphologically tokenised. Examining the output, performance on Jeju appeared sufficient upon visual inspection. Which convinced me to use Kkma for Jeju as well along side with Korean. The final `jje-kor_sorted.tsv` was gone through Kkma morphological parser, and the final output for the model training was saved to `jje-kor_tokenised.tsv`.

The format of the data set is as shown in figure 1. The sentence starts with the Jeju sentence, and continued with Standard Korean. These two sentences are separated with 'tab', and both of the sentences are morphologically tokenised.

(1) 듣도 못 해신 디 . 듣도 못 하 었 는데.
deutdo mot haesin di . deutdo mot ha eot nunde.
(I) had never even heard of it.

Fig. 1. An example Jeju and Korean sentence from `jje-kor_tokenised.tsv`

The whole file was checked through using regex to find if there are extra maskings or special characters left. Since there are two models to be trained, two parallel files are required: Jeju to Jeju and jeju to Korean. Hence, as already Jeju to Korean file is prepared, Jeju to Jeju file had been prepared by copying the first column of Jeju into two columns. Which looks like figure 2.

(2) 듣도 못 해신 디 . 듣도 못 해신 디 .
deutdo mot haesin di . deutdo mot haesin di .
(I) had never even heard of it.

Fig. 2. An example Jeju and Jeju sentence from `jje-jje_tokenised.tsv`

At the end, two files of jje-jje and jje-kor were prepared. Before the files are fed into models, they had to be separated into three files of train, validation, and test sets. By randomly selecting lines, the whole dataset was split into train set which was 80%, validation set which was 10% and, and test set which was the last 10%.

4 METHODS

4.1 DESCRIPTION OF THE MODEL

It was necessary to train the model twice with two input dataset: Jeju to Jeju, which was the baseline of the experiment, and Jeju to Korean. Among all possible models, transformer with sequence to sequence encoder-decoder model was chosen to be used.

For Jeju to Jeju, an auto-encoder model was chosen because it's simply doing a copy mechanism from Jeju to Jeju. Auto-encoder model is a type of neural network architecture that encodes input data down to some features, then decode back to the original input. [11] Within the simple copy model, model will be fed Jeju form in and

have to decode and reconstruct the same Jeju sentence.

Then for Jeju to Korean, a normal encoder-decoder model was used because here the input and ouput format is different. To use auto-encoder model here as well, a transfer learning was required by changing the last layer of the model to output Korean instead of Jeju; however, this was not necessary to see the variation between two languages of Jeju and Korean. In jje-kor model, it will be fed with Jeju form, and the decoder should reconstruct it to the equivalent Standard Korean form.

After training two separate models, it was tested using test sets. By evaluating the model on test sets, it showed reconstruction loss of two models, which would be used to later on analyse morphological diverse of two languages.

4.2 TRAIN SET UP

To train a transformer model, vocabulary file is necessary because transformers operate on token IDs not directly on raw text. Hence, vocabulary files that define the mapping between morphemes and token IDs is required to train the model. The file was first built with the whole data with minimum occurrence frequency of 1. However, when the minimum frequency was 1, the vocabulary file was too big to be trained. Consequently, I shifted the minimum frequency to be 5; which reduced input and output dimension, which helped with training speed by reducing model size and the space of the output options. Furthermore, the morphemes that had a fewer frequency than 5 would have less data and high variance, which showed less value.

For the purpose of training Jeju to Korean, two vocabularies were prepared: one for Jeju and one for Korean. The final size of Jeju vocabulary is 29793 and the final size of Korean vocabulary is 23406.

After forming vocabulary, hyper-parameters were set up for the model. Both encoder and decoder had several common configuration features: hidden size, number of hidden layers, number of attention heads, intermediate size, and max position of embeddings. These values were experimented to find the best for the model to perform well and does not take too long to train. Two different config set ups were used, which are shown in figure 3 and 4.

$config_1$	
hidden_size	512
num_hidden_layers	6
num_attention_heads	8
itermediate_size	1024
sequence_length	32

Fig. 3. A model config set up for a bigger trainer.

$config_2$	
hidden_size	128
num_hidden_layers	3
num_attention_heads	2
itermediate_size	512
max_position_embeddings	32

Fig. 4. A model config set up for a smaller trainer.

Both configurations were trained with Jeju to Jeju dataset, and comparing the performance of the models obtained with the different configurations, the bigger model was performing significantly better than the smaller model. The loss function was cross entropy. Hence, it has been decided to be used. The training results were shown figure 5.

	jje-jje smaller trainer	jje-jje bigger trainer
train loss	0.15	0.0075
eval loss	0.070	0.039

Fig. 5. Training loss on the Jeju to Jeju training set and evaluation loss on the Jeju to Jeju validation set.

From the data, the bigger trainer has a better train loss compared to the smaller trainer. Also, bigger trainer's evaluation loss is roughly half of smaller trainer's loss. Within these two results, the bigger trainer was chose to be used for the experiment.

Other configurations were kept constant: vocab_size which was set as the size of Jeju vocab, hidden_dropout_prob which was set as 0.1, and attention_probs_dropout_probs which was set as 0.1 as well. For the encoder model, it had is_decoder and add_cross_attention as false; meanwhile, the decoder model had both of the features as true. Last but not least, to set up the model, it was necessary to specify the indices for the special token IDs for <PAD>(padding token), <BOS>(beginning of sentence token), and <EOS>(end of sentence token).

To use the dataset in the model, it is necessary to instantiate a method that prepare paired input and ouput sequences for training a sequence-to-sequence model on sentence data. This was written as `JjeKorDataset` class with the path to the parallel corpus and the rebuild vocabularies. The sentences were vectorised with the vocab indices and fed to the models. While encoding, for each sentence, it also added special tokens of <BOS>and <EOS>to mark the sentence boundary. Then, the following prepared token indices will be ready to be fed into a neural model.

The dataset stored these aligned input–output pairs and provides them through the __getitem__ method as PyTorch tensors, which can be efficiently loaded in batches during training. The __getitem__ method retrieves one aligned

Jeju–Korean sentence pair at a time. For a given index, it looks up both sentences in the dataset, converts their token ID sequences into PyTorch tensors of type long, and returns them as a tuple. This makes each data sample directly usable by the model. When used together with PyTorch's DataLoader, these samples can be automatically batched and shuffled, allowing efficient training without extra preprocessing. Hence, with ready vocab files, prepared models, and raw data converter, training started.

4.3 TRAINING

The training set that had been prepared was fed to an auto-encoder Jeju to Jeju that was trained on config_1 in figure 3. The training was done on T4 GPU on Google Colab. Then, model path had been set up to the directory where `jeju-tranformer-final` was saved.

Two datasets, the training set and the validation set, were prepared and converted into the appropriate format for model training. To handle batching, I applied a collate function that ensured all sequences within a batch had the same length. Any incomplete or invalid items were filtered out, and the sequences were truncated or padded to a fixed maximum length of 32 tokens. Each batch contained 8 sentence pairs, where the source and target sequences varied depending on the training objective: for Jeju to Jeju, both sides consisted of Jeju sentences, whereas for Jeju to Korean, the source was Jeju and the target was Korean.

By standardising sequence length to 32 tokens, the model could process batches efficiently in parallel while avoiding excessive padding. This choice balanced training stability with computational efficiency, ensuring consistent GPU utilisation.

At the end, trainer was ran with training arguments. In training argument, it included the following features shown in the figure 6.

output_dir	"./results"
num_trained_epochs	3
per_device_train_batch_size	8
weight_decay	0.01
logging_dir	"./logs"
learning_rate	5e-5
save_total_limit	4

Fig. 6. Training argument features.

I set epochs as 3, batch size as 8, and learning rate $5e^{-5}$. As I used a bigger trainer model, with Google Colab T4 GPU [12], it took 6 hours to train. Even if the smaller model only took 2 hours and half, due to the performance level, bigger train was used to train. The result could be seen in 7. After 6 hours each for Jeju to Jeju and Jeju to Korean, training had been finished.

5 EVALUATION

The loss values are reported as the training was done. To make sure the model does not only work on the training data, it was evaluated on test data set. Which resulted the loss shown in figure 7.

result	jje-jje	jje-kor
training loss	0.0075	0.03959
evaluation loss	0.0386	0.2184
test loss	0.037	0.215

Fig. 7. Model evaluation.

6 EXPERIMENTAL RESULTS

Comparing the loss values of two models, model trained on Jeju to Jeju performed better with a test loss of 0.037 compared to a test loss of 0.215 for Jeju to Korean. This could be interpreted that as Korean is morphologically diverse from Jeju. Even if Korean and Jeju have very similar syntactic structure as Jeju is a branch of Korean, many of vocabs diverse. Verb forms changes similar in Jeju and Korean, but as many nouns are different, it might be harder for the model to learn.

7 DISCUSSION

In general the training of the transformer was successful based on the loss result on training set, validation set, and test set. Furthermore, via a visual evaluation through manual inference showed promising results. An example output of Jeju to Jeju is shown in figure 8.

(3) 거 사투리 랜 . 거 사투리 <UNK>.
geo saturi len . geo saturi <UNK>.
That is a diaelct.

Fig. 8. An example output generation of Jeju to Jeju trained model

It was expected that Jeju would be diverging from Korean, so the goal of the experiment was reached. Though, that low loss value for Jeju to Korean was unexpected. In my perspective, Jeju contrasts quite a lot from Korean. Which lead me to expect a higher loss value like 0.1 or 0.2 for Jeju to Korean.

Unfortunately, there are quite some limitations I had. The first limitation is that Kkma was not built for Jeju. Even if it was attempted to check if the parser works for Jeju, it does not guarantee that every single sentence would be parsed correctly. Hence, it would be better if time allowed, to build a separate morphological parser for Jeju.

To improve the accuracy, the model might have performed better if I had used a fine-tuned version, for exam-

ple one trained on Korean–Japanese data. Since Japanese is relatively close to Korean, such a model could have provided useful benefits for training.

The study was limited by available technological resources and runtime error of Colab service.

For the current project, the sentences that included masking words were ignored. Most of the masking words were anonymity masks. In our everyday life, these words are inevitable. It would be useful if the model were also trained on these sentences. Therefore, if those special sentences are collected and processed to be treated specially, it would help models to learn those special tokens. This could be done by substituting random sampling from the list.

8 CONCLUSION

In this study, I investigated the similarity between Jeju and Korean using transformer based sequence to sequence models. There were two main tasks: Jeju to Jeju reconstruction using an auto-encoder model, and Jeju to Korean translation using an encoder-decoder model. The result showed that Jeju to Jeju model achieved lower training, evaluation, and test losses. While the Jeju to Korean model showed higher losses (See figure 7). Difference in performance and the challenges in computational modelling might be due to different factors such as different morpheme inflections, but it hints that the languages aer quite distinct.

The work has potential applications in the preservation and revitalisation of Jeju via developing translation tools, educational resources, and digital archives for future generation. The future application that I would like to focus on would be the translator as this would easily increase the interest in Jeju for non-native Jeju speakers. To start with, the improvements could be made by developing a morphological parser specifically for Jeju that can deal with unique Jeju's linguistic features. This would also open wider doors for NLP workers who are interested in Jeju. To enhance the performance of the transformer, Korean Japanese pre-trained model can be adapted in the algorithm layer. Moreover, it would be necessary to expand dataset to include masked entities that are mentioned earlier in the paper.

9 REFERENCES

[1] "Flexible renewable energy planning based on multi-step forecasting of interregional electricity supply and demand: Graph-enhanced AI approach," .

[2] "Exploring Jeju Island," .

[3] "Only approximately 1% of Jeju citizens now speak Jeju language fluently," .

[4] M. Saltzmann, "A History of Jejueo," .

[5] "Jejueo: The endangered language of Jeju Island," .

[6] UNESCO, "UNESCO Project: Atlas of the World's Languages in Danger," .

[7] DBpedia, "About: Jeju language," .

[8] "Distinction between dialects and languages: The case of South Korea's Jeju language and Gyeongsang Dialect," .

[9] "Jejuan," .

[10] E. L. Park, S. Cho, "KoNLPy: Korean natural language processing in Python," (2014 October), accessed on Sep 5, 2025.

[11] D. Bergmann, C. Stryker, "What is an autoencoder?" (2023 November).

[12] *NVIDIA TURING GPU ARCHITECTURE* (2018), accessed on Sep 8, 2025.