
VIETify - Vietnamese Diacritics Restoration

Cao Thuy An, Tran Ba Hoang, Hoang Van Khanh, Trinh Thi Minh Tam

Ha Noi Team

Abstract

Restoring missing diacritics in Vietnamese text is crucial for maintaining linguistic accuracy and comprehension. This research paper presents a comprehensive study involving two distinct approaches for diacritics restoration. The first approach employs a language model with beam search, enhancing prediction accuracy by exploring multiple possible diacritic sequences. The second approach utilizes a transformer-based encoder-decoder architecture to restore diacritics and learn the mapping between unaccented and accented text. Both approaches are thoroughly implemented and evaluated using established metrics, showcasing their respective effectiveness in diacritics restoration. This research underscores the significance of diacritics restoration and offers insights into the capabilities of diverse techniques for preserving linguistic fidelity in Vietnamese text.

1 Introduction

The Vietnamese language is renowned for its distinctive tonal nature, characterized by the use of diacritics, or accent marks, that play an indispensable role in conveying meaning and pronunciation accuracy. These diacritics modify the tones of the basic alphabet letters, transforming simple words into a rich tapestry of nuanced meanings. While the diacritics in Vietnamese are critical for comprehensibility and semantic precision, they are frequently omitted in various contexts, such as digital communication, informal writing, and even due to technical limitations. The omission of diacritics, although common, often leads to ambiguities, misinterpretations, and an overall reduction in the linguistic richness of the text.

In Vietnamese, diacritics are used extensively, contributing to the language's depth and subtlety. The language employs six different tones to differentiate the meanings of words, and a single word can have various meanings based on its tonal pronunciation. Diacritics act as essential markers to distinguish between these tones, which can drastically alter the meaning of a sentence. For instance, the word "ma" with different diacritics can mean "mother"(má), "rice seedling" (mạ), "tomb" (mả), or "horse" (mã), underscoring the intricate interplay between diacritics and semantics.

2 Methods

2.1 Data

The dataset was downloaded from wikipedia, which restores a large number of Vietnamese articles from various sources and was extracted using wikiextractor package.

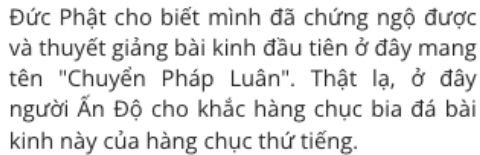
After extracting only text from the original data, some preprocessing steps were carried out:

- Split the text into sentences, each line has one sentence
- Remove numbers, special characters and punctuations
- Convert the sentences to lowercase
- Retain sentences with more than 10 words and less than 200 words
- No-tone sentences were generated by removing tone from the text.

Dataset Split After preprocessing, the dataset consists of a total of 4,315,334 sentences with diacritics and is about 1.1 GB. The dataset is shuffled and divided into train/val/test with the below config. Each of them includes two files: tone sentences with no-tone sentences.

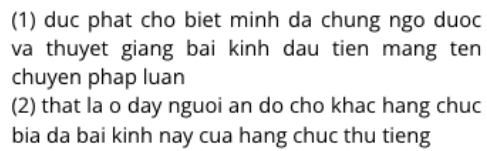
- Train dataset: 4,295,334 sentences
- Evaluate dataset: 10,000 sentences
- Test dataset: 10,000 sentences

Here is an example of data format after preprocessing.



Đức Phật cho biết mình đã chứng ngộ được
và thuyết giảng bài kinh đầu tiên ở đây mang
tên "Chuyển Pháp Luân". Thật lạ, ở đây
người Ấn Độ cho khắc hàng chục bia đá bài
kinh này của hàng chục thứ tiếng.

Figure 2: Train data with tone sentences



(1) duc phat cho biet minh da chung ngo duoc
va thuyet giang bai kinh dau tien mang ten
chuyen phap luan
(2) that la o day nguoi an do cho khắc hàng chục
bia da bai kinh nay cua hàng chục thu tieng

Figure 3: Train data with no-tone sentences

2.2 Models

The challenge this problem addresses is to transform an unaccented sentence into an accented one while preserving its information and meaning. The input to the problem will be an unaccented sentence, and the output will be a fully accented sentence. For instance:

- Input: "anh dang o dau the"
- Output: "anh đang ở đâu thế"

In this project, two approaches are proposed to tackle this problem:

- **Using Language Model + Beam Search:** This approach involves utilizing a language model along with beam search. A language model predicts appropriate diacritics for each word in the unaccented sentence. The beam search algorithm explores various possible diacritic combinations, selecting the one that best fits the context and preserves the intended meaning of the sentence.
- **Treating it as a Machine Translation:** Another strategy is to reframe the problem as a machine translation task, where the source language is unaccented Vietnamese sentences, and the target language is fully accented Vietnamese sentences. This approach employs an Encoder-Decoder architecture using the Transformer model. The Encoder processes the input sentence without diacritics, and the Decoder generates the accented version of the sentence.

Both of these approaches leverage advanced language processing techniques to ensure accurate diacritic restoration while maintaining the semantic integrity of the original sentences. The details of these two approaches are investigated more in section 3.1 and 3.2

3 Experiments

3.1 Language model

3.1.1 Overview

A statistical language model is a probability distribution over sets of texts. Put simply, a language model can indicate the probability of a sentence (or a phrase) belonging to a language. Language models are commonly applied in tasks such as spell-checking, machine translation, and more. For instance, when applied to Vietnamese:

$$P[\text{"hôm nay là thứ hai"}] = 0.01$$

$$P[\text{"hôm nay hai là thứ"}] = 0$$

The example above shows that the sentence "hôm nay là thứ hai" is more likely to be a Vietnamese sentence than the sentence "hôm nay hai là thứ". So the next question is how do we calculate that probability, or how to build a language model?

The general formula for a language model is:

$$P(w_1 w_2 \dots w_n) = P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_1 w_2) \cdot \dots \cdot P(w_n|w_1 w_2 \dots w_{n-1}) \quad (1)$$

Based on equation 1, it is necessary to have a large dataset and a substantial amount of memory to store the entire probability distribution among word sequences. However, this becomes infeasible in practice. Consequently, the N-gram model emerged to address this limitation. The n-gram model calculates probabilities solely based on the context of the previous n-1 words. In this scenario, due to the complexity, we use the Markov formula as follows:

$$P(w_1 w_2 \dots w_n) \approx P(w_{n-N+1} w_{n-N+2} \dots w_{n-1}) \quad (2)$$

Another problem is that as calculating probabilities involves many cases, encountering unseen n-grams or uneven distribution within the training set can lead to incorrect calculations. To address this, a smoothing method is introduced.

Some common smoothing methods are as follows:

- Discounting: Reduce the probability of n-grams with probabilities greater than 0 to compensate for unseen n-grams.
- Back-off: Calculate the probability of unseen n-grams using shorter n-grams with non-zero probabilities.
- Interpolation: Calculate the probability of all n-grams using shorter n-grams.

These smoothing techniques are employed to address the issues of unseen n-grams and uneven distribution, improving the accuracy of probability calculations.

In this project, we chose Kneser-Ney smoothing as an interpolation method.

- Kneser-Ney is a primary method used to compute the probability distribution of n-grams within a document based on existing calculations.
- Kneser-Ney is regarded as the most effective smoothing method because it uses absolute discounting by subtracting a fixed value from the lower-order frequency of the probability to discount n-grams with lower frequencies. It is particularly effective for bigrams and higher-order n-gram models.
- Formula

$$P_{KN}(w_i|w_{i-1}) = \begin{cases} \frac{\max\{C(w_{i-1}w_i) - D, 0\}}{C(w_{i-1})} & C(w_{i-1}w_i) > 0 \\ \alpha(w_{i-1})P_{KN}(w_i) & \text{otherwise} \end{cases}$$

where

$$P_{KN}(w_i) = \frac{C(\bullet w_i)}{\sum_{w_i} C(\bullet w_i)}$$

and $C(\bullet w_i)$ is the number of unique words preceding w_i .

$\alpha(w_{i-1})$ is chosen to make the distribution sum to 1:

$$\alpha(w_{i-1}) = \frac{1 - \sum_{w_i: C(w_{i-1}w_i) > 0} \frac{\max\{C(w_{i-1}w_i) - D, 0\}}{C(w_{i-1})}}{1 - \sum_{w_i: C(w_{i-1}w_i) > 0} P_{KN}(w_i)}$$

Figure 4: Kneser-ney fomula

3.1.2 Language model training

We use an available pipeline for building an n-gram model with nltk library. Due to time and computational hardware, the model was trained only with 2-gram and 3-gram versions. The model training time for each version is more than 2 hours on our personal computer with CPU Intel Core i5 and 10GB RAM.

3.1.3 Vocabulary set

During the prediction process, we need to know how many potential accented word forms a given word can generate. For instance, for the word "*cho*" all the possible accented forms that can be generated are *cho*, *chó*, *chợ*, *chở*, *chờ*, *chỗ*. To obtain this mapping dictionary, it is necessary to construct a vocabulary set of words. Fortunately, Luong Hieu Thi - the author of the article "All syllables in the Vietnamese language" has built a library on GitHub about syllables in Vietnamese. Now the work left is quite simple, the mapping dictionary is created by removing diacritics from the words in the vocabulary set.

For example with the word "*hoang*", its diacritical words version is "*hoang*", "*hoàng*", "*hoáng*", "*hoăng*", "*hoǎng*", "*hoạng*", "*hoảng*", "*hoǎng*", "*hoằng*", "*hoẵng*", "*hoặng*".

3.1.4 Pipeline with language model

In conclusion, here are the steps that we carried out using a language model.

1. Language Model Training
2. Consider each word in the input no-tone sentence
3. For each word being considered, use the mapping dictionary constructed in Section 3.1.3 to generate possible accented forms of the word. This results in generating multiple possible sentences. For words not in the mapping dictionary, directly append them to the output sentence and proceed to the next word.
4. Use the beam search algorithm to identify the sentence with the highest probability as the output.

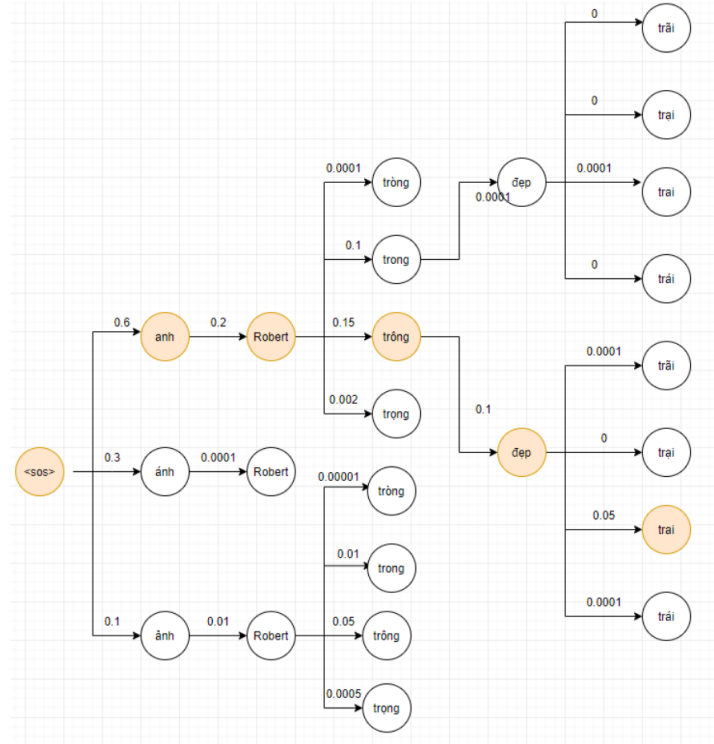


Figure 5: Example with beam search

3.2 Transformer

The Transformer model follows the Encoder-Decoder architecture and has gained increasing popularity not only in natural language processing but also in computer vision tasks. Transformer has achieved remarkable results in various tasks, particularly in machine translation, where it has achieved state-of-the-art performance.

The Transformer consists of two main parts: the Encoder, which encodes the input sentence (source language), and the Decoder, which generates the output sentence (target language). The Encoder consists of 6 stacked identical Encoder layers. Each layer contains sub-layers: self-attention, layer normalization, and fully connected layers. The Decoder follows a similar structure to the Encoder.

The use of self-attention in the Transformer model accelerates training due to its parallel processing capability, unlike previous sequential models like LSTM.

The Transformer model excels in tasks involving contextual understanding, such as machine translation and text summarization. For the task of adding accents to Vietnamese text, context is crucial, as different contexts lead to different predictions. Furthermore, this task can be framed as a machine translation problem, with a source language (unaccented sentence) and a target language (accented sentence), making it suitable for the Encoder-Decoder architecture of the Transformer.

3.2.1 Vocabulary set

In the Encoder-Decoder architecture, it's necessary to construct vocabularies for both the input and output. In this task, the vocabulary consists of an unaccented word vocabulary for the Encoder and an accented word vocabulary for the Decoder. We constructed word-level vocabularies as inputs and outputs for the problem.

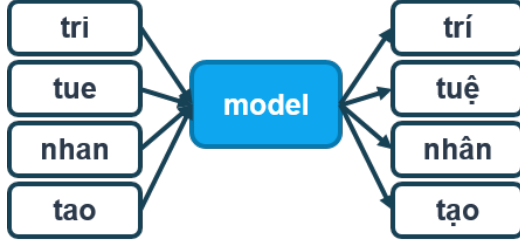


Figure 6: Word2Word level model

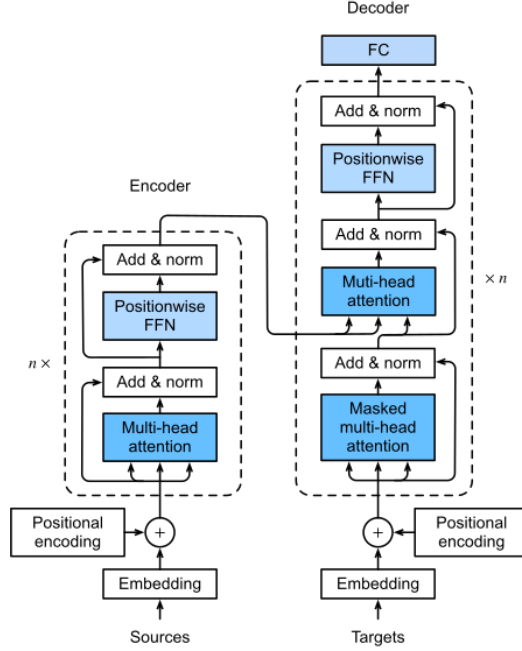


Figure 7: Transformer model

Similar to building vocabulary for the n-gram model, we use tokenize function of nltk library and Vietnamese syllables to tokenize the input and its ground truth. Besides, special tokens like <unk>, <eos>, <sos>, and <pad> were added for use during training.

3.2.2 Model training

To explore and deeply understand the model's architecture, it was built from scratch instead of directly implemented. We built from the small components like self-attention blocks until the whole encoder-decoder blocks.

Name	Value
Encoder layers	6
Decoder layers	6
Heads	8
Hidden size	512
Dropout	0.1

Table 1: Model details information

Name	Value
Epochs	25
Optimizer	Adam
Initial learning rate	10e-9
Beta ₁	0.9
Beta ₂	0.98

Table 2: Training information

The learning rate is scheduled for adjustment after a certain number of steps. Our testing phase, conducted on a subset of the dataset, shows a significant decrease in convergence time. Encouraged by these results, we applied this approach to the complete original dataset. The concrete formula is depicted below:

$$\text{learning_rate} = d_{\text{model}}^{-0.5} \cdot \min(\text{step_num}^{-0.5}, \text{warmup_steps}^{-1.5}) \quad (3)$$

In this experiment, the training data batch will be created based on the number of tokens, optimizing GPU VRAM usage. Each batch size will consist of 6000 tokens and every 30000 tokens will be considered one step (one parameter update during training).

The model is trained on AWS EC2 instance, utilizing a T4 GPU with 16GB VRAM. The training process spans over a single day.

4 Results

The experimental results were evaluated on a dataset comprising 10,000 sentences. To assess the approaches, we employed an accuracy metric as follows:

$$\text{accuracy} = \frac{\text{number of correctly predicted words}}{\text{total number of words}}$$

For example: the predicted sentence is "*cô ấy trông kia xinh gái quá*" while the correct sentence is "*cô ấy trông kia xinh gái quá*". It's observed that the word "*gai*" was predicted as "*gài*" incorrectly. In this case, the accuracy score would be: $\text{score} = \frac{6}{7} = 0.86$

The achieved results are as follows:

Table 3: Comparison of Methods and Results

Method	Accuracy	Processing Time
2-gram with beam-size = 3	86.03%	3.6s
2-gram with beam-size = 5	86.55%	4.59s
3-gram with beam-size = 3	87.60%	4.26s
3-gram with beam-size = 5	88.50%	5.84s
Transformer	97.96%	1.15s

When considering n-gram models, a greater value of 'n' leads to an increased accuracy percentage. Nevertheless, the computational processing times reveal the limitations in practical production scenarios.

Conversely, the Transformer model delivers outcomes aligned with our objectives. It efficiently processes a sentence in just about one second on average.

5 Conclusion and Discussion

We contend that both n-gram models and transformer models exhibit distinct advantages and disadvantages.

What we achieved

- Implementing and training beam search, transformer model from scratch provides a deeper comprehension of the model and the task
- The transformer and n-gram models exhibit satisfactory levels of accuracy, when it comes to the challenging Vietnamese language — where even fully diacritical sentences can make readers confusing
- We build an browser demo on Streamlit including all the models that we trained, which you can try on this <http://52.42.174.2:8501/>
- We also built an API and Dockerfile to facilitate the deployment and accessibility of our trained models.

Limitations

- The n-gram model time prediction is quite slow. This can be done by applying kenlm - a language model toolkit built on C++
- The implementation and training of transformer model demands time and effort. But this can be solved by utilizing the pretrained tokenize, model or platform like HuggingFace

6 References

- [1] Do DT, Nguyen HT, Bui TN, Vo HD. PRICAI'21. VSEC: Transformer-based Model for Vietnamese Spelling Correction (2021)
- [2] Tran, Hieu, et al. IEA/AIE'21. Hierarchical Transformer Encoders for Vietnamese Spelling Correction (2021)
- [3] Dang TDA, Nguyen TTT. PACLIC'20. TDP – A Hybrid Diacritic Restoration with Transformer Decoder (2020)
- [4] Nguyen, Ha-Thanh, Dang, Tran, Nguyen, Le. Deep Learning Approach for Vietnamese Consonant Misspell Correction (2020)
- [5] Bui, Hung. Vietnamese Diacritics Restoration Using Deep Learning Approach. 347-351 (2018)
- [6] Jakub et al. LREC'18. Diacritics Restoration Using Neural Networks (2018)
- [7] Pham et al. IALP'17. On the Use of Machine Translation-Based Approaches for Vietnamese Diacritic Restoration (2017)