Turkish Diacritization with Deep Learning

Emirberk Almacı
AI and Data Engineering
Istanbul Technical University
150220751
almaci20@itu.edu.tr

Furkan Öztürk
AI and Data Engineering
Istanbul Technical University
150200312
ozturkahm20@itu.edu.tr

Abstract

This project addresses the challenge of diacritization in the Turkish language through the implementation of deep learning methodologies. Diacritization, the addition of diacritical marks, plays a critical role in the correct interpretation of Turkish text as it can significantly alter word meanings. The goal of this research is to develop a robust deep learning model that achieves high accuracy in automatically applying diacritics to Turkish text. This will enhance readability and comprehension in automated text processing applications. The approach involves preprocessing textual data, followed by the design and training of a neural network that learns the contextual usage of diacritics in written Turkish.

Introduction

Problem

Diacritics play a crucial role in many languages, as they provide essential phonetic information that alters the pronunciation and meaning of words. However, due to keyboard constraints and individual preferences, the widespread use of ASCII equivalents for diacritic letters has become common. This practice, while convenient, presents challenges, especially in languages like Turkish, where ASCII equivalents are valid letters in the alphabet.

In Turkish, the words "sinir" and "sınır" illustrate the importance of diacritization. Without diacritics, both words are written as "sinir" in ASCII form. However, with diacritics, "sinir" means "nerve" or "irritation," while "sınır" means "border" or "limit". This ambiguity demonstrates the challenge of accurately diacritizing words written partly or entirely in ASCII form. To achieve accurate diacritization, it is necessary to consider not only the current token but also neighboring words to disambiguate between possible meanings.

This task becomes even more complex in contexts where highlighting is vital to understanding the intended meaning, such as religious texts or legal documents. Therefore, it is important to develop powerful accenting algorithms to transcribe texts accurately and preserve their intended meaning.

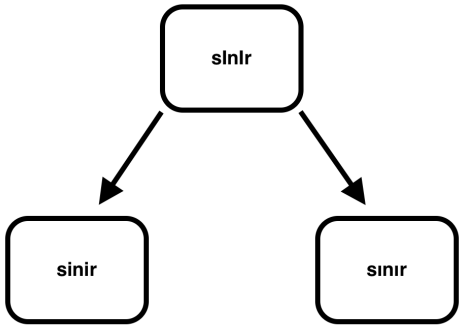

Figure 1: Example Word

Dataset

The project consists of 2 data sets. The first of these is the train.csv file. There are around 53 thousand rows in the data set provided by the project. Each line contains 1 sentence. There are approximately 95 thousand words in these sentences.

As an addition to this data set, a data set containing Turkish words from the internet was used. There are approximately 115 thousand Turkish words in the file used as random_data.txt from Turkish Data Depository (http://data.tdd.ai).

These datasets together are only intended to train a more effective and versatile model. In this way, a higher success rate is aimed at solving the problems that will be encountered.

Literature Review

Many studies have been done regarding this situation and many methods have been applied to solve the problem. Before deep learning models became widespread, one of the methods used in studies on this subject was maximum entropy (Zitouni and

Sarikaya, 2009). Maximum entropy models provide a probabilistic framework for estimating the probability distribution over the possible diacritics for each word in the input text.

In another research(Adali and Eryiğit, 2014) conducted specifically for Turkish, the best sequences are produced at the character level by using CRF. Afterwards, the one with the highest probability is selected with Language Validator and produced as output.

By using deep learning models, significant improvements have been achieved in terms of both performance and the ability to apply the same model to more than one language(Belinkov and Glass, 2015). It is seen that deep learning models, especially LSTM models, are widely used in RNN structures (Madhfar and Qamar, 2021). Additionally, in this study, studies conducted with encoder-decoder and decoder-only models reached a higher success rate. Especially thanks to the attention model used, not only that word but also the semantic expressions of the surrounding words gained importance and had a positive impact on the result.

Model

The model phase consists of 3 stages. The first is pre-processing, then the establishment and training of the model, and the last is testing the model.

Preprocessing

The quality of the data is important for the model to show adequate performance. For this reason, several stages were applied to prepare the data to be used in training the model.

First, the number of words in the train.csv file was calculated. Then, the number of characters in these words was calculated.

```
Number of occurrences of each unique character:
 : 771171
a: 493191
e: 389428
i: 364900
n: 316712
r: 289752
l: 281941
ı: 202301
k: 194806
```

Figure 2: Most mentioned characters

During the training of the model, only the most commonly used characters, including those from the Turkish alphabet and frequently used foreign letters, were selected. Following the identification of these characters, words that did not contain any of them were excluded from the dataset. Additionally, setting a specific word length was crucial for optimizing model performance; thus, a maximum length of 20 characters was established. Analysis revealed that the majority of the words in the dataset fall below this character limit.

As a result of these operations, approximately 86 thousand words remained in the data set. In order to increase this number, Turkish words data obtained from the internet was added. This data set contains approximately 117 thousand words. As a result of the combination, a data set of approximately 203 thousand words was created and a sufficient resource was provided for training.

After a sufficient number of words were provided, the preparation of the data set for training began. At this stage, characters that are likely to be diacritized in words are translated into incorrect use. For example, since there is no letter ı on the English keyboard, the word "sınıf" can be written as "sinif". Here, the letters ı in the word class in the data set have been changed to i. This process was applied to other letters (ç,ğ,ö,ş,ü) that could be diacritized.

```
['kazandıktan',
 'kullanılırsa',
 'yönlerinden',
 'saptık',
 'fahr',
 'sapmadın',
 'kıymetleştirme',
 'ödeyip',
 'müdahalesi',
 'suyunda']
```

```
['kazandiktan',
 'kullanilirsa',
 'yonlerinden',
 'saptik',
 'fahr',
 'sapmadin',
 'kiymetlestirme',
 'odeyip',
 'mudahalesi',
 'suyunda']
```

Figure 3: Example words for this process

After the character change process was completed, approximately 70% of this data set and 30% of the first data set were combined to create the main data set. The purpose of using the first data set is to prevent the model from overfitting, that is, changing all special characters.

Once the dataset was fully prepared for use, the next step involved converting the characters into indices. During this process, each valid character was assigned a unique index value, while invalid characters were assigned a constant value (one greater than the total number of valid characters). Following the conversion of characters into numerical

values, these numeric lists were then transformed into tensors. To standardize these tensor values, padding was applied, ensuring uniformity in tensor dimensions across the dataset.

Establishment and Training

The model was constructed using an RNN architecture that incorporates three consecutive bidirectional LSTM layers. These LSTMs sequentially process the input data, capturing dependencies in both forward and backward directions of the input sequence, enhancing the model's ability to learn from context. After the LSTM layers, a linear layer is utilized to map the output to the desired number of character classes. For the initial input processing, PyTorch's built-in 'nn.Embedding' function is employed to convert character indices into dense vector representations, facilitating more effective learning by the LSTM layers.

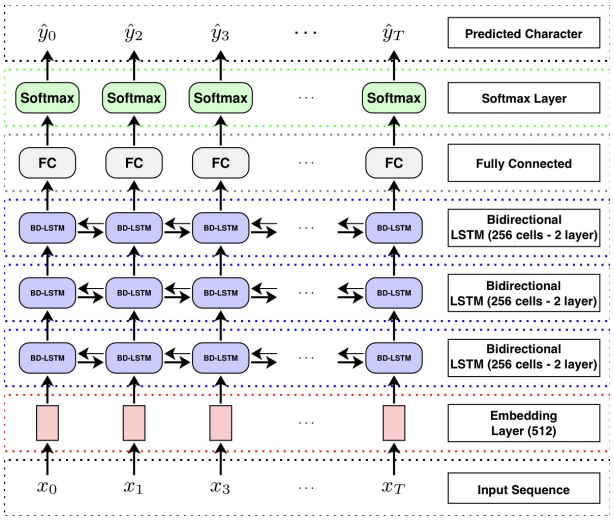

Figure 4: Model Structure

The hyperparameters for the model include a vocabulary size set to the one more than the number of characters (valid characters plus the value assigned to invalid characters), an embedding dimension of 512, a hidden size of 256. The model predicts a number of classes equal to the vocabulary size. The data is managed using a custom dataset class MyDataset, loaded into a DataLoader with a batch size of 64 and shuffling enabled. The model uses the cross-entropy loss function and the AdamW optimizer with a learning rate of 0.001 and a weight decay of 0.01.

Training occurred on a single NVIDIA GTX 1660Ti GPU with 6 GB of VRAM, completing in approximately 40 minutes.

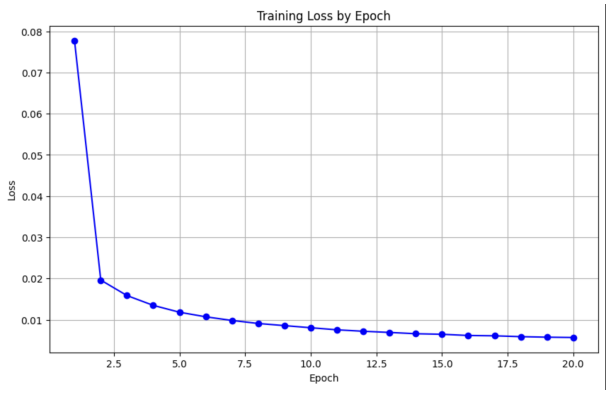

Figure 5: Training Loss Graph

Testing the Model

After the training phase is completed, the words in the data set to be tested in the testing phase are passed through the model one by one. In this process, sentences are first taken from the data set. These sentences are divided into words with the split method. If all the characters in the word are not suitable, these words are not passed through the model. For example, numerical expressions, punctuation marks. The characters in valid words are converted to index values and inserted into the model. The model produces new indexes in response to these indexes. These indexes are converted back into characters and given as output. During this process, attention is paid to whether the first letter starts with a capital letter and the first letter of the output is determined accordingly. This condition is applied to all rows and saved in the data set.

```
# Example usage
input_sentence = "Istanbul zavalli adami oracikta astilar "
output_sentence = predict(model, input_sentence, char_to_index, index_to_char)
print("Predicted sentence:", output_sentence)

Predicted sentence: İstanbul zavallı adamı oracıkta astılar
```

Figure 6: Test Example

Analysis and Discussion

As a result of applying the model on the entire test set, a success rate of approximately 0.92 was achieved. Although it is a good result using only RNN, performance can be improved by adding to the model or using higher performance models.

Integrating an attention mechanism could be particularly beneficial. Attention allows the model to focus on specific parts of the input sequence that are more relevant to the current output, effectively

addressing situations where the context or meaning of words may vary significantly depending on other words in the sentence. This capability could lead to a more nuanced understanding and generation of sequences, thus potentially improving performance.

Moreover, adopting an encoder-decoder architecture could further boost the success rate. This architecture, which has shown promising results in various NLP tasks across different languages, separates the encoding of input data from the generation of output, enabling the model to handle complex patterns and long-range dependencies more effectively.

Additionally, experiments can be conducted with more advanced models such as Transformer-based architectures that include attention mechanisms. Studies have shown that these models perform well for other languages, even for the Turkish language(Náplava et al., 2021).

References

Kübra Adali and Gülşen Eryiğit. 2014. Vowel and diacritic restoration for social media texts. pages 53–61.

Yonatan Belinkov and James Glass. 2015. Arabic diacritization with recurrent neural networks. pages 2281–2285.

Mokthar Ali Hasan Madhfar and Ali Mustafa Qamar. 2021. Effective deep learning models for automatic diacritization of arabic text. *IEEE Access*, 9:273–288.

Jakub Náplava, Milan Straka, and Jana Straková. 2021. Diacritics Restoration using BERT with Analysis on Czech language. *The Prague Bulletin of Mathematical Linguistics*, 116:27–42.

Imed Zitouni and Ruhi Sarikaya. 2009. Arabic diacritic restoration approach based on maximum entropy models. *Computer Speech and Language*, 23:257–276.