Can Perplexity Predict Fine-Tuning Performance? An Investigation of Tokenization Effects on Sequential Language Models for Nepali

Nishant Luitel, Nirajan Bekoju, Anand Kumar Sah and Subarna Shakya
Dept. of Electronics and Computer Engineering,
Pulchowk Campus, Tribhuwan University,
Lalitpur, Nepal
{076bct041.nishant, 076bct039.nirajan, anand.sah}@pcampus.edu.np, drss@ioe.edu.np

Abstract

Recent language models use subwording mechanisms to handle Out-of-Vocabulary(OOV) words seen during test time and, their generation capacity is generally measured using perplexity, an intrinsic metric. It is known that increasing the subword granularity results in a decrease of perplexity value. However, the study of how subwording affects the understanding capacity of language models has been very few and only limited to a handful of languages. To reduce this gap we used 6 different tokenization schemes to pretrain relatively small language models in Nepali and used the representations learned to finetune on several downstream tasks. Although byte-level BPE algorithm has been used in recent models like GPT, RoBERTa we show that on average they are sub-optimal in comparison to algorithms such as SentencePiece in finetuning performances for Nepali. Additionally, similar recent studies have focused on the Bert-based language model. We, however, pretrain and finetune sequential transformer-based language models.

1 Introduction

Nepali language, written in Devanagari script, is an Indo-Aryan language and the official language of Nepal. As per the NPHC 2021 report in Nepal, 44.9% (about 13 million) speak the Nepali language as their mother tongue, while 46.2%(nearly 13.5 million) speak it as their second language. Nepali is also spoken in neighboring parts of India, Bhutan, Brunei, and Myanmar. The structure of Nepali sentences differs from that of many other languages. The dominant sentence structure in the Nepali language is subject-object-verb. Due to the lack of quality data resources and computational resources, there has not been a significant advancement in the NLP domain of the Nepali language. A rich set of vocabulary and morphology has made it difficult to write clear, concise, and correct Nepali content. Hence, investigating the applicability of various recent NLP technologies in Nepali can be beneficial to both researchers and Nepali Speakers. Moreover, the findings on the Nepali language can also be generalized to several other languages with Devanagari script like Hindi, Sanskrit, Maithili, Bhojpuri, etc.

Tokenization is a process in NLP that involves breaking down textual data into smaller units, typically words or subwords, referred to as tokens. This fundamental technique serves as a cornerstone in various NLP tasks, including machine translation, sentiment analysis, and named entity recognition. By segmenting text into tokens, the complexity of language is reduced to manageable units, facilitating computational analysis and understanding. Moreover, tokenization plays a crucial role in preprocessing raw text data for machine learning algorithms, enabling efficient feature extraction and model training. In essence, tokenization is the first step in unlocking the power of language data for analysis and understanding.

Most language models today generate human-like text by leveraging deep learning models and huge datasets. Researchers have developed a wide range of transformer-based language models for text generation or representation learning. These language models are pre-trained using either a Masked language model approach where the model learns to predict the masked tokens based on the surrounding context like BERT (Devlin et al., 2018) or an auto-regressive language model approach where the model is trained to predict the next word given the previous context words like GPT-X (Radford et al., 2019; Brown et al., 2020) or PaLM (Chowdhery et al., 2022). The strength of the auto-regressive language models is that they only need previous time contextual information to predict the next token which is essential when gen-

erating sequences. The strength of the masked language is that it takes into account the surrounding context in both directions to learn powerful representations that can used for finetuning. In this paper, instead of using a BERT-based Nepali language model, we use sequential language models trained with various tokenization methods to finetune on downstream tasks. We have used the term sequential LM to mean an autoregressive LM which we use interchangeably in this paper.

The major contributions of our paper are as follows:

1. We pretrained 7 different sequential language models with following tokenizers: Word Tokenizer (30,000 and 60,507 vocabs), SentencePiece tokenizer, WordPiece tokenizer, BPE tokenizer, Morpheme tokenizer, combination of Morpheme and BPE tokenizers (all with 30,000 vocabs).

2. Comparison of performance language models based on perplexity was done during pretraining with different tokenization methods.

3. Comparison of performance of the pretrained language models was done based on fine-tuning over several Nepali Natural Language Understanding(NLU) tasks. We have made all our code and models open via github.

2 Related Works

The goal of the language model is to predict the next word given the context words.(Bengio et al., 2000) presented the Neural Probabilistic Language Model(NPLM) that can learn distributed representation for each word along with the probability function for the word sequences. Before the introduction of RNNs, various approaches based on parse trees, and n-gram statistics were used. (Mikolov et al., 2010) introduced an RNN-based language model (RNN LM) with application to speech recognition which proved the superiority of connectionist language models to the standard n-gram techniques, except for their high computational complexity. (Sutskever et al., 2011) showed the development of character-level modeling for text generation by training the RNNs with the Hessian-Free optimizer. The introduction of Transformer in (Vaswani et al., 2017) revolutionalized the field of natural language processing. (Vaswani et al., 2017) implemented the attention mechanism to develop the SOTA machine translation model, i.e. to generate the text in one language, given the context in another language. The parallelization ability in the transformer model solves the issue of the high computational and training complexity of previous sequential models. Various transformer-based models such as BERT (Devlin et al., 2018) and GPT(Brown et al., 2020) were developed which are the basis of many NLP tasks present today.

In recent years there have been some research towards pretraining and finetuning of NLP tasks in low-resource language like Nepali. (Maskey, 2023) pre-trained text-generation model with the same configuration as the (Sanh et al., 2019) on a combined dataset: Oscar, cc100, and a set of scraped Nepali Articles on Wikipedia by using the SentencePiece Model as a tokenizer, with vocabulary size 24, 576. (Maskey et al., 2022) trained three distinct transformer-based masked language models (distilbert-base, deberta-base, and XLM-ROBERTa) for Nepali text sequences which were evaluated and compared with other Transformer-based models on a downstream classification task. Similarly, (Niraula and Chapagain, 2022) fine-tuned Multilingual Bert for NER task. (Timilsina et al., 2022) trained another Bert-based language model for Nepali using the WordPiece vocabulary of 30,522 sub-word tokens and showed that their model performed better than other Bert-based LMs (Rajan, 2021; Devlin et al., 2018; Conneau et al., 2020) when fine-tuned on four distinct tasks namely, Content Classification, Named Entity Recognition, Part of Speech Tagging, and Categorical Pair Similarity. Although, several pretraining and finetuning have been performed in Nepali, a comparative study on the performance of language model on downstream tasks due to various tokenization haven't been performed.

However, there have been some similar research in other languages. (Toraman et al., 2022) analyzed the efficiency(in terms of training time, carbon emissions) and effectiveness(in terms of performance) of various tokenization techniques by finetuning a Turkish Bert-based LM on various downstream NLP tasks. They found that for similar and smaller vocab sizes BPE(char-level) and WordPiece are better than other tokenization techniques like word-based. (Park et al., 2020) found that for Korean, morpheme tokenization fol-

lowed by BPE(char-level) achieved best performance. This type of tokenization schemes forces BPE to not consider the sequence of bytes that span across multiple morphemes. Similar result was obtained by (Alrefaie et al., 2024) for Arabic where BPE combined with Morphemes was optimal. Finally, (Alyafeai et al., 2021) also evaluated performance of different tokenization methods on three Arabic NLP classifications task but didn't use transformer based architecture. Our method is different from these studies because we finetune on sequential language model rather than Bert-based language models. Secondly, we also analyze the performance of byte-level BPE tokenization algorithm which hasn't been performed in these studies. Finally, we try to reason with emperical evidence that the intrinsic metric that is often used during pretraining of a language model i.e. perplexity has no prediction ability about the finetuning performances.

3 Methodology

3.1 Tokenization Techniques

We have trained 6 different tokenizers keeping the vocabulary size at the constant of 30000. We intend to perform a comparison of LMs(perplexity and finetuning performance) but the perplexity scores tend to decrease with decreasing vocabulary size. Hence comparison through constant vocab size across models makes more sense. The table 1 shows encoded text for the same input by every tokenizer. Below are the specifics of how we trained these tokenizers.

1. **Word-based**: In the word-based tokenization scheme we used top 30k vocab based on frequency. We include a <unk>token to represent all the OOV words during training and evaluation. Further, we also include <num>token in the vocab to encode all the number strings in Nepali. We used torchtext from PyTorch to build the vocabulary.

2. **Morphemes**: Morphemes are the smallest subdivisions of a word that contain meaning. We used the morfessor 2.0 library to train a model that learns to break a compound word into morphemes using MAP estimate (Smit et al., 2014). We used this morfessor model to prepare a morpheme-level training corpus by using around a third of the OSCAR corpus. As suggested in (Park et al., 2020), we add a '*' token to represent the presence of space between words in the corpus so that during decoding we know when to add space and when to combine the morphemes to make compound words. Following the mentioned scheme, the text 'AB C' would be segmented as 'A B * C'.

3. **WordPiece**: The concept behind WordPiece is to divide a given word into multiple subwords that are frequently used to form a compound word. The algorithm works by first dividing a word into characters with the addition of '##' in the beginning to non-starting tokens. For example 'जीवन' would be initially divided as '(ज, ##ी, ##व, ##न)'. Then the word pieces are combined in order given by score in equation 1 where 'f' means frequency.

$$score = \frac{f_{pair}}{f_{1st} * f_{2nd}} \quad (1)$$

This score prioritizes the frequent combination of infrequent subtokens. The encoding in WordPiece works by finding the largest subtoken that is present in the vocabulary made during training. We used the Bert WordPiece algorithm to train this tokenizer using the 'Tokenizers' Python package. This didn't support some of the modifier tokens(diacritics) in the characters hence we replaced all these tokens in the corpus with English letters during the preprocessing phase. The reverse of this preprocessing was done during decoding.

4. **SentencePiece(with BPE)**: In this tokenizer, we used character level BPE compatible with SentencePiece. Unlike WordPiece, the BPE algorithm sequentially merges the characters/subtokens based on the frequency of the merged token directly, and during encoding, the rules that are learned in training are applied sequentially (Sennrich et al., 2016). Our approach is similar but with the feature of white space handling as introduced in (Kudo and Richardson, 2018). We used the 'Tokenizers' Python package to train this tokenizer.