# Contextual Bangla Neural Stemmer: Finding Contextualized Root Word Representations for Bangla Words

**Md Fahim, Amin Ahsan Ali, M Ashraful Amin, A K M Mahbubur Rahman**
Center for Computational & Data Sciences
Independent University, Bangladesh
Dhaka-1229, Bangladesh
fahimcse381@gmail.com, {aminali,aminmdashraful,akmmrahman}@iub.edu.bd

## Abstract

Stemmers are commonly used in NLP to reduce words to their root form. However, this process may discard important information and yield incorrect root forms, affecting the accuracy of NLP tasks. To address these limitations, we propose a Contextual Bangla Neural Stemmer for Bangla language to enhance word representations. Our method involves splitting words into characters within the Neural Stemming Block, obtaining vector representations for both stem words and unknown vocabulary words. A loss function aligns these representations with Word2Vec representations, followed by contextual word representations from a Universal Transformer encoder. Mean Pooling generates sentence-level representations that are aligned with BanglaBERT's representations using a MLP layer. The proposed model also tries to build good representations for out-of-vocabulary (OOV) words. Experiments with our model on five Bangla datasets shows around 5% average improvement over the vanilla approach. Notably, our method avoids BERT retraining, focusing on root word detection and addressing OOV and sub-word issues. By incorporating our approach into a large corpus-based Language Model, we expect further improvements in aspects like explainability.

## 1 Introduction

Large Language Models (LLMs) like BERT (Devlin et al., 2019), GPT (Brown et al., 2020), and others have proven their efficacy in various Natural Language Processing (NLP) tasks. They excel at capturing contextual information and cultural subtleties in specific languages. These models exhibit strong capabilities for addressing diverse NLP tasks, especially during their unsupervised pretraining phase. However, in low resource language like Bangla, there are so many language specific problems that haven't been resolved yet since Bangla language lack comprehensive lexicons, word embeddings, or linguistic resources. **F**irstly, there may be a good number of out-of-vocabulary (OOV) words which may hamper the NLP tasks. **S**econdly, in LLMs, tokenizing one word can result splitting into different subwords that make the model difficult to explain. In the following paragraphs, we clarify these problems with examples.

| **Method** | **Tokens** |
|---|---|
| Original Text | সে বাড়িতে যাওয়ার পর আর যোগাযোগ করেনি |
| BanglaBERT Tokenizer | ['সে', '[UNK]', '[UNK]', 'পর' 'আর', 'যোগাযোগ', 'করেনি'] |
| Bangla Stemmer | [সে, 'বাড়ি', 'যাওয়', 'পর' 'আর', 'যোগাযোগ', 'করেনি'] |

Table 1: The Limitations of Bangla BERT which gives [UNK] tokens for many informative words of a sentence and Bangla Stemmer sometimes produce a word with no meaning and also losses the context information.

In Table 1, we show an example of Bangla sentence and outputs of the tokenizer of the Bangla BERT (Bhattacharjee et al., 2022): a BERT model trained on the Bangla Corpus to demonstrate the first kind of problems. We can easily see that the occurrence of OOV tokens represented as "[UNK]" is very frequent. This significantly impacts the model's ability to comprehend semantic and linguistic information in the sentence. One possible solution to solve the OOV problem is to find root words.

**S**econd set of problems are observed due to the use of bangla stemmer/lemmitizer. There are many existing way for finding the root words like stemming and lemmitizer. Lemmitizer needs ground truth word mapping to find the word. On the other hand, stemming algorithms typically use heuristics to identify the suffixes of words that can be removed to obtain the root form. However, by re-



*Proceedings of the First Workshop on Bangla Language Processing (BLP-2023)*, pages 94–103

| Method | Tokens |
|---|---|
| Original Text | নটরডেম, হলিক্রস ও ভিকারুননিসা কলেজে ভর্তি হতে পারবে না ধূমপায়ী শিক্ষার্থীরা। |
| BanglaBERT Tokenizer | ['নট', '##র', '##ডেম', ',', 'হলি', '##ক্র', '##স', 'ও', 'ভিক', '##ার', '##ুন', '##নি', '##সা','কলেজে' 'ভর্তি', 'হতে', 'পারবে', 'না', '[UNK]', 'শিক্ষার্থীরা' , '।'] |

Table 2: Subwords Problem in Bangla BERT.

ducing words to their root form, a stemmer discards important information that could be useful in natural language processing tasks. In cases of bangla, a stemmer may reduce a word to an incorrect root form, leading to incorrect results. For example in Table 1, for a given bangla sentence, the bangla stemmer creates some incorrect roots that have no vector representations at all. Moreover, LLMs like Bangla-BERT also faces OOV problem very recurrently because Bangla-Bert tokenizer splits a word into one or more subwords. It has been shown that splitting words into multiple subwords is not the best option all the time. There are some cases where this might not work well (Nayak et al., 2020), (Toraman et al., 2023). Some words might have a prefix or suffix that changes the meaning of the word, but BERT's subword tokenizer might split it into separate subwords. For example, in Table 2, Bangla BERT tokenizer splits words into multiple subwords, leading to a significant increase in the number of subwords. This excessive subword splitting makes it challenging to extract the actual meaning of individual words in the sentence, thereby affecting the overall interpret-ability and comprehensibility of the model.

Considering the aforementioned limitations of Rule Based Stemmer, we want to create a Contextual Bangla Neural Stemmer for Bangla language to find better representation of words. Especifically, in our proposed method, by splitting each word of a sentence into characters in Neural Stemming Block, we will get vector representation for not only the stem word but also the unknown vocab word. A loss is used to make sure that the representations of the words should be aligned with the Word2Vec representations. Then after a liner layer transformations those representations is passed into Universal Transformer (UT) (Dehghani et al., 2018) encoder to assure of getting contextual representation of a word via self attention. Mean Pooling is used to get a sentence level representation for a sentence from those contextual word representations. A MLP layer is used and a loss is defined to align the sentence representation with the BanglaBERT's one. The whole model pipeline is described in Section 3.

Our model employs character-based representations to find root word representations, which effectively addresses the issues of OOV tokens and subword tokenization. By combining these representations with BERT, our model is capable of obtaining contextual representations for these root words, enhancing its ability to capture the semantic nuances and context of the language. We evaluate our model performance in 5 different Bangla dataset. In every dataset, our model outperforms the vanilla approach by a good margin (around 5% improvement on average). More details are described in Section 5. Please note that our goal is not retraining the BERT at all. Instead of retraining the BERT, our proposed method is used finding the root words with contextual representations and address OOV & sub-word problem. If we create a LLM based on our methodology with a large corpus, our method may outperform the Vanilla BERT in different aspects and may improve the explainability also. Therefore, the summary of the contributions of this paper is given below.

- We propose a neural network based stemmer that can be contextualized
- We propose new losses to learn root word representations with contextual information
- We design a number of experiments to show the efficacy of the proposed appoach

## 2 Related Work

Finding root words for Bangla word is one of the most popular tasks in Bangla Natural Language Processing (NLP). Several works have been done already. We can categorize those works in two different perspective, one is morphological method base and another is heuristic base. In morphological method based approach for root word finding Lemmitizer and Stemming are used. In heuristic base, rule base or model base approaches are used. In (Mahmud et al., 2014), a rule based stemming technique is used for finding the root word in Bangla. They use different set of rules so that they can find the stem word by cutting down the prefixes. (Das et al., 2020) enhanced

the rule based stemming techniques by improving the rules for different categories. They also incorporate Bangla Corpus and different inflections for noun, verb, and other parts of speech. (Rahit et al., 2018) introduces BanglaNet which is an approach to make a WordNet for Bangla Language. (Chakrabarty et al., 2016) uses a single layer Multi Layer Perceptron MLP for finding the lemmitize word of a word along with its contextual neighbours. (Chakrabarty and Garain, 2016) uses a distance based algorithm to find the lemma of a word with respect to a given context and part of speech of the word. (Chakrabarty et al., 2017) and (Islam et al., 2022) propose algorithm to find lemma word based on the contextutal representations. The contextual representations are derived from Bi-LSTM or Bi-GRU. No works have been proposed to find the stemmed word representations from the contextual information.

## 3 Methodology

The tokenizers that are used in Transformer based model like BPE, Wordpiece, Unigram split a word into multiple subwords. This may cause information discrepancy between the actual meaning of the actual word which may affect the low resource language models like Bangla language model. Besides, there also may have a good amount of out of vocabulary words in those low resource language model. One option is to find the root form of the words but this approach miss the contexutal information. Considering all scenarios, we propose a character based contextual neural stemmer which not only find the stemmed root word to surpass subword techniques but also give the contextual embeddings. For being character based model, our proposed model can also deal with the out of vocabulary issues. Our model have two major components, Character Level Neural Stemming Block, Universal Transformer Encoding Block along with two different losses for fulfilling our criteria.

### 3.1 Character Based Neural Stemmer

After passing a sentence into the Basic Tokenizer, we get the tokens of the sentence. Let $S = [x_1, x_2, \ldots, x_m]$ represent a sentence, where each word token $x_i$ is split into characters.

For each word token $x_i$, we denote the character embeddings as $C_i = [c_{1i}, c_{2i}, c_{3i}, \ldots]$, where $c_{ij}$ represents the embedding of the $j$th character.

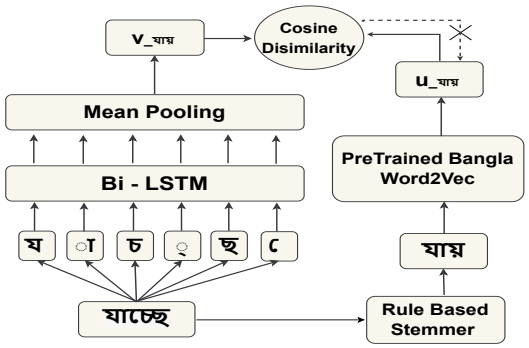


Figure 1: Model Architecture of Neural Stemmer Block

We pass $C_i$ through an LSTM layer, which produces a sequence of hidden representations $H_i = [h_{1i}, h_{2i}, h_{3i}, \ldots]$. Each $h_{ij}$ represents the hidden state at time step $j$.

The LSTM layer takes the character embeddings as input and generates hidden states using the following equations:

$$h_{ij} = \text{LSTM}(c_{ij}, h_{i(j-1)}) \tag{1}$$

Once we have obtained the sequence of hidden states $H_i$, we compute the mean of these hidden states to obtain the word embedding $v_i$:

$$v_i = \frac{1}{m} \sum_{j=1}^{m} h_{ij} \tag{2}$$

Here, $m$ represents the total number of characters in the word token $t_i$. The mean aggregation operation captures the overall representation of the word by considering the contextual information contained in the LSTM hidden states.

This process allows us to derive word embeddings $v_i$ from character embeddings, enabling us to capture fine-grained information and enhance the representation of word tokens within the given sentence. In this block, we also apply a stemming loss with the pre-trained Word2Vec representations of the stemming words. The stemming loss is described in Section 3.4.

### 3.2 Universal Transformer Encoder

After obtaining the neural stemming output $V = [v_1, v_2, \ldots, v_m]$ for a sentence from the Neural Stemming Block described in Section 3.1, we perform a linear transformation on each $v_i$ to map them into a $d$-dimensional vector space. Now those transformed representations $V' = [v'_1, v'_2, \ldots, v'_m]$, is fed into the Universal Transformer (UT) encoder which consists of several UT

encoder blocks for finding contextual representations. We choose UT because it is a tied weight model which also uses Adaptive Computational Time (ACT). So we need less computational time for training and fine-tuning the Universal Transformer model than the vanilla Transformer model. The components of the UT encoder blocks are:

#### 3.2.1 Positional Encoding and Time Signal

To incorporate positional and temporal information, the combined positional encoding and time signal embeddings $P_t \in \mathbb{R}^{m \times d}$, are applied where $m$ represents the total number of positions and $d$ is the dimensionality of the embeddings. The combined embeddings are obtained by computing the sinusoidal position and time embeddings separately for each vector dimension $1 \leq j \leq d$ and summing them:

$$P_t[i, 2j] = \sin\left(\frac{i}{10000^{(2j/d)}}\right) + \sin\left(\frac{t}{10000^{(2j/d)}}\right) \quad (3)$$

$$P_t[i, 2j+1] = \cos\left(\frac{i}{10000^{(2j/d)}}\right) + \cos\left(\frac{t}{10000^{(2j/d)}}\right) \quad (4)$$

where $i$ represents the position index ($1 \leq i \leq m$), $t$ represents the time-step index ($1 \leq t \leq T$), and $j$ represents the vector dimension index ($1 \leq j \leq d$).

#### 3.2.2 Attention Mechanism

At each step $t$, the UT computes revised representations $H_t \in \mathbb{R}^{m \times d}$ for all $m$ input positions. This is done by applying the scaled dot-product attention mechanism, which combines queries $Q$, keys $K$, and values $V$ as follows:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V \quad (5)$$

Here, $d$ is the number of columns of $Q$, $K$, and $V$. In the Universal Transformer, a multi-head version of the attention mechanism is used, with $k$ heads:

$$\text{MultiHeadSelfAttention}(H_t) = \text{concat}(\text{head}_1, \ldots, \text{head}_k) \times W_o \quad (6)$$

Each head $\text{head}_i$ is calculated as $\text{Attention}(H_t W_{Qi}, H_t W_{Ki}, H_t W_{Vi})$, where $W_Q \in \mathbb{R}^{d \times d/k}$, $W_K \in \mathbb{R}^{d \times d/k}$, and $W_V \in \mathbb{R}^{d \times d/k}$ are learned parameter matrices. The output of the multi-head attention is then transformed using the weight matrix $W_o \in \mathbb{R}^{d \times d}$.

#### 3.2.3 Encoder Block Representation

After applying the multi-head self-attention, the UT computes the revised representations $H_t$ by combining the attention output $A_t$ with the previous representation $H_{t-1}$ and the positional encoding and time signal embeddings $P_t$:

$$A_t = \text{LayerNorm}((H_{t-1} + P_t) + \text{MultiHeadSelfAttention}(H_{t-1} + P_t)) \quad (7)$$

Here, LayerNorm() represents the layer normalization function. Finally, the revised representations are obtained by applying a transition function:

$$H_t = \text{LayerNorm}(A_t + \text{Transition}(A_t)) \quad (8)$$

The transition function Transition() applies non-linear transformations to the attention output $A_t$ and integrates it with the previous representation. The resulting revised representations $H_t$ capture the refined information at step $t$.

The UT encoder utilizes an iterative computation process, repeating for a total of $T$ steps. This iterative process progressively refines the representations of the input sequence, capturing intricate dependencies. To determine the number of steps, the Universal Transformer employs the Adaptive Computation Time (ACT) mechanism. After undergoing $T$ steps, where each step updates all positions of the input sequence simultaneously, the final output of the Universal Transformer encoder is a matrix $H^T \in \mathbb{R}^{m \times d}$. This matrix consists of $d$-dimensional vector representations for the $m$ tokens present in the input sequence.

By considering the hidden representation obtained after $T$ iterations, we obtain the contextual

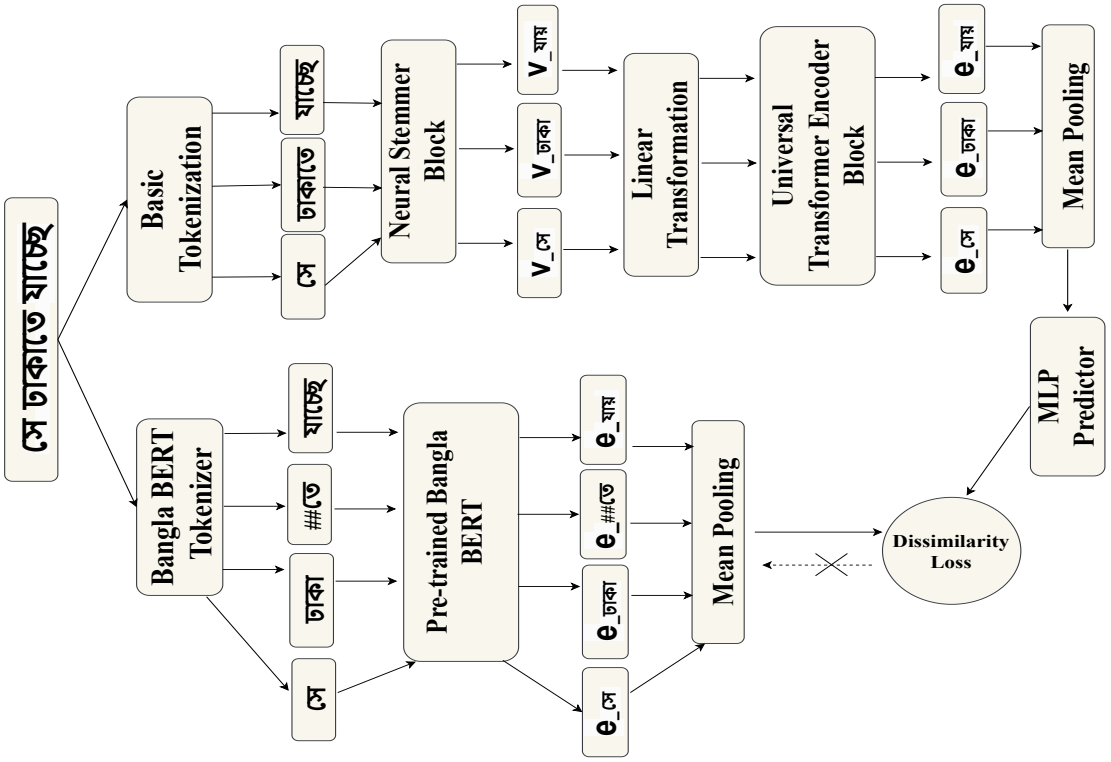


Figure 2: Model Architecture of Contextual Neural Stemmer Block

embeddings of the word tokens. Denoting the embeddings as $E = [e_1, e_2, \ldots, e_m]$, we can equivalently express $E$ as $E = H^T$. Therefore, the matrix $H^T$ represents the contextual embeddings for the sentence $S$.

### 3.3 Mean Pooling

Let $E = [e_1, e_2, \ldots, e_m]$ be the input sequence $S$ with embedding $e_i \in \mathbb{R}^d$ that we get from the universal transformer encoder block. The sequence may contain padded values for equal length. Define the mask vector $M = [\text{mask}_1, \text{mask}_2, \ldots, \text{mask}_m]$ to indicate valid tokens. $\text{mask}_i = 1$ for valid tokens and 0 for padded values. The masked mean pooling operation is:

$$\text{MeanPooling}(X, M) = \frac{1}{\sum_{i=1}^{n} \text{mask}_i} \sum_{i=1}^{n} \text{mask}_i \cdot e_i \quad (9)$$

After applying mean pooling to the sentence $S$, the sentence-level representation $e_S$ from the UT encoder are obtained. An MLP (Multi Layer Percptron) is applied to the $e_S$ to get the final sentence level representation $e_S$.

### 3.4 Stemming Loss

We utilize a loss function called *Stemming Loss* in the Neural Stemmer Block, as described in Section 3.1. The main objective of this loss is that the character based representations for the word tokens should be similar with their word2vec representation of their stemmed words. Given a sentence $S = [x_1, x_2, \ldots, x_m]$, we feed it into the neural stemmer block, which generates character-based representations $v_i$ for each word token $x_i$ in the sentence $S$. Additionally, each token in sentence $S$ is passed through a rule-based stemmer to obtain the root form, resulting in the stemmed version $S = [r_1, r_2, \ldots, r_m]$ of the sentence.

Subsequently, we input each root word $r_i$ into a pre-trained word2vec model, which produces a static embedding $u_i$ for the word $r_i$. We didn't train the pre-trained word2vec model during training. To align the predicted embedding $v_i$ with the static embedding $u_i$, we employ *Cosine Similarity* based loss as follows:

$$\textit{Stemming\_Loss}(u_i, v_i) = 1 - \frac{u_i \cdot v_i}{\|u_i\| \|v_i\|} \quad (10)$$

This loss ensures that the representation from the Neural Stemmer Block should be aligned with the stemming representation from pre-trained

Word2vec. If the word2vec representations of a stemmed words is not found, we simply ignore the word while calculating the stemming loss.

### 3.5 Dissimilarity Loss

During training our model, we also employ another loss named *Cosine Dissimilarity Loss*. The objective to use the loss is that the contextual embeddings for a sentence from the UT encoder block should be aligned with the pre-trained BERT contextual embeddings for that sentence. To calculate the loss, we also feed our input text into the pre-trained BERT and we get contextual embeddings for that sentences S, $E' = [e'_1, e'_2, \ldots, e'_m]$. We apply Mean Pooling described in Section 3.3 to get the sentence level representation $e'_S$ from the pre-trained BERT for sentence $S$. The pre-trained BERT model is not trained during the training process.

On the other hand, we also get another sentence level representation $e_S$ from UT encoder as described in Section 3.3. Then, we apply the cosine dissimilarity based loss as follows:

$$\textit{Cosine\_Dissimilarity}(e_S, e'_S) = 1 - \frac{e_S \cdot e'_S}{\|e_S\|\|e'_S\|} \tag{11}$$

### 3.6 Model Loss

To obtain high-quality contextual representations from our model, we rely on the Stemming Loss (Section 3.4) and cosine dissimilarity (Section 3.5). The cosine dissimilarity is based on the pretrained BERT representations, which face challenges such as the subword problem and out-of-vocabulary (OOV) problem. To ensure effective training of our model, we adopt a guided training schema. In this schema, we prioritize training our model on samples where the BERT tokenizer yields a lower number of OOV and subword tokens. Additionally, we incorporate a penalty score based on BERT tokenization techniques when calculating the final loss. Hence, our final training loss is defined as:

$$\begin{aligned} \textit{Loss} &= \gamma \times \textit{Stemming\_Loss} \\ &+ \left(1 - \frac{a+b}{m}\right) \times \beta \times \textit{Dissimilarity\_Loss} \end{aligned} \tag{12}$$

Her, $a$ represent the number of subword tokens, $b$ denotes the count of unknown ([UNK]) tokens, and $m$ indicate the total number of tokens. The weights $\gamma$ and $\beta$ determine the contribution of the stemming loss and cosine dissimilarity loss, respectively, to the main loss.

## 4 Experimental Setup

### 4.1 Experimental Design

Our model follows a two-step training process for each experimental dataset. In the first step, we engage in unsupervised training to learn contextual representations between words. The primary objective of this unsupervised training is to transfer the knowledge from the pre-trained BERT model to our contextual neural stemmer. In the second step, we conduct supervised fine-tuning, where we further train our model in a supervised fashion, focusing solely on the classification loss. To prioritize the development of semantic/contextualized representations for stemming words only, rather than building a language model (LM), we opted not to train our model extensively on a large corpus during the unsupervised training phase.

To identify stemming words in Bangla, a rule-based stemmer is employed, utilizing the Bangla Stemmer library. In unsupervised training, we choose Bangla-NLP Toolkit for find finding representations of the stemmed words as Bangla Pre-trained Word2vec. BanglaBERT (Bhattacharjee et al., 2022) model is used as pre-trained BERT algorithm in our model. Bangla-Word2vec provides 100-dimensional vector representations for each word. Consequently, we set the character embedding size to 100. For contextualized embedding, we define an embedding dimension of 768 to align with the 768-dimensional word representations obtained from Bangla BERT. To convert the 100-dimensional vectors to 768 dimensions, we employ a linear transformation block comprising a single linear layer. If we don't have the word2vec representations of a stemmed word, we neglect the word representations while calculating *Stemming Loss*. We evaluated our model's performance in different evaluation metrics like accuracy, macro f1 score and roc-auc. The details can be found about at Appendix B.

### 4.2 Model Training Setup and Training Scheme

We choose AdamW (Loshchilov and Hutter, 2017) optimizer for our training where $\beta_1 = 0.9$ and $\beta_2 = 0.99$. Character embedding size is 100 dim

| Dataset | Experiment | Pretraining Perplexity | Performance Metrics Accuracy | Macro F1 | Weighted F1 |
|---|---|---|---|---|---|
| BanFake News | Rule Based Stemmer | - | 88.1 | 87.2 | 91.1 |
| | Neural Stemmer | - | 90.4 | 89.6 | 93.6 |
| | CNS | 125.51 | 93.1 | 92.2 | 94.8 |
| Sarcasm Detection | Rule Based Stemmer | - | 85.4 | 45.1 | 89.9 |
| | Neural Stemmer | - | 87.8 | 46.0 | 91.4 |
| | CNS | 117.47 | 90.3 | 48.7 | 95.6 |
| SentNoB | Rule Based Stemmer | - | 64.5 | 60.8 | 64.1 |
| | Neural Stemmer | - | 69.2 | 62.3 | 68.8 |
| | CNS | 134.97 | 73.3 | 68.3 | 72.2 |
| Emotion Detection | Rule Based Stemmer | - | 62.5 | 35.4 | 61.2 |
| | Neural Stemmer | - | 64.6 | 39.7 | 62.1 |
| | CNS | 87.38 | 68.4 | 40.26 | 64.41 |
| Sentiment Classification | Rule Based Stemmer | - | 48.5 | 31.7 | 32.4 |
| | Neural Stemmer | - | 50.1 | 32.2 | 32.8 |
| | CNS | 103.49 | 52.3 | 34.5 | 35.9 |

Table 3: Experimental result for CNS in 5 different dataset. In every dataset, 3 different experiments along with CNS are done, Rule Based Stemmer, Neural Stemmer, and CNS. In every dataset, our CNS method outperform rule based stemmer with a good margin. Here, *CNS* means the *Contextual Neural Stemmer*

and contextual word representations has 768 dim as described in Section 4.1. We use a learning rate of $2 * 10^{-5}$ for unsupervised training and $10^{-3}$ for supervised finetuning. A LSTM (Hochreiter and Schmidhuber, 1997) based decoder with dropout of 0.1 while finetuning the model. A batch size of 32 is used for unsupervised training and 16 for supervised finetuning. We also experiment with the different combinations of $\gamma$ and $\beta$ and found that $\gamma = 0.7$ and $\beta = 0.5$ gives better performance most of the cases. All the experiments run with Python (version 3.8) and Pytorch with free NVIDIA Tesla K80 GPU in Google Colab and single Nvidia Tesla P100 GPU provided by Kaggle. The training time for both unsupervised and supervised varies but on average it takes around 6 mins on average for training one epoch in unsupervised training and 4 mins in supervised training.

## 5 Result and Discussion

### 5.1 Effects in Different NLP Datasets

To measure the performance of our model, we consider five different Bangla dataset. The dataset tasks and information are listed in the Appendix A. In every dataset, we run three different experiment.

- **Rule Based Stemmer**: In this experiment, we use a rule based stemmer to find the stemmed word of a word in a sentence. We consider the stemmed words as the tokens of a sentence. Finding the embeddings of the tokens a single LSTM layer is used to find contextual representations. We consider last lstm cell output as sentence representation and passed it into MLP for classification.

- **Neural Stemmer**: Instead of rule based stemmed word, we use Neural Stemmer Block described in Section 3.1. After finding neural stemming representations, we passed them into a single LSTM layer and MLP layers for classifcation as same as Rule Based Stemmer.

- **CNS**: CNS stands for Contextual Neural Stemmer which is our proposed model as described in Figure 2. We use last MLP layer representations for classification. In every dataset, we first pretrained our model in un-

supervised fashion and then finetune it using classification loss.

Table 3 shows the experimental result in five different dataset. In every dataset Neural Stemmer slightly outperforms the rule based one. This is because, sometimes the rule-based stemmer results in the stemmed words which may have no meaning. In this case, character based word representations improves model performance. In every dataset, our method surpass both rule based and neural stemmer based approaches by a good margin (around 2 - 7% improvement on average in different metrics). The reason behind this is, the stemming word (either rule based or neural model based) losses the contextual and semantic information like tense, expression, grammatical context which are very useful for a model to find a good representations. CNS captures those information along with the stemmed word and that's why our model surpassed the other methods.

| Dataset Name | Average Cosine |
|---|---|
| BanFake News | 0.7014 |
| Sarcasm Detection | 0.7862 |
| SentNoB | 0.6776 |
| Emotion Detection | 0.7569 |
| Sentiment Classification | 0.7980 |

Table 4: Average Cosine Similarity from CNS Model in Test Sentences between Word2vec of Stemming word and Word Representations from Neural Stemmer Layer.

### 5.2 Preserving the Stemming Words in Neural Stemmer

We also further investigate on how much stemming information are captured by our model. To find this, we consider the test dataset in aforementioned datasets. We find the pretrained Word2vec presentations of the word in text sentences. We find average cosine similarity between those stemmed word's pretrained word2vec representations and the representations from Neural Stemmer layer. The results are reported in Table 4. From this table we can see our model is able to capture stemming information. By tuning $\gamma$ we can control how much stemming information should be captured by our model.

| Dataset Name | Average Cosine |
|---|---|
| BanFake News | 0.6572 |
| Sarcasm Detection | 0.7284 |
| SentNoB | 0.6397 |
| Emotion Detection | 0.7128 |
| Sentiment Classification | 0.7329 |

Table 5: Average Cosine Similarity from CNS Model in Test Sentences between Token Representations of Pre-Trained Bangla BERT and Contextual Word Representations from UT Encoder.

### 5.3 How Contextualized the Contextual Neural Stemmer

We were also interested in experimenting how much contextual information is capturing like BERT. For doing this we reported two experiment. For the first one, we average cosine similairties between the word representations of a sentence of pretrained Bangla-BERT in the test samples and the Universal Transformer (UT) encoder representations from CNS. The results are in Table 5. Another experiment is done on the sentence level representations. We consider mean of the word representations of pretrained model as sentence level representations and measure a cosine similarities with MLP representations from CNS in Tabale 6. From this experiment, we can see that our CNS model is also able to capture contextual information.

| Dataset Name | Average Cosine |
|---|---|
| BanFake News | 0.9563 |
| Sarcasm Detection | 0.9790 |
| SentNoB | 0.9227 |
| Emotion Detection | 0.9673 |
| Sentiment Classification | 0.9872 |

Table 6: Average Cosine Similarity of Sentences in Test Sentences between Mean Pooling Output from Pretrained Bangla BERT Representations and Last Layer MLP Representations from CNS.

## 6 Conclusion

In this research, we proposed a Contextual Bangla Neural Stemmer to overcome the limitations of traditional rule-based stemmers. By obtaining vector representations for both stem words and unknown vocabulary words, our method offers improved word representations for Bangla language processing tasks. The model leverages the Uni-

versal Transformer encoder and Mean Pooling to capture contextual word and sentence-level representations. Our evaluation on five Bangla datasets demonstrated significant performance gains, outperforming the vanilla approach. Notably, our approach focuses on root word detection and addressing OOV and sub-word problems rather than retraining the BERT.

Our findings suggest that a large corpus-based language model incorporating our methodology could further enhance NLP tasks and potentially improve explainability. By addressing the limitations of stemmers and providing better word representations, our proposed approach opens new avenues for research in Bangla language processing and contributes to advancing natural language understanding in the context of Bangla text.

## Limitations

As we mentioned above, the proposed method works well against the stemming method but it can't beat the finetuning BanglaBERT. (The performance of BanglaBERT is reported in Appendix C.) The reason behind this BanglaBERT is a language model which was trained on huge corpus. As our method isn't trained on the huge corpus so our model can't beat the BanglaBERT. If we trained our proposed model in a huge corpus, it may be possible to beat BanglaBERT.

## Acknowledgements

This project has been jointly sponsored by Independent University, Bangladesh and the ICT Division of the Bangladesh Government.

## References

Abhik Bhattacharjee, Tahmid Hasan, Wasi Ahmad, Kazi Samin Mubasshir, Md Saiful Islam, Anindya Iqbal, M. Sohel Rahman, and Rifat Shahriyar. 2022. BanglaBERT: Language model pretraining and benchmarks for low-resource language understanding evaluation in Bangla. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 1318–1327, Seattle, United States. Association for Computational Linguistics.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Abhisek Chakrabarty, Akshay Chaturvedi, and Utpal Garain. 2016. A neural lemmatizer for bengali. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 2558–2561.

Abhisek Chakrabarty and Utpal Garain. 2016. Benlem (a bengali lemmatizer) and its role in wsd. *ACM Transactions on Asian and Low-Resource Language Information Processing (TALLIP)*, 15(3):1–18.

Abhisek Chakrabarty, Onkar Arun Pandit, and Utpal Garain. 2017. Context sensitive lemmatization using two successive bidirectional gated recurrent networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1481–1491.

Souvick Das, Rajat Pandit, and Sudip Kumar Naskar. 2020. A rule based lightweight bengali stemmer. In *Proceedings of the 17th International Conference on Natural Language Processing (ICON)*, pages 400–408.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. 2018. Universal transformers. *arXiv preprint arXiv:1807.03819*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Md Zobaer Hossain, Md Ashraful Rahman, Md Saiful Islam, and Sudipta Kar. 2020. BanFakeNews: A dataset for detecting fake news in Bangla. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 2862–2871, Marseille, France. European Language Resources Association.

Khondoker Ittehadul Islam, Sudipta Kar, Md Saiful Islam, and Mohammad Ruhul Amin. 2021. SentNoB: A dataset for analysing sentiment on noisy Bangla texts. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 3265–3271, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Md Ashraful Islam, Md Towhiduzzaman, Md Tauhidul Islam Bhuiyan, Abdullah Al Maruf, and Jesan Ahammed Ovi. 2022. Banel: An encoder-decoder based bangla neural lemmatizer. *SN Applied Sciences*, 4(5):138.

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Md Redowan Mahmud, Mahbuba Afrin, Md Abdur Razzaque, Ellis Miller, and Joel Iwashige. 2014. A rule based bengali stemmer. In *2014 international conference on advances in computing, communications and informatics (ICACCI)*, pages 2750–2756. IEEE.

Anmol Nayak, Hariprasad Timmapathini, Karthikeyan Ponnalagu, and Vijendran Gopalan Venkoparao. 2020. Domain adaptation challenges of BERT in tokenization and sub-word representations of out-of-vocabulary words. In *Proceedings of the First Workshop on Insights from Negative Results in NLP*, pages 1–5, Online. Association for Computational Linguistics.

KM Tahsin Hassan Rahit, Khandaker Tabin Hasan, Md Al-Amin, and Zahiduddin Ahmed. 2018. Banglanet: Towards a wordnet for bengali language. In *Proceedings of the 9th Global Wordnet Conference*, pages 1–9.

Cagri Toraman, Eyup Halit Yilmaz, Furkan Şahinuç, and Oguzhan Ozcelik. 2023. Impact of tokenization on language models: An analysis for turkish. *ACM Transactions on Asian and Low-Resource Language Information Processing*, 22(4):1–21.

Nafis Irtiza Tripto and Mohammed Eunus Ali. 2018. Detecting multilabel sentiment and emotions from bangla youtube comments. In *2018 International Conference on Bangla Speech and Language Processing (ICBSLP)*, pages 1–6. IEEE.

## A Dataset Description

- **BanFake News** - (Hossain et al., 2020) introduces a dataset for detecting fake news. The dataset is consisted of 48K authentic and 1k fake news articles of different category. The tasks is classification tasks to find if a news is fake or not.

- **Sarcasm Detection** - This is Kaggle Compeition Dataset [1] where the organizer curated a dataset comprised of around 50K news headlines labeled in two categories: Sarcastic (1) or Not-Sarcastic (0).

- **SentNoB** - In SentNoB(Islam et al., 2021), public comments on news and videos were collected from social media for detecting the sentiment. The sentiment were labeled as Positive, Negative and Neutral. The training dataset size is 13.5K where validation and test dataset size is 1.5K

- **Emotion Detection** - (Tripto and Ali, 2018) collected user emotion dataset from YouTube user comments. The emotion detection dataset has 5 types of emotion: anger/disgust, joy, sadness, fear/surprise, and none.

- **Sentiment Classification** - (Tripto and Ali, 2018) also find the sentiment of the comments in the pervious dataset. We use five class sentiment dataset in this case. The sentiment were labeled as Strongly Positive ,Positive, Strongly Negative, Negative and Neutral.

## B Evaluation Metrics

In our experiment, we calculate Perplexity Score (PPL Score) for evaluation the model performance. It measures how well a probability distribution or language model predicts a given sample.

$$\text{Perplexity} = 2^{-\frac{1}{N}\sum_{i=1}^{N}\log_2 p(x_i)} \quad (13)$$

Here, $N$ represents the number of samples, and $p(x_i)$ is the probability assigned by the language model to the $i$-th sample $x_i$. A lower perplexity indicates better predictive performance, as the model can more accurately predict the given samples.

For the downstream tasks, we trace down Accuracy, F1 Score and ROC-AUC Score. The ROC-AUC metric measures the ability of a model to distinguish between positive and negative classes based on the area under the receiver operating characteristic curve.

## C BanglaBERT Baseline

For most of the dataset the performance of BanglaBERT isn't reported. For each dataset that mentioned above we finetuned BanglaBERT. The baseline result for BanglaBERT is reported below:

| Model Name | Acc ↑ | Macro F1 ↑ |
|---|---|---|
| BanFake | 96.65 | 92.99 |
| Sarcasm Detection | 93.30 | 49.00 |
| SentNoB | 74.46 | 69.55 |
| Emotion Detection | 70.78 | 41.26 |
| Sentiment Analysis | 54.11 | 42.59 |

Table 7: BanglaBERT baseline performance after fine-tuning it on afermentioned datasets.

[1] https://www.kaggle.com/competitions/nlp-competition-cuet-ete-day-2022/data