CNN for Modeling Sanskrit Originated Bengali and Hindi Language

Chowdhury Rafeed Rahman, MD. Hasibur Rahman,
Mohammad Rafsan, Samiha Zakir, Rafsanjani Muhammod
United International University
Mohammed Eunus Ali
Bangladesh University of Engineering and Technology
rafeed@cse.uiu.ac.bd

Abstract

Though recent works have focused on modeling high resource languages, the area is still unexplored for low resource languages like Bengali and Hindi. We propose an end-to-end trainable memory efficient CNN architecture named *CoCNN* to handle specific characteristics such as high inflection, morphological richness, flexible word order and phonetical spelling errors of Bengali and Hindi. In particular, we introduce two learnable convolutional sub-models at word and at sentence level that are end-to-end trainable. We show that state-of-the-art (SOTA) Transformer models including pretrained BERT do not necessarily yield the best performance for Bengali and Hindi. *CoCNN* outperforms pretrained BERT with 16X less parameters and achieves much better performance than SOTA LSTMs on multiple real-world datasets. This is the first study on the effectiveness of different architectures from Convolution, Recurrent, and Transformer neural net paradigm for modeling Bengali and Hindi. Code and data related to this research are available at: `https://bit.ly/3MkQUuI`

1 Introduction

Bengali and Hindi are the fourth and sixth most spoken language in the world, respectively. Both of these languages originated from Sanskrit (Staal, 1963) and share some unique characteristics that include (i) high inflection, i.e., each root word may have many variations due to addition of different suffixes and prefixes, (ii) morphological richness, i.e., there are large number of compound letters, modified vowels and modified consonants, and (iii) flexible word-order, i.e., the importance of word order and their positions in a sentence are loosely bounded (Examples shown in Figure 1). Many other languages such as Nepali, Gujarati, Marathi, Kannada, Punjabi and Telugu also share these characteristics. Neural language models (LM) have shown great promise recently in solving several key NLP tasks such as word prediction and sentence completion in major languages such as English and Chinese (Athiwaratkun et al., 2018; Takase et al., 2019; Pham et al., 2016; Gao et al., 2002; Cai and Zhao, 2016; Yang et al., 2016). To the best of our knowledge, none of the existing study investigates the efficacy of recent LMs in the context of Bengali and Hindi. We conduct an in-depth analysis of major deep learning architectures for LM and propose an end-to-end trainable memory efficient CNN architecture to address the unique characteristics of Bengali and Hindi.

Root Word	Inflected Variations
শোধ (repay)	পরি**শোধ** (pay back), প্রতি**শোধ** (revenge), **শোধি**ত (purified)
চল (trend)	**চল**তি (current), **চাল**ক (driver), **চল**মান (moving)
হার (lose)	পরি**হার** (leave), উপ**হার** (prize), **হার**জিত (competition)

High Inflection: different types of words derived from same root word

Valid Sentence Samples
আজ বিকালে রাষ্ট্রপতি আসবেন রাষ্ট্রপতি **আজ** বিকালে আসবেন রাষ্ট্রপতি আসবেন **আজ** বিকালে (The president will come this afternoon)
খুনি সন্দেহে পাঁচজনকে আটক করেছে পুলিশ পুলিশ পাঁচজনকে আটক করেছে **খুনি** সন্দেহে পুলিশ পাঁচজনকে **খুনি** সন্দেহে আটক করেছে (The police have arrested five people because of murder suspicion)

Flexible Word-Order: Each of the three sentences are valid and carry the same meaning, but their word order is very different from one another

Compound Character	Component Characters
ক্ত	ক + ্ + ত
ক্ষ	ক + ্ + ষ
ব্র	ব + ্ + র
ন্ত্য	ন+ ্ +ত+ ্ +য

Morphological Richness: Around 170 compound characters in Bengali each consisting of 3-5 simple characters

Figure 1: Bengali language unique characteristics

State-of-the-art (SOTA) techniques for LM can be categorized into three sub-domains of deep learning: (i) convolutional neural network (CNN) (Pham et al., 2016; Wang et al., 2018) (ii) recurrent neural network (Bojanowski et al., 2017; Mikolov et al., 2012; Kim et al., 2016; Gerz et al., 2018), and (iii) Transformer attention network (Al-Rfou et al., 2019; Vaswani et al., 2017; Irie et al., 2019; Ma et al., 2019). Long Short Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) based models, which are suitable for learning sequence and word order information, are not effective for modeling Bengali and Hindi due to their flexible word order characteristic. On the other hand, Transformers use

Proceedings of the 2nd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 12th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), pages 47–56
November 20–23, 2022.

dense layer based multi-head attention mechanism. They lack the ability to learn local patterns in sentence level, which in turn puts negative effect on modeling languages with loosely bound word order. Most importantly, neither LSTMs nor Transformers use any suitable measure to learn intra-word level local pattern necessary for modeling highly inflected and morphologically rich languages.

We observe that learning inter (flexible word order) and intra (high inflection and morphological richness) word local patterns is of paramount importance for Bengali and Hindi LM. To accommodate such characteristics, we design a novel CNN architecture, namely **Coordinated CNN (CoCNN)** that achieves SOTA performance with low training time. In particular, *CoCNN* consists of two learnable convolutional sub-models: word level (*Vocabulary Learner (VL)*) and sentence level (*Terminal Coordinator (TC)*). *VL* is designed for syllable pattern learning, whereas *TC* serves the purpose of word coordination learning while maintaining positional independence, which suits the flexible word order of Bengali and Hindi. *CoCNN* does not explicitly incorporate any self attention mechanism like Transformers; rather it relies on *TC* for emphasizing on important word patterns. *CoCNN* achieves significantly better performance than pretrained BERT for Bengali and Hindi LM with 16X less parameters. We further enhance *CoCNN* by introducing skip connection and parallel convolution branches in *VL* and *TC*, respectively. This modified architecture (with negligible increase in parameter number) is named as *CoCNN+*. We validate the effectiveness of *CoCNN+* on a number of tasks that include next word prediction in erroneous setting, text classification, sentiment analysis and spell checking. *CoCNN+* shows superior performance than contemporary LSTM based models and pretrained BERT.

In summary, the contributions of this paper are as follows:

- An end-to-end trainable *CoCNN* model based on the coordination of two CNN sub-models
- In-depth analysis and comparison on different SOTA LMs in three paradigms: CNN, LSTM, and Transformer
- Some simple modifications in *CoCNN* to achieve even better performance
- Using *VL* sub-model of *CoCNN+* as an effective spell checker for Bengali

2 Our Approach

Traditional CNN based approaches (Pham et al., 2016) represent the entire input sentence/ paragraph using a matrix of size $S_N \times S_V$, where S_N and S_V represent number of characters in the sentence/ paragraph and the character representation vector size, respectively. In such character based approach, the model does not have the ability to consider each word in the sentence as a separate entity. However, it is important to understand the contextual meaning of each word and to find out relationship among those words for sentence semantics understanding. **Coordinated CNN (CoCNN)** is aimed to achieve this feat. Figure 2 illustrates *CoCNN* that has two major components. *Vocabulary Learner* component works at word level, while *Terminal Coordinator* component works at sentence/ paragraph level. Both of these components are 1D CNN based sub-model at their core and are trained end-to-end.

2.1 *Vocabulary Learner*

Vocabulary Learner (VL) is used to transform each input word into a vector representation called *CNNvec*. We represent each input word $Word_i$ by a matrix W_i. W_i consists of m vectors each of size len_C. These vectors $\vec{C_1}, \vec{C_2}, \ldots \vec{C_m}$ represent one hot vector of character $C_1, C_2, \ldots C_m$, respectively of $Word_i$. Representation detail has been depicted in the bottom right corner of Figure 2. Applying 1D convolution (*conv*) layers on matrix W_i helps in deriving key local patterns and sub-word information of $Word_i$. After passing W_i matrix through the first *conv* layer, we obtain feature matrix W_i^1. Passing W_i^1 through the second *conv* layer provides us with feature matrix W_i^2. So, the L^{th} *conv* layer provides us with feature matrix W_i^L. *VL* sub-model consists of such 1D *conv* layers standing sequentially one after the other. *Conv* layers near matrix W_i are responsible for identifying key sub-word patterns of $Word_i$, while *conv* layers further away focus on different combinations of these key sub-word patterns. Such word level local pattern recognition plays key role in identifying semantic meaning of a word irrespective of inflection or presence of spelling error. Each intermediate *conv* layer output is batch normalized. The final *conv* layer output matrix W_i^L is flattened and formed into a vector F_i of size len_F. F_i is the *CNNvec* representation of $Word_i$. We obtain *CNNvec* representation from each of our input words in a similar fashion

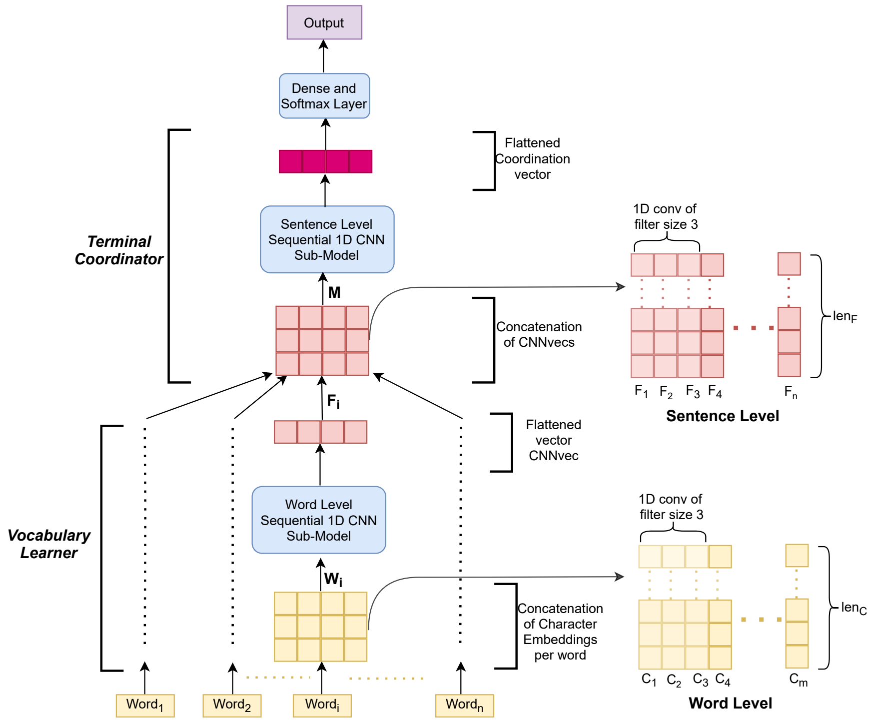


Figure 2: 1D CNN based CoCNN architecture

applying the same *CNN* sub-model.

2.2 *Terminal Coordinator*

Terminal Coordinator (TC) takes the *CNNvecs* obtained from *VL* as input and returns a single *Coordination vector* as output which is used for final prediction. For n words $Word_1, Word_2, \ldots Word_n$; we obtain n such *CNNvecs* $\vec{F_1}, \vec{F_2}, \ldots \vec{F_n}$, respectively. Each *CNNvec* is of size len_F. Concatenating these *CNNvecs* provide us with matrix M (details shown in the middle right portion of Figure 2). Applying 1D *conv* on matrix M facilitates the derivation of key local patterns found in input sentence/ paragraph which is crucial for output prediction. A sequential 1D CNN sub-model with design similar to *VL* having different sets of weights is employed on matrix M. *Conv* layers near M are responsible for identifying key word clusters, while *conv* layers further away focus on different combinations of these key word clusters important for sentence or paragraph level local pattern recognition. The final output feature matrix obtained from the 1D CNN sub-model of *TC* is flattened to obtain the *Coordination vector*, a summary of important information obtained from the input word sequence in order to predict the correct output.

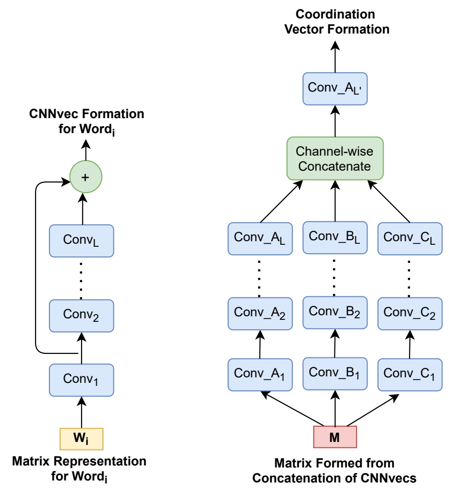


Figure 3: *CoCNN+* architecture with its modified *VL* (left) and *TC* (right). $Conv_L$ means L^{th} *conv* layer, whereas *Conv_A* means a *conv* layer with filter size A.

2.3 Extending CoCNN

We perform two simple modifications in *CoCNN* to form *CoCNN+* architecture with minimal increase in parameter number (see Figure 3).

First, we modify the CNN sub-model of *VL*. We add the output feature matrix of the first *conv* layer $Conv_1$ with the output feature matrix of the last *conv* layer $Conv_L$. We pass the resultant feature matrix on to subsequent layers (same as *CoCNN*) for *CNNvec* formation of $Word_i$. Such modification helps in two cases - (i) it eliminates the gradient vanishing problem of the first *conv* layer of *VL* and (ii) it gives *CNNvec* access to both low level and high level features of the corresponding input word.

Second, we modify the CNN sub-model of *TC* by passing matrix M simultaneously to three 1D CNN branches. The *conv* filter sizes of the left, middle and right branches are A, B and C, respectively; where, $A < B$ and $B < C$. The outputs from the three branches are concatenated channel-wise and are then passed on to the final *conv* layer having filter size A. The output feature matrix is passed on to subsequent layers (same as *CoCNN*) for *Coordination vector* formation. Multiple *conv* branches with different filter sizes help in learning both short and long range local patterns, especially when the input sentence or document is long.

3 Experimental Setup

3.1 Dataset Specifications

Bengali dataset consists of articles from online public news portals such as Prothom-Alo (Rahman, 2017), BDNews24 (Khalidi, 2015) and Nayadiganta (Mohiuddin, 2019). The articles encompass domains such as politics, entertainment, lifestyle, sports, technology and literature. The Hindi dataset consists of Hindinews (Pandey, 2018), Livehindustan (Shekhar, 2018) and Patrika (Jain, 2018) newspaper articles available open source in Kaggle encompassing similar domains. Nayadiganta (Bengali) and Patrika (Hindi) datasets have been used only as independent test sets. Detailed statistics of the datasets are provided in Table 1. Top words have been selected such that they cover at least 90% of the dataset. For each Bengali dataset, we have created a new version of the dataset by incorporating spelling errors using a probabilistic error generation algorithm (Sifat et al., 2020), which enables us to test the effectiveness of LMs for erroneous datasets.

3.2 Performance Metric

We use perplexity (PPL) to assess the performance of the models for next word prediction task. Suppose, we have sample inputs $I_1, I_2, \ldots, I_n$ and our model provides probability values $P_1, P_2, \ldots, P_n$, respectively for their ground truth output tokens. Then the PPL score of our model for these samples can be computed as:

$$PPL = \exp(-\frac{1}{n}\sum_{i=1}^{n}\ln(P_i))$$

For text classification and sentiment analysis, we use *accuracy* and *F1 score* as our performance metric.

3.3 Model Optimization

For model optimization, we use SGD optimizer with a learning rate of 0.001 while constraining the norm of the gradients to below 5 for exploding gradient problem elimination. We use Categorical Cross-Entropy loss for model weight update and dropout (Hinton et al., 2012) with probability 0.3 between the dense layers for regularization. We use Relu (Rectified Linear Unit) as hidden layer activation function. We use a batch size of 64. As we apply batch normalization on CNN intermediate outputs, we do not use any other regularization effect such as dropout on these layers (Luo et al., 2018).

We use Anaconda 3 with Python 3.8 version and Tensorflow 2.6.0 framework (Abadi et al., 2016) for our implementation. We use two GPU servers for training our models: (i) 12 GB Nvidia Titan Xp GPU, Intel(R) Core(TM) i7-7700 CPU (3.60GHz) processor model (ii) 32 GB RAM with 8 cores 24 GB Nvidia Tesla K80 GPU, Intel(R) Xeon(R) CPU (2.30GHz) processor model

3.4 CoCNN Hyperparameters

3.4.1 *Vocabulary Learner* Details

Vocabulary Learner sub-model consists of a character level embedding layer producing a 40 size vector from each character, then four consecutive layers each consisting of 1D convolution (batch normalization and Relu activation between each pair of convolution layers) and finally, a 1D global max-pooling in order to obtain *CNNvec* representation from each input word. The four 1D convolution layers consist of $(32, 2), (64, 3), (64, 3), (128, 4)$ convolution, respectively. Here the first and second element of each tuple denote number of convolution filters and kernel size, respectively. As we can see, the filter size and number of filters of the convolution layers are monotonically increasing as architecture depth increases. It is because deep convolution layers need to learn the combination of various low level features which is a more difficult task compared to the task of shallow layers that include extraction of low level features.

Datasets	No. of Unique words	No. of Unique Characters	No. of Top Words	No. of Training Samples	No. of Validation Samples
Prothom-Alo	260 K	75	13 K	5.9 M	740 K
BDNews24	170 K	72	14 K	2.9 M	330 K
Nayadiganta	44 K	73	_	_	280 K
Hindinews	37 K	74	5.5 K	87 K	10 K
Livehindustan	60 K	73	4.5 K	210 K	20 K
Patrika	28 K	73	_	_	307 K

Table 1: Dataset details (K and M denote 10^3 and 10^6 multiplier, respectively)

3.4.2 *Terminal Coordinator* Details

The *Terminal Coordinator* sub-model used in *CoCNN* architecture uses six convolution layers which consist of $(32, 2), (64, 3), (64, 3), (96, 3), (128, 4), (196, 4)$ convolution. Its design is similar to that of *Vocabulary Learner* sub-model. The final output feature matrix obtained from this CNN sub-model is flattened to get the *Coordination vector*. After passing this vector through a couple of dense layers, we use *Softmax* activation function at the final output layer to get the predicted output.

3.5 CoCNN+ Hyperparameters

The CNN sub-model of *Vocabulary Learner* in *CoCNN+* is the same as *CoCNN* except for one aspect (see Figure 3) - we change the first convolution layer to have 128 filters of size 2 instead of 32 filters. This is done to respect the matrix dimensionality during skip connection based addition.

Instead of providing a sequential 1D CNN sub-model in *Terminal Coordinator*, we provide three parallel branches each consisting of four convolution layers (see Figure 3) where the filter numbers are 32, 64, 96 and 128. The filter size of the leftmost, middle and the rightmost branch are 3, 5 and 7, respectively. All convolution operations are dimension preserving through the use of padding. The feature matrices of all three of these branches are concatenated channel-wise and finally, this concatenated matrix is passed on to a final convolution layer with 196 filters of size 3.

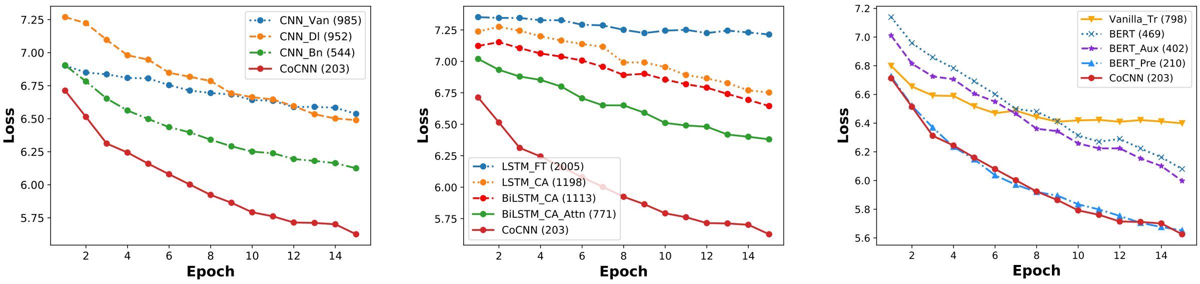
4 Results and Discussion

4.1 Comparing *CoCNN* with Other CNNs

We compare *CoCNN* with three other CNN-based baselines (see Figure 4a). *CNN_Van* is a simple sequential 1D CNN model of moderate depth (Pham et al., 2016). It considers the full input sentence/ paragraph as a matrix. The matrix consists of character representation vectors. *CNN_Dl* uses dilated *conv* in its CNN layers which allows the model to have a larger field of view (Roy, 2019). Such a change in *conv* strategy shows slight performance improvement. *CNN_Bn* has the same setting as of *CNN_Van*, but uses batch normalization on intermediate *conv* layer outputs. Such a measure shows significant performance improvement in terms of loss and PPL score. Proposed *CoCNN* surpasses the performance of *CNN_Bn* by a wide margin. We believe that the ability of *CoCNN* to consider each word of a sentence as a separate meaningful entity is the reason behind this drastic improvement.

4.2 Comparing *CoCNN* with SOTA LSTMs

We compare *CoCNN* with four LSTM-based models (see Figure 4b). Two LSTM layers are stacked on top of each other in all four of these models. We do not compare with LSTM models that use *Word2vec* (Rong, 2014) representation as this representation requires fixed size vocabulary. In spelling error prone setting, vocabulary size is theoretically infinite. We start with *LSTM_FT*, an architecture using sub-word based *FastText* representation (Athiwaratkun et al., 2018; Bojanowski et al., 2017). Character aware learnable layers per LSTM time stamp form the new generation of SOTA LSTMs (Mikolov et al., 2012; Kim et al., 2016; Gerz et al., 2018; Assylbekov et al., 2017). *LSTM_CA* acts as their representative by introducing variable size parallel *conv* filter output concatenation as word representation. The improvement over *LSTM_FT* in terms of PPL score is almost double. Instead of unidirectional many to one LSTM, we introduce bidirectional LSTM in *LSTM_CA* to form *BiLSTM_CA* which shows slight performance improvement. We introduce Bahdanu attention (Bahdanau et al., 2014) on *BiLSTM_CA* to form *BiLSTM_CA_Attn* architecture. Such measure shows further performance boost. *CoCNN* shows almost four times improvement in PPL score compared to *BiLSTM_CA_Attn*. If we compare Figure 4b and 4a, we can see that CNNs perform relatively better than LSTMs in general for Bengali



(a) CNN paradigm (b) LSTM paradigm (c) Transformer paradigm

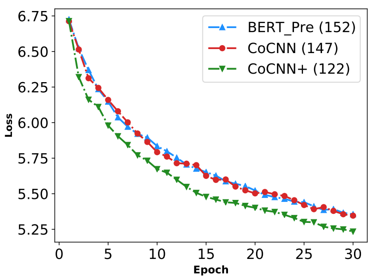
Figure 4: Comparing *CoCNN* with SOTA architectures from CNN, LSTM and Transformer paradigm on Prothom-Alo validation set. The score shown beside each model name denotes that model's PPL score on Prothom-Alo validation set after 15 epochs of training. Note that this dataset contains synthetically generated spelling errors.

LM. LSTMs have a tendency of learning sequence order information which imposes positional dependency. Such characteristic is unsuitable for Bengali and Hindi with flexible word order.

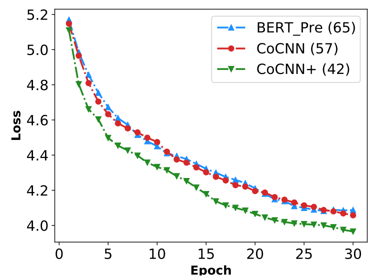
4.3 Comparing *CoCNN* with SOTA Transformers

We compare *CoCNN* with four Transformer-based models (see Figure 4c). We use popular *FastText* word representation with all compared transformers. Our comparison starts with *Vanilla_Tr*, a single Transformer encoder (similar to the Transformer designed by Vaswani et al. (2017)). In *BERT*, we stack 12 transformers on top of each other where each Transformer encoder has more parameters than the Transformer of *Vanilla_Tr* (Kenton and Toutanova, 2019; Irie et al., 2019). *BERT* with its large depth and enhanced encoders almost double the performance shown by *Vanilla_Tr*. We do not pretrain this *BERT* architecture. We follow the Transformer architecture designed by Al-Rfou et al. (2019) and introduce auxiliary loss after the Transformer encoders situated near the bottom of the Transformer stack of *BERT* to form *BERT_Aux*. Introduction of such auxiliary losses shows moderate improvement of performance. *BERT_Pre* is the pretrained version of *BERT*. We follow the word masking based pretraining scheme of Liu et al. (2019). The Bengali pretraining corpus consists of Prothom Alo (Rahman, 2017) news articles dated from 2014-2017 and BDNews24 (Khalidi, 2015) news articles dated from 2015-2017. The performance of *BERT* jumps up more than double when such pretraining is applied. *CoCNN* without utilizing any pretraining achieves marginally better performance than *BERT_Pre*. Unlike Transformer encoders, *conv* imposes attention with a view to extracting important patterns from the input to provide the correct output. Furthermore, *VL* of *CoCNN* is suitable for deriving semantic meaning of each input word in highly inflected and error prone settings.

4.4 Comparing *BERT_Pre*, *CoCNN* and *CoCNN+*



(a) Plot on Bengali dataset



(b) Plot on Hindi dataset

Figure 5: Comparing *BERT_Pre*, *CoCNN* and *CoCNN+* on Bengali (Prothom-Alo) and Hindi (Hindinews and Livehindustan merged) validation set. The score shown beside each model name denotes that model's PPL score after 30 epochs of training on corresponding training set.

BERT_Pre is the only model showing perfor-

Datasets	Error?	BERT_Pre	Co-CNN	Co-CNN+
Prothom Alo	Yes	152	147	**122**
	No	117	114	**99**
BDNews 24	Yes	201	193	**170**
	No	147	141	**123**
Hindinews Hindustan	No	65	57	**42**
Naya Diganta	Yes	169	162	**143**
	No	136	133	**118**
Patrika	No	67	57	**44**

Table 2: PPL Score Comparison

mance close to *CoCNN* in terms of validation loss and PPL score (see Figure 4). We compare these two models with *CoCNN+*. We train the models for 30 epochs on several Bengali and Hindi datasets and obtain their PPL scores on corresponding validation sets (training and validation set were split at 80%-20% ratio). Bengali datasets include Prothom-Alo, BDNews24; while Hindi dataset includes Hindinews, Livehindustan. We use Nayadiganta and Patrika dataset for Bengali and Hindi independent test set, respectively. The Hindi pretraining corpus consists of Hindi Oscar Corpus (Thakur, 2019), preprocessed Wikipedia articles (Gaurav, 2019), HindiEnCorp05 dataset (Bojar et al., 2014) and WMT Hindi News Crawl data (Barrault et al., 2019). From the graphs of Figure 5 and PPL score comparison Table 2, it is evident that *CoCNN* marginally outperforms its nemesis *BERT_Pre* in all cases, while *CoCNN+* outperforms both *CoCNN* and *BERT_Pre* by a significant margin. There are 8 sets of PPL scores in Table 2 for the three models on eight different dataset settings. We use these scores to perform a one-tailed paired t-test in order to determine whether the reduction of PPL score seen in *CoCNN+* is statistically significant when P-value threshold is set to 0.05. The test shows that the improvement is indeed significant compared to both *BERT_Pre* and *CoCNN*. Number of parameters of *BERT_Pre*, *CoCNN* and *CoCNN+* are 74 M, 4.5 M and 4.8 M, respectively. Though the parameter number of *CoCNN+* and *CoCNN* is close, *CoCNN+* has 15X fewer parameters than *BERT_Pre*.

4.5 Comparison in Downstream Tasks

We have compared *BERT_Pre* and *CoCNN+* in three different downstream tasks:

Dataset	BERT_Pre	CoCNN+
Question Classify	0.905	**0.926**
Product Review	0.841	**0.86**
Hate Speech	0.77	**0.781**

Table 3: Performance comparison between *BERT_Pre* and *CoCNN+* in three downstream tasks (F1 score)

(1) **Bengali Question Classification (QC):** This task consists of six classes (entity, numeric, human, location, description and abbreviation type question). The dataset has 3350 question samples (Islam et al., 2016).
(2) **Hindi Product Review Classification:** The task is to classify a review into positive or negative class where the dataset consists of 2355 sample reviews (Kakwani, 2020).
(3) **Hindi Hate Speech Detection:** The task is to identify whether a provided speech is a hate speech or not. The dataset consists of 3654 speeches (HASOC, 2019).

We use **five fold cross validation** while performing comparison on these datasets (see mean results in Table 3) in terms of F1 score. One tailed independent t-tests with a P-value threshold of 0.05 has been performed on the 5 validation F1 scores obtained from five fold cross validation of each of the two models. Our **statistical test** results validate the significance of the improvement shown by *CoCNN+* for all three of the mentioned tasks.

Spell Checker Algorithm	Synthetic Error	Real Error
Vocabulary Learner	71.1%	61.1%
Phonetic Rule	61.5%	32.5%
Clustering Rule	51.8%	43.8%

Table 4: Bengali spelling correction (accuracy)

We also investigate the potential of *VL* of *CoCNN+* as a Bengali spell checker (SC). Both *CoCNN* and *CoCNN+* model use *VL* for producing CNNvec representation from each input word. We extract the CNN sub-model of *VL* from our trained (on Prothom-Alo dataset) *CoCNN+* model. We produce CNNvec for all 13 K top words of Prothom-Alo dataset. For any error word, W_e, we can generate its CNNvec V_e using *VL*. We can calculate cosine similarity, Cos_i between V_e and CNNvec

V_i of each top word W_i. Higher cosine similarity means greater probability of being the correct version of W_e. We have discovered such approach to be effective for correct word generation. Recently, a phonetic rule based approach has been proposed by Saha et al. (2019), where a hybrid of Soundex (UzZaman and Khan, 2004) and Metaphone (UzZaman and Khan, 2005) algorithm has been used for Bengali word level SC. Another SC proposed in recent time has taken a clustering based approach (Mandal and Hossain, 2017). We compare our proposed *VL* based SC with these two existing SCs (see Table 4). Both the real and synthetic error dataset consist of 20k error words formed from the top 13 K words of Prothom-Alo dataset. The real error dataset has been collected from a wide range of Bengali native speakers using an easy to use web app. Results show the superiority of our proposed SC over existing approaches.

5 Related Works

Although a significant number of works for LM of high resource languages like English and Chinese are available, very few researches of significance for LM in low resource languages like Bengali and Hindi exist. In this section, we mainly summarize major LM related research works.

Sequence order information based statistical RNN models such as LSTM and GRU have been popular for LM tasks (Mikolov et al., 2011). Sundermeyer et al. (2012) showed the effectiveness of LSTM for English and French LM. The regularizing effect on LSTM was investigated by Merity et al. (2017). SOTA LSTM models learn sub-word information in each time stamp. Bojanowski et al. (2017) proposed a morphological information oriented character N-gram based word vector representation. It was improved by Athiwaratkun et al. (2018) and is known as FastText. Mikolov et al. (2012) proposed a technique for learning sub-word level information from data, while such an idea was integrated in a character aware LSTM model by Kim et al. (2016). Takase et al. (2019) further improved word representation by combining ordinary word level and character-aware embedding. Assylbekov et al. (2017) showed that character-aware neural LMs outperform syllable-aware ones. Gerz et al. (2018) evaluated such models on 50 morphologically rich languages.

Self attention based Transformers have become the SOTA mechanism for sequence to sequence modeling in recent years (Vaswani et al., 2017). Some recent works have explored the use of such models in LM. Deep Transformer encoders outperform stacked LSTM models (Irie et al., 2019). A deep stacked Transformer model utilizing auxiliary loss was proposed by Al-Rfou et al. (2019) for character level language modeling. The multi-head self attention mechanism was replaced by a multi-linear attention mechanism with a view to improving LM performance and reducing parameter number (Ma et al., 2019). Bengali and Hindi language, having unique characteristics, remain open as to what strategy to use for model development in such domains.

One dimensional version of CNNs have been used recently for text classification oriented tasks (Wang et al., 2018; Moriya and Shibata, 2018; Le et al., 2018). Pham et al. (2016) studied CNN application in LM showing the ability of CNNs to extract LM features at a high level of abstraction. Furthermore, dilated *conv* was employed in Bengali LM with a view to solving long range dependency problem (Roy, 2019).

6 Conclusion

We have proposed *Coordinated CNN (CoCNN)* that introduces two 1D CNN based key concepts: word level *VL* and sentence level *TC*. Detailed investigation in three deep learning paradigms (CNN, LSTM and Transformer) shows the effectiveness of *CoCNN* in Bengali and Hindi LM. We have also shown a simple but effective enhancement of *CoCNN* by introducing skip connection and parallel *conv* branches in the *VL* and *TC* portion, respectively. Future research may incorporate interesting ideas from existing SOTA 2D CNNs in *CoCNN*. Over-parametrization and innovative scheme for *CoCNN* pretraining are expected to increase its LM performance even further. Code has been provided as supplementary material. Dataset will be made publicly available upon acceptance.

Acknowledgments

This work was supported by Bangladesh Information and Communication Technology (ICT) division [grant number: 56.00.0000.028.33.095.19-85] as part of their Enhancement of Bengali Language project.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*.

Rami Al-Rfou, Dokook Choe, Noah Constant, Mandy Guo, and Llion Jones. 2019. Character-level language modeling with deeper self-attention. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3159–3166.

Zhenisbek Assylbekov, Rustem Takhanov, Bagdat Myrzakhmetov, and Jonathan N. Washington. 2017. Syllable-aware neural language models: A failure to beat character-aware ones. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1866–1872, Copenhagen, Denmark. Association for Computational Linguistics.

Ben Athiwaratkun, Andrew Wilson, and Anima Anandkumar. 2018. Probabilistic FastText for multi-sense word embeddings. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1–11, Melbourne, Australia. Association for Computational Linguistics.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Loïc Barrault, Ondřej Bojar, Marta R. Costa-jussà, Christian Federmann, Mark Fishel, Yvette Graham, Barry Haddow, Matthias Huck, Philipp Koehn, Shervin Malmasi, Christof Monz, Mathias Müller, Santanu Pal, Matt Post, and Marcos Zampieri. 2019. Findings of the 2019 conference on machine translation (WMT19). In *Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1)*, pages 1–61, Florence, Italy. Association for Computational Linguistics.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Ondřej Bojar, Vojtěch Diatka, Pavel Straňák, Aleš Tamchyna, and Daniel Zeman. 2014. HindEnCorp 0.5. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Deng Cai and Hai Zhao. 2016. Neural word segmentation learning for chinese. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 409–420.

Jianfeng Gao, Joshua Goodman, Mingjing Li, and Kai-Fu Lee. 2002. Toward a unified approach to statistical language modeling for chinese. *ACM Transactions on Asian Language Information Processing (TALIP)*, 1(1):3–33.

Gaurav. 2019. Wikipedia. `https://www.kaggle.com/disisbig/hindi-wikipedia-articles-172k`.

Daniela Gerz, Ivan Vulić, Edoardo Ponti, Jason Naradowsky, Roi Reichart, and Anna Korhonen. 2018. Language modeling for morphologically rich languages: Character-aware modeling for word-level prediction. *Transactions of the Association for Computational Linguistics*, 6:451–465.

HASOC. 2019. Hindi hate speech dataset. `https://hasocfire.github.io/hasoc/2019/dataset.html`.

Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

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

Kazuki Irie, Albert Zeyer, Ralf Schlüter, and Hermann Ney. 2019. Language modeling with deep transformers. *arXiv preprint arXiv:1905.04226*.

Md Aminul Islam, Md Fasihul Kabir, Khandaker Abdullah-Al-Mamun, and Mohammad Nurul Huda. 2016. Word/phrase based answer type classification for bengali question answering system. In *2016 5th International Conference on Informatics, Electronics and Vision (ICIEV)*, pages 445–448. IEEE.

Bhuwnesh Jain. 2018. Patrika. `https://epaper.patrika.com/`.

Divyanshu Kakwani. 2020. Ai4bharat. `https://github.com/ai4bharat`.

Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, pages 4171–4186.

Toufique Imrose Khalidi. 2015. Bdnews24. `https://bdnews24.com/`.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. 2016. Character-aware neural language models. In *Thirtieth AAAI conference on artificial intelligence*.

Hoa T Le, Christophe Cerisara, and Alexandre Denis. 2018. Do convolutional networks need to be deep for text classification? In *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Ping Luo, Xinjiang Wang, Wenqi Shao, and Zhanglin Peng. 2018. Towards understanding regularization in batch normalization. In *International Conference on Learning Representations*.

Xindian Ma, Peng Zhang, Shuai Zhang, Nan Duan, Yuexian Hou, Ming Zhou, and Dawei Song. 2019. A tensorized transformer for language modeling. In *Advances in Neural Information Processing Systems*, pages 2232–2242.

Prianka Mandal and BM Mainul Hossain. 2017. Clustering-based bangla spell checker. In *2017 IEEE International Conference on Imaging, Vision & Pattern Recognition (icIVPR)*, pages 1–6. IEEE.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2017. Regularizing and optimizing lstm language models. *arXiv preprint arXiv:1708.02182*.

Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. 2011. Extensions of recurrent neural network language model. In *2011 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5528–5531. IEEE.

Tomáš Mikolov, Ilya Sutskever, Anoop Deoras, Hai-Son Le, Stefan Kombrink, and Jan Cernocky. 2012. Subword language modeling with neural networks. *preprint (http://www. fit. vutbr. cz/imikolov/rnnlm/char. pdf)*, 8:67.

Alamgir Mohiuddin. 2019. Nayadiganta. https://www.dailynayadiganta.com/.

Shun Moriya and Chihiro Shibata. 2018. Transfer learning method for very deep cnn for text classification and methods for its evaluation. In *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, volume 2, pages 153–158. IEEE.

Sanjay Pandey. 2018. Hindinews. https://www.dailyhindinews.com/.

Ngoc-Quan Pham, German Kruszewski, and Gemma Boleda. 2016. Convolutional neural network language models. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1153–1162.

Matiur Rahman. 2017. Prothom-alo. https://www.prothomalo.com/.

Xin Rong. 2014. word2vec parameter learning explained. *arXiv preprint arXiv:1411.2738*.

Shuvendu Roy. 2019. Improved bangla language modeling with convolution. In *2019 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT)*, pages 1–4. IEEE.

Sourav Saha, Faria Tabassum, Kowshik Saha, and Marjana Akter. 2019. *BANGLA SPELL CHECKER AND SUGGESTION GENERATOR*. Ph.D. thesis, United International University.

Shashi Shekhar. 2018. Livehindustan. https://www.livehindustan.com/.

Md Habibur Rahman Sifat, Chowdhury Rafeed Rahman, Mohammad Rafsan, and Hasibur Rahman. 2020. Synthetic error dataset generation mimicking bengali writing pattern. In *2020 IEEE Region 10 Symposium (TENSYMP)*, pages 1363–1366. IEEE.

J Fritz Staal. 1963. Sanskrit and sanskritization. *The Journal of Asian Studies*, 22(3):261–275.

Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. 2012. Lstm neural networks for language modeling. In *Thirteenth annual conference of the international speech communication association*.

Sho Takase, Jun Suzuki, and Masaaki Nagata. 2019. Character n-gram embeddings to improve rnn language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 5074–5082.

Abhishek Thakur. 2019. Hindi oscar corpus. https://www.kaggle.com/abhishek/hindi-oscar-corpus.

Naushad UzZaman and Mumit Khan. 2004. A bangla phonetic encoding for better spelling suggesions. Technical report, BRAC University.

Naushad UzZaman and Mumit Khan. 2005. A double metaphone encoding for bangla and its application in spelling checker. In *2005 International Conference on Natural Language Processing and Knowledge Engineering*, pages 705–710. IEEE.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Shiyao Wang, Minlie Huang, and Zhidong Deng. 2018. Densely connected cnn with multi-scale feature attention for text classification. In *IJCAI*, pages 4468–4474.

Tzu-Hsuan Yang, Tzu-Hsuan Tseng, and Chia-Ping Chen. 2016. Recurrent neural network-based language models with variation in net topology, language, and granularity. In *2016 International Conference on Asian Language Processing (IALP)*, pages 71–74. IEEE.