STATISTICS, OPTIMIZATION AND INFORMATION COMPUTING
Stat., Optim. Inf. Comput., Vol. **9**, June 2021, pp 334–350.
Published online in International Academic Press (www.IAPress.org)

Context-driven Bengali Text Generation using Conditional Language Model

Md. Raisul Kibria [*], Mohammad Abu Yousuf

Institute of Information Technology, Jahangirnagar University, Bangladesh

Abstract Text generation is a rapidly evolving field of *Natural Language Processing* (NLP) with larger Language models proposed very often setting new state-of-the-art. These models are extremely effective in learning the representation of words and their internal coherence in a particular language. However, an established context-driven, end to end text generation model is very rare, even more so for the Bengali language. In this paper, we have proposed a Bidirectional *gated recurrent unit* (GRU) based architecture that simulates the conditional language model or the decoder portion of the *sequence to sequence* (seq2seq) model and is further conditioned upon the target context vectors. We have explored several ways of combining multiple context words into a fixed dimensional vector representation that is extracted from the same GloVe language model which is used to generate the embedding matrix. We have used beam search optimization to generate the sentence with the maximum cumulative log probability score. In addition, we have proposed a human scoring based evaluation metric and used it to compare the performance of the model with unidirectional LSTM and GRU networks. Empirical results prove that the proposed model performs exceedingly well in producing meaningful outcomes depicting the target context. The experiment leads to an architecture that can be applied to an extensive domain of context-driven text generation based applications and which is also a key contribution to the NLP based literature of the Bengali language.

Keywords Natural Language Processing, Bengali Text Generation, Context Representation, Bidirectional GRU

AMS 2010 subject classifications 68T20, 68T50, 91E40

DOI: 10.19139/soic-2310-5070-1061

1. Introduction

Text generation is the task of producing texts automatically given some contexts or goals, that are indistinguishable from human-written texts. It is a subfield of Natural Language Processing (NLP) and a derivative of Computer linguistics and Artificial Intelligence (AI). The scope of a text generation system is large including text summarization, typing assistant, machine translation, image captioning, automatic report generation, article generation, etc. The success of such a system mainly depends on the correctness and quality of the generated text depending on how comprehensible they are and how accurate they are grammatically.

Implementations of automatic text generation for structured texts such as codes or URLs have been widely available for a long time. Generation processes for such structured data are relatively straightforward and can be addressed through conventional programming approaches eg.- a starting tag in a markup language will have an ending tag of the same name. But for unstructured texts, the task is very difficult as these data can not be identified through any particular probability distribution. The earliest approach for such data involved direct translation of texts from knowledge structures [1]. Many other early attempts involved the use of a pipelined generic architecture [2] [3]. But with the rise of deep neural networks, the field has almost been revolutionized. Since deep neural networks are extremely effective in representing complex probability distributions, as in natural

[*]Correspondence to: Md. Raisul Kibria (Email: raisul.inc@gmail.com). Institute of Information Technology, Jahangirnagar University, Savar, Dhaka-1342, Bangladesh.

ISSN 2310-5070 (online) ISSN 2311-004X (print)

language texts, different neural networks have been explored frequently by researchers to achieve tremendous feats in text generation tasks.

In order to model the temporal dependency of sequential data like texts, Recurrent Neural Networks (RNNs) were introduced. RNNs are an extension to standard feedforward networks, where each time step incorporates a high dimensional hidden state and passes it to the next time step during forward propagation to flow the sequential information. But standard RNNs can not maintain long-range temporal dependencies due to vanishing or much rarely, exploding gradients. Several variants of standard RNNs have since been introduced which effectively addressed all the gradient issues. The most notable of these are- Long-short Term Memory (LSTM) and Gated Recurrent Unit (GRU) nets. LSTM and GRU both have special gated units controlling memory cells to determine how and when to update their internal cell states, the difference between the two being that GRU is a much simplified version of LSTM.

RNNs and their variants are widely used to implement Language Models (LM). LMs can learn the various representations of the vocabulary of a particular language and its syntax and semantics. These models can be trained through maximum likelihood estimation and can predict sequences by sampling from the learned probability distribution. But such implementation requires some input $(x_0, x_1, \dots)$ in order to predict the next words in the sequence. This input has to be sufficiently large to control the context of the target sequence, thus being not quite effective in generating a sentence within the target context.

Sequence to Sequence (seq2seq) models, first introduced by Sutskever et al. in 2014 [4] are largely used in Neural Machine Translation (NMT) which can produce sequences with minimal assumption on the sequence structure. The architecture of such a model includes an encoder RNN (often implemented as LSTM or GRU), that transforms a reference sequence into a fixed dimensional context vector using a LM, which is then fed to the decoder RNN to reproduce the sequence into the target distribution. These models are exceptionally accurate in mapping sequences to other target sequences. The decoder model is also called a conditional language model because unlike LMs the decoder produces sequences based on the provided conditions with or without any direct input to itself. A general architecture of a seq2seq model is shown in Figure 1 where the encoder model takes reference sequence $(x_0, x_1, \dots, x_t)$ and generates a context vector c_t. This context vector is then used in the decoder model to reproduce the target sequence $(y_0, y_1, \dots, y_t')$ one by one.

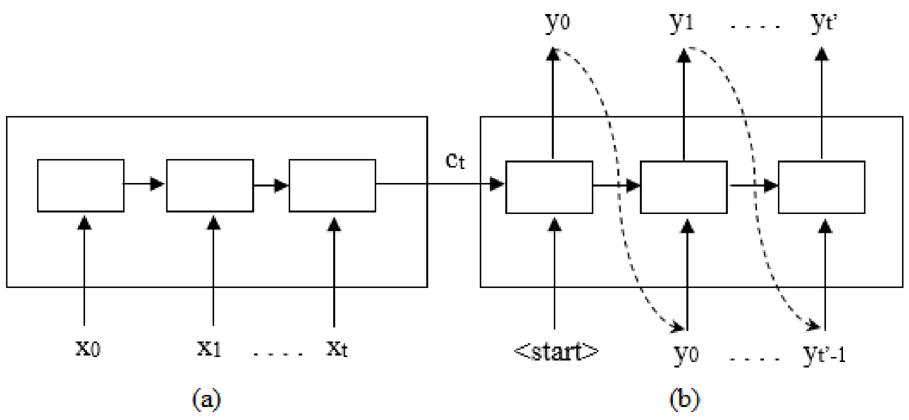


Figure 1. seq2seq Architecture (a) Encoder (b) Decoder

Most of these advances in literature and research works are confined to languages such as- English, Chinese, French, Spanish, etc. Bengali is the official and most widely spoken language of Bangladesh and the second most spoken regional language in India, behind Hindi. With almost 230 million speakers, Bengali is the seventh most spoken language in the world [5]. Even with such popularity, present NLP works for Bengali are very few in number, most of which are autoregressive language models.

This work is motivated by this lack of effectiveness of existing Bengali text generation based works. In particular, the issue has never been addressed for the language with the highly successful seq2seq recurrent networks. Besides, the ability to generate coherent and semantically meaningful text plays a key role in many NLP applications.

In this paper, we have presented context-based Bengali text generation using a conditional language model. The decoder model in seq2seq architecture is used here to produce the target sentence using a Bengali language model where the context for the decoder is also provided from the same language model.

We have used GloVe word embedding to map the words of every sentence to a vector space. The same representation is also used to generate context vectors. We have explored some ways of combining multiple context words into a fixed-length vector by determining its resemblance with the vector representation of a single word with a similar combined meaning. Then we used the Bengali Wikipedia dataset to train the baseline training model and used the weights from this network for inferring predictions given some seed words or phrases and the target context as input using a separate network. We have then used beam search to search for the most probable outcome. Finally, we proposed a scoring mechanism which is then used to analyze the comparative performance with different recurrent units and empirical results along with some sample generated texts are also presented.

Overall, the major contributions of this research work are:

1. We present an embellished overview of a successful work on context-driven text generation for the Bengali language on which no established work of research has been conducted.
2. We experiment and analyze several associated factors including the inspection of context vectors, NN architectures, and also optimization algorithms to find the most effective and apt combination. All these profound insights can be crucial for future works of research, especially on the Bengali language.
3. Finally, we propose a novel evaluation mechanism that involves real-world feedback for this kind of scenario, which is not possible to be evaluated with existing methods. According to this mechanism, on average, each generated sequence holds 70.86% of the expected properties.

2. Related Works

Research works on Natural language generation first came into light as the process of transforming non-linguistic machine representation of information into structured natural language. Although Recurrent Networks existed for a long time, these were extremely hard to train on sequential data with long-range temporal dependency due to vanishing or exploding gradient issues. In 2011 Sutskever et al. [6] proposed that combining standard RNNs with Hessian-Free (HF) optimizer mitigates the gradient issues. The authors referred to the model as multiplicative RNN (MRNN) due to its multiplicative nature of transforming the hidden state weight matrix into a function of the current input. The effectiveness of the proposed model is proven by building a large character level LM trained on three different datasets to predict the next characters given a sequence of characters as input.

Sundermeyer et al. presented an alternate solution of using the LSTM architecture for language modeling [7]. The gating mechanism in LSTM controls what information might contribute more to any present context. The method is applied to two LM tasks achieving 8% relative perplexity than standard RNNs and is also computationally more efficient.

A very notable work of Chinese poetry generation using RNNs was proposed by Zhang et al. The implementation takes some keywords for a poem as input and generates a set of all possible phrases containing the keywords. Among these, n phrases are selected using a tri-gram RNN LM with a stack decoder at character level to produce the first line of the poem. Then all the next lines of the poem are generated iteratively from previous lines using 3 different models (convolutional sentence model, recurrent context model, recurrent generation model) [8].

Cho et al. proposed an encoder-decoder architecture very similar to the seq2seq model, where the encoder RNN encodes an input sequence to a fixed length vector and the decoder produces an output sequence conditioned on the vector. The joined-model is trained using a gradient based algorithm to maximize the conditional log-likelihood [9].

Extending the applications of encoder-decoder model pairs, Jiwei Li et al. [10] introduced a neural autoencoder for paragraph and document generation. The encoder model generates a high-level embedding for paragraphs from low-level sentences and word embeddings, forming a hierarchical structure. The decoder model is then used to reconstruct the paragraphs from these embeddings and verifying through ROGUE and BLEU metrics it is shown how these encoded texts preserved the syntax, semantics, and coherence.

Wen et al. introduced an NLG application of dialogue generation conditioned on a context vector type and slot value pair [11]. They accomplished the task of integrating the context vector values by modifying the LSTM cell to have an additional gate to control the sentence planning. This solution is named semantically controlled LSTM (SC-LSTM). To preserve the coherence of words from both directions, they stacked two layers of SC-LSTM, one for each of forward and backward directions. They also presented human critics as evaluation.

The next revolution in NMT architectures came with the introduction of the Transformer model proposed by Vaswani et al. in 2017 [12]. The architecture completely replaces the use of recurrent or convolutional units with the attention mechanism, often used with the encoder-decoder model. As a result, the simpler network achieves tremendous computational efficiency as well as achieving state-of-the-art performance on several NLP tasks. Later, Devlin et al. proposed a powerful LM representation using the transformer architecture, referred to as the Bidirectional Encoder Representations from Transformers (BERT) in 2018. These representations were fascinating and outperformed all existing attempts on several natural language tasks [13].

In 2019, Egonmwan et al. used the seq2seq architecture for a work of Paraphrase generation [14]. The novelty of the work lied in combining the advantages of the Transformer model with the seq2seq architecture. The encoder consisted of two layers- the first one a Transformer model for rich language properties, and the second one a unidirectional recurrent unit. It used the GloVe embedding to represent input sequences. The framework performed extremely well in practice improving the state-of-the-art in two different paraphrasing datasets.

Santhanam (2020) introduced a context-based text generation using LSTM networks. This work demonstrated the use of context vectors in an NLG application, where the context vector effectively learns the semantic meaning of sentences. The author also inspected two different ways to calculate the context vector, namely word importance, and word clustering [15].

For text generation specific to the Bengali language, Sheikh Abujar et al. introduced a bi-directional RNN model in which hidden state transitions for every time step occur both in forward and backward direction [16]. The model used pre-trained embeddings for a custom dataset of Bengali texts. On experimentation, training accuracy for the model was very good but validation accuracy was not evaluated.

Most Bengali text generation tasks are based on RNNs with LSTM cells. In 2018 Sadidul et al. proposed an encoder-decoder based LSTM network that handled 3 types of error (missing words, misplaced words, and wrong arrangements). The model achieves good test set accuracy on a limited dataset [17]. Sanzidul et al. implemented a LSTM network with 100 nodes and a softmax activation function to generate output sequences of given length from a seed word [18].

Faruk et al. proposed an extended version of n-gram language models by introducing GRU based RNN on an n-gram dataset. The model is trained on a Bengali text corpus to iteratively predict the next word to form a complete sequence from an input sequence and experimentation shows that on average the model achieves better accuracy than LSTM based models [19].

3. Background and Preliminaries

3.1. Recurrent Neural Network variants

Due to the gradient issues, training RNNs with backpropagation through time (BPTT) is very difficult. Several gated variants of RNNs effectively eliminate these issues while producing robust models that are capable of maintaining long term dependencies. We have evaluated the performance of two of the most popular ones- LSTM and GRU for this work.

3.1.1. Long Short-Term Memory (LSTM) LSTM network was initially proposed by Hochreiter et al. in 1997 [20] as a remedy to the gradient issues of standard RNNs with many further improvements, but it gained the most popularity in sequence modeling very recently. LSTMs, in addition to the activation signal as in RNNs, maintain and propagate a high-level cell state in every time step. LSTMs employ three different gated units- an update gate, an output gate and a forget gate to determine, store and regulate the flow of information that holds more significance over long-range relations. The Figure 2a represents the architecture of a single LSTM unit.

The equations that govern a single LSTM cell (based on the implementation of Graves et al. 2013 [21]) are (1) - (6):

$$\tilde{c_t} = \tanh\left(W_c[c_{t-1}, a_{t-1}, x_t] + b_c\right) \tag{1}$$

$$u_t = \sigma(W_u[c_{t-1}, a_{t-1}, x_t] + b_u) \tag{2}$$

$$f_t = \sigma(W_f[c_{t-1}, a_{t-1}, x_t] + b_f) \tag{3}$$

$$o_t = \sigma(W_o[c_{t-1}, a_{t-1}, x_t] + b_o) \tag{4}$$

$$c_t = u_t * \tilde{c_t} + f_t * c_{t-1} \tag{5}$$

$$a_t = o_t * \tanh\left(c_t\right) \tag{6}$$

Here, u_t, f_t and o_t represents the update, forget and output gate at time step t respectively. c_t holds the cell state information and a_t is the activation information at time step t. Asterisk (*) denotes element-wise multiplication here.

3.1.2. Gated recurrent unit (GRU) GRU, proposed by Cho et al., is a much simpler version of LSTM. In contrast to LSTM, the cell state information at time step t is the same as the activation of that time step. Moreover, GRUs employ only two gates- the update gate and the relevance gate. Due to fewer gates, GRUs have less control over maintaining more important long-term information. But the lower number of gates also means that GRUs have a lower number of parameters to train, thus, making them easier and faster to train compared to LSTMs.

The GRU implementation we used for this paper is based on the work of Chung et al. [22] as in Figure 2b. The equations defining the implementation are (7) - (10):

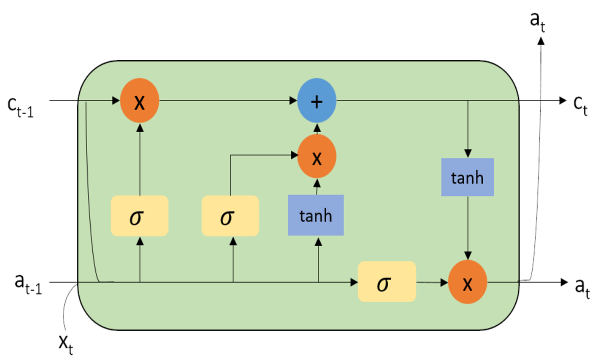
$$\tilde{c_t} = \tanh\left(W_c[r_t, a_{t-1}, x_t] + b_c\right) \tag{7}$$

$$u_t = \sigma(W_u[a_{t-1}, x_t] + b_u) \tag{8}$$

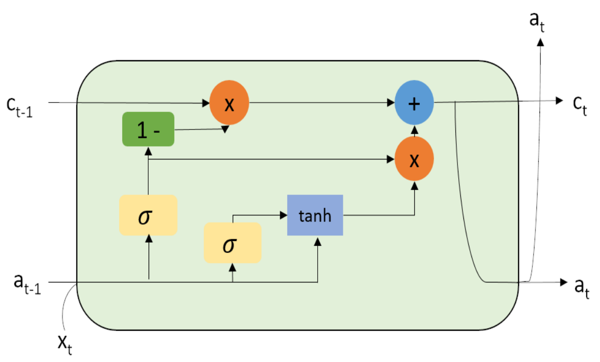
$$r_t = \sigma(W_r[a_{t-1}, x_t] + b_r) \tag{9}$$

$$c_t = u_t * \tilde{c_t} + (1 - u_t) * c_{t-1} \tag{10}$$

Here, only the update gate u_t determines how much information should be forgotten and how much should be updated. Relevance gate, r_t, is used to measure the relevance of the input at time step t over long-term dependency.



(a) LSTM cell



(b) GRU cell

Figure 2. Architecture of a single Recurrent Unit

3.2. Optimization Algorithms

The optimization of a stochastic objective function for a problem like a language modeling, which involves a large amount of data, is computationally very expensive. Moreover, as the parameters lie in high-dimensional spaces, it takes a lot of time to converge to the minima of the function using the standard gradient descent algorithm. In order to enable rapid convergence for problems with large numbers of data, and high-dimensional parameters several variants of gradient-based algorithms also with adaptive estimation of the learning rate are often used for machine learning problems. We have experimented few of the most frequently used optimization algorithms, which are discussed briefly in this section.

3.2.1. Stochastic Gradient Descent (SGD) In contrast to batch gradient descent, Stochastic gradient descent (SGD) algorithm iteratively calculates and applies gradient-based optimization for each training sample [23]. Although SGD increases the number of operations, with a small enough learning rate, convergence to a local minimum is almost certain. Moreover, for large training batches, progress can be made without calculating the accumulated cost for the whole training batch. For a function $f(\theta)$ with parameter θ, in each iteration of SGD, the approximation of gradient is given by (11).

$$\theta = \theta - \alpha \times \nabla_\theta f_t(\theta) \tag{11}$$

Here, $f_t(\theta)$ refers to the value of objective function for t^{th} training example, where t = $1, 2, \ldots, n$. $\nabla_\theta f_t(\theta)$ refers to the first-order gradient of the function and α is the learning rate.

3.2.2. RMSprop RMSprop algorithm, proposed by Hinton et al. [24], aims to incorporate adaptive learning rates with the mini-batch gradient descent algorithm. The algorithm keeps an exponentially weighted moving average of the squared gradients for each parameter and adaptively estimates the parameters by dividing by the square root of these values. As a result, for non-stationary objective functions, RMSprop performs exceedingly well by speeding the learning process as well as resulting in better convergence. Also, it reduces the memory requirements drastically. However, as the method puts no attempt on trying to correct bias terms, the initial approximations of the algorithm diverge from the actual data points. The moment vector $V(\theta, T)$ for the T^{th} mini-batch and parameter θ is calculated as (12).

$$V(\theta, T) = \gamma \times V(\theta, T-1) + (1-\gamma) \times (\nabla_\theta f(\theta, T))^2 \tag{12}$$

Here, γ is the decay rate and $\nabla_\theta f(\theta, T)$ denotes the first-order gradient of the objective function with the parameter θ at $T^t h$ iteration.

The gradient is used for optimization as shown in (13).

$$\theta = \theta - \alpha \times \frac{\nabla_\theta f(\theta, T)}{\sqrt{V(\theta, T)}} \tag{13}$$

3.2.3. Adagrad Adagrad or Adaptive gradient algorithm by Duchi et al. is a customized version of the SGD algorithm with adaptive learning rates for each parameter. In an online setting, the algorithm works very well, particularly with sparse gradients. The algorithm functions by scaling the learning rate with a scaling factor that corresponds to the sparsity of the data.

The update operation in Adagrad for parameter θ is given by (14).

$$\theta_{t+1} = \theta_t - \alpha \times \frac{\nabla_\theta f(\theta_t)}{\sqrt{\sum_1^t (\nabla_\theta f(\theta_t))^2}} \tag{14}$$

Here, θ_t denotes the parameter θ at time step t, where $t = 1, 2, \ldots, n$ and α is the global learning rate.

3.2.4. Adam Adam or Adaptive motion estimation presented by Kingma et al. is another very effective adaptive optimization method for stochastic cost functions. It is suitable for both stationary and non-stationary objective functions and also performs very well with noisy objectives [26]. In fact, the algorithm combines the advantages of momentum-based optimization with the RMSprop algorithm. Consequently, the algorithm results in rapid convergence with memory efficiency, and also as bias-correction is incorporated, the approximation is also very accurate. So, it is very well suited for many machine learning problems including NLP and Computer Vision.

The algorithm estimates two moments. These are as shown in (15), (16).

$$V_1(\theta,t) = \beta_1 \times V(\theta,t-1) + (1-\beta_1) \times \nabla_\theta f(\theta,t) \tag{15}$$

$$V_2(\theta,t) = \beta_2 \times V(\theta,t-1) + (1-\beta_2) \times (\nabla_\theta f(\theta,t))^2 \tag{16}$$

Here, β_1 and β_2 denote the two decay rates for exponential moment estimation for the moment vectors, V_1 and V_2 respectively. These moment vectors are then bias-corrected as in (17) and (18) respectively, before being used to optimize the parameter θ as in (19).

$$\tilde{V_1}(\theta,t) = \frac{V_1(\theta,t)}{(1-\beta_1^t)} \tag{17}$$

$$\tilde{V_2}(\theta,t) = \frac{V_2(\theta,t)}{(1-\beta_2^t)} \tag{18}$$

$$\theta_t = \theta_{t-1} - \alpha \times \frac{\tilde{V_1}(\theta,t)}{\sqrt{\tilde{V_2}(\theta,t)} + \epsilon} \tag{19}$$

Here, ϵ is an infinitesimal number used to prevent a possible division by zero when the denominator is very small and close to zero.

4. Proposed Framework

The proposed framework is based on a generative conditional language model, similar to the decoder of the autoencoder architecture. Let the total set of the vocabulary, $V = (w_0, w_1, \ldots, w_m)$. Given some input sequence from the vocabulary, the model uses a LM to transform this word representations into the vector space as $(x_0, x_1, \ldots, x_{t-1})$ and predicts the probability distribution over the set V, further conditioned by some context words to direct the overall meaning of the output sequence. This context is also provided as input to the model and transformed into a fixed dimensional vector c_t using the same LM. The output of the model is $p(x_t|c_t, x_0, x_1, \ldots, x_{t-1})$. This probability distribution is then used to determine the output sequence until the end tag or <eof> is found. A generic diagram depicting the workflow of the model is shown in Figure 3.

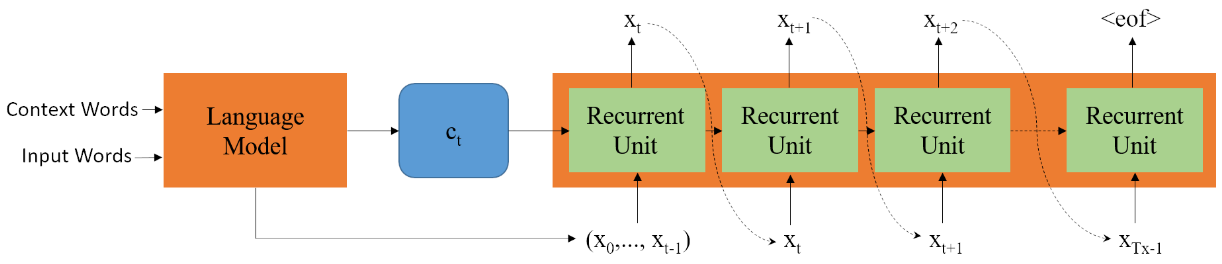


Figure 3. Context-driven Text Generation Model

4.1. *Word Embeddings: GloVe*

Word embedding is the representation of words into a fixed dimensional vector space. This mapping operation results in a vector in close proximity in the vector space for words with similar meaning. GloVe or Global vectors for word representation is used for this work. Proposed by Pennington et al., GloVe embeddings leverage usage of statistical information efficiently from a global corpus [27]. The model also produces meaningful substructures enabling arithmetic operations on these vectors to preserve semantic and syntactic regularities.

We have used pre-trained GloVe embeddings (from `https://github.com/sagorbrur/GloVe-Bengali`) trained on a vocabulary of size 178152 from the Bengali Wikipedia dataset. The word vectors are 300 dimensional. Each of these dimensions represents a different attribute for each word. Similarity or dissimilarity between two word vectors $(\vec{u}, \vec{v})$ can be determined by the cosine similarity given by (20) and an example of a cosine similarity cross matrix is given in Table 1 for words representing a common context and a diagram of GloVe Vector Visualization (800 most frequent words) using the t-SNE algorithm is shown in Figure 4.

$$\cos(\vec{u}, \vec{v}) = \frac{(\vec{u}, \vec{v})}{|\vec{u}|.|\vec{v}|} \tag{20}$$

Table 1. Cosine similarity cross matrix example

	বিজ্ঞান (Science)	কম্পিউটার (Computer)	প্রযুক্তি (Technology)	ডিজিটাল (Digital)	প্রকৌশল (Engineering)
বিজ্ঞান	1.00	0.46	0.60	0.25	0.53
কম্পিউটার	0.46	1.00	0.53	0.51	0.49
প্রযুক্তি	0.60	0.53	1.00	0.50	0.57
ডিজিটাল	0.25	0.51	0.50	1.00	0.24
প্রকৌশল	0.53	0.49	0.57	0.24	1.00

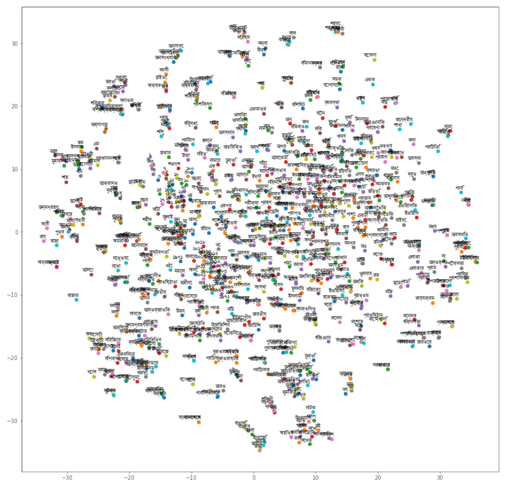


Figure 4. t-SNE Visualization of GloVe Vectors

The context vector, c_t, is passed to the model as the initial hidden state of the model and must be a single vector of a specific length. However, the model must also be able to interpret provided contexts using several combinations of words. So, one single vector representing a combination of all the context words must be found. As GloVe embedding preserves semantic information, we have compared different ways to combine the embedding vectors for each word in the context and evaluated the similarity with a single word that best describes the combination of all context words as in Table 2.

Table 2. Different techniques for combining multiple vectors

Context Words	Target Word	Cosine Similarity		
		Addition of Vectors	Multiplication of Vectors	Multiplication after adding each pair of Vectors
রাজধানী, মহানগরী, বাংলাদেশ (Capital, metropolis, Bangladesh)	ঢাকা (Dhaka)	0.46	0.06	0.05
সার, বীজ, সেচ, চারা (Fertilizers, seeds, irrigation, seedlings)	কৃষি (Agriculture)	0.35	-0.12	0.29
বজ্র, মেঘ (Lightning, clouds)	বৃষ্টি (Rain)	0.33	-0.05	0.33
পান্তা, ইলিশ, কালবৈশাখী (Panta, Hilsa, Kalbaishakhi)	বৈশাখ (First Bangla Month)	0.31	0.14	0.23

From the table, it can be seen that the sum of all the embedding vectors achieves almost 30% of similarity with the embedding vector of a single word that best defines the context, which is very decent considering the accuracy of the embeddings. So we have combined the context vectors using addition for this work and a visual representation of the effectiveness of this combination process is shown in Figure 5. Although a model with a dense layer can be used to provide a context vector with even better similarity, we have omitted this as it puts a limit on the number of context words, and obtaining training-testing pairs of data is also very tough.

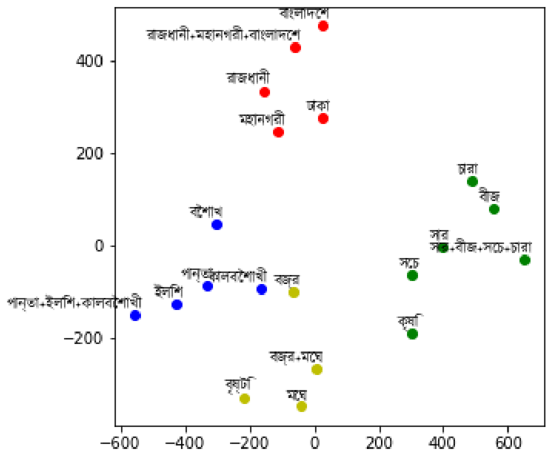
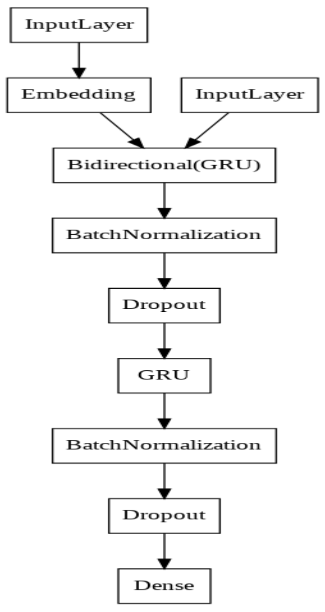


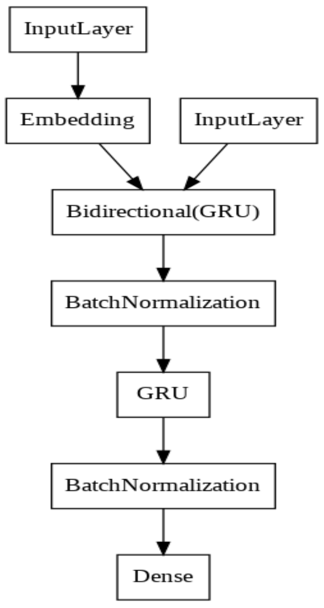
Figure 5. Visualizing addition of GloVe Vectors through t-SNE

4.2. *Neural Network Architecture*

The key strategy behind the work is to use two different networks for training and inferring as shown in Figure 6. The training network employs regularization and is used directly on the training data to train the model. The inference network, as suggested by the name, is used to infer new texts given some input.



(a) Training Model



(b) Inference Model

Figure 6. Proposed Model Architecture

4.2.1. Training Model The training model takes two inputs- an embedding vector representing the context and the n-gram training sequences from the dataset. As illustrated in Figure 6a, the network employs 3 different types of layers-

4.2.1.1 Bidirectional GRU

A bidirectional recurrent unit is a combination of two separate recurrent units where the hidden states flow in both directions for each unit ie- hidden state, $h = [\vec{h}, \overleftarrow{h}]$. Thus bidirectional recurrent units consider inputs from both the past and the future. The advantage of using bidirectional GRUs is that they can capture the context either from the end or from the beginning of the sentence.

The first GRU layer in the network is a bidirectional layer and every time step provides an output. The other GRU layer is unidirectional and only the final time step provides an output. In order to keep the shape of the hidden state consistent with the shape of context vectors, the number of GRU units is equal to the embedding dimensions.

4.2.1.2 Batch Normalization

The batch normalization layer normalizes the activation from the previous layer by subtracting the batch mean and dividing by the batch standard deviation. Then these zero mean/zero variation activations are rescaled and shifted to follow some random distribution for every mini-batch. Thus helping to reduce covariance shift and making the training process more stable.

4.2.1.3 Dropout

Dropout is a regularization technique. We have used two 20% dropout layers in the training model.

4.2.1.4 Softmax Layer

The final layer in the training network is a softmax layer with a number of units equal to the vocabulary size. The layer uses softmax activation to output a probability distribution over the complete vocabulary where the class with maximum probability denotes the expected output. The softmax activation is given by the equation (21).

$$\text{Softmax Activation}, S(z_i) = \frac{e^{z_i}}{\sum_k e^{z_j}} \tag{21}$$

4.2.2. Inference Model The inference model, as the training model, also takes some context vector as input. There might be some input sequences as well. The inference model predicts the rest of the words in the sequence conditioned by the context. The model has the bidirectional GRU, the GRU, Batch Normalization layers, and the final softmax layer, where it acquires the weights for each of these layers from the trained model. As output, apart from the softmax distribution, the model also outputs the hidden state of the bidirectional GRU layer for the prediction of the next time step. Dropout regularization is opted out, as during inference, this would create random noise in prediction. The model is depicted in Figure 6b.

4.3. Training

Britz et al. presented a massive exploratory analysis of hyperparameters in NMT architectures [28]. Because the model used in this paper is also based on the NMT architecture, most of the hyperparameters required for training are directly taken from the analysis of Britz et al. Several other factors involved in training the model are discussed in this section.

4.3.1. Usage of Context Vectors during training The dataset does not contain any separate column for the context words. So from each sentence, we randomly picked an arbitrary number of words without replacement and used them as the context vector for all the n-gram sequences produced from that particular sentence. The number of context words can range from a minimum of one word to a maximum of the length of the sentence. In order to combine multiple context words, we follow the method described in an earlier section which is represented by (22).

$$\text{Context Vector}, c_t = \sum_{n}^{k} c_n \tag{22}$$

where $0 < K \leq$ length of sentence

So a single context vector is of the same shape as the embedding dimensionality.

4.3.2. Teacher Forcing The model is trained using the teacher forcing method. This method enables efficient training of RNNs with faster convergence. Generally during training, the output from a recurrent unit at time step t, $\hat{y}_t$, is fed back to the next time step as input, so $x_{t+1} = \hat{y}_t$. But in teacher forcing $\hat{y}_t$ is calculated but the ground truth at time step t, y_t, from the training data is fed as the input to the next time step. Thus forcing the model to learn based on the ground truth sequences.

4.3.3. Regularization Regularization is a technique applied to reduce the risk of the model overfitting the training data. Overfitting causes learning complex approximations of the training data, thus affecting the accuracy of data it has never seen. We have used dropout regularization in this paper. The dropout method randomly shuts off nodes in a layer. Thus shrinking the network, preventing it from being able to learn more complex functions on the data. Here we have used 20% dropout as per the requirements of this particular application.

4.3.4. Loss Function As the prediction of the model is for categorical data with a softmax output, the loss function used here is the categorical cross-entropy. The cross-entropy loss is given by the formula (23).

$$\mathcal{L}(y, \hat{y}) = -\sum_i y_i \log(\hat{y_i}) \tag{23}$$

4.3.5. Optimization In order to optimize the objective function, we have used the Adam algorithm. The motivation behind using the algorithm is the stochasticity of the objective function of the problem at hand and also the robustness to noises. Because the function induces noise through dropout regularization. Besides, it results in faster convergence to a global minimum at an exceptionally low computational cost. The advantage can also be observed empirically, as it outperforms the SGD, Adagrad, and RMSprop algorithms while resulting in a plunge in terms of loss after 20 training iterations for the whole training batch. The comparison among these optimization algorithms is depicted in Figure 7.

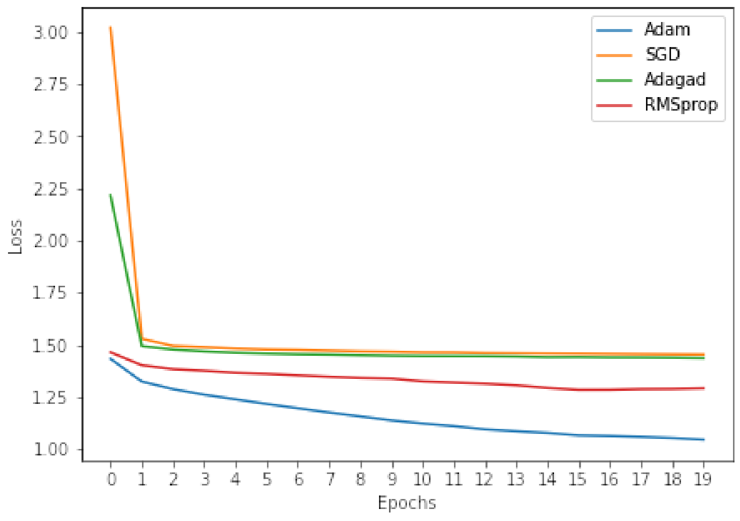


Figure 7. Comparison of optimization algorithms

We have used Adam with a fixed learning rate of 0.001. (The other parameters- $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 1 \times 10^{-8}$). The training accuracy and loss per epoch for the target vocabulary size, after 100 iterations, is shown in Figure 8.

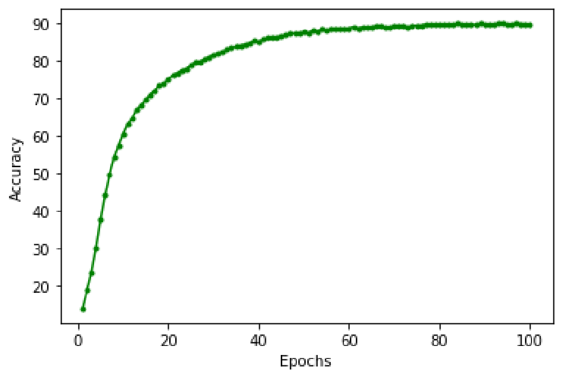
5. Experiment

We have carried out experimentation on the proposed framework using a single dataset. Various steps involved with the experiment, along with a comprehensive evaluation of the outcomes, are demonstrated in this section.

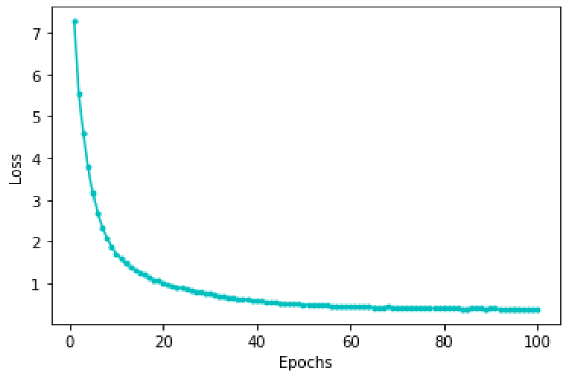
5.1. Experimental Setup

5.1.1. Dataset We have used the Bengali Wikipedia dataset, which is extracted from the raw dump of Wikipedia's Bengali version [29] and contains four columns (id, text, title, url). Among the columns, we have only used the text column that contains a total of 70377 articles. Different characteristics of the dataset are presented in Table 3.

5.1.2. Preprocessing The dataset is preprocessed by first stripping all the punctuations and characters from other languages. All the stopwords are also removed. Stopwords are the words that appear more frequently in a language but are not very significant towards the meaning. We have considered a total of 398 Bengali stopwords (from



(a)



(b)

Figure 8. Training (a) accuracy and (b) loss per epoch

Table 3. Properties of the Bengali Wikipedia dataset

Number of Samples	70377
Total Words	13131206
Unique Words	696876
Total occurrence of stopwords	4084078
Number of sentences	1244973
Average Length of sentences	11
Longest sentence	1893
Shortest sentence	2

https://github.com/stopwords-iso/stopwords-bn) which, from the table, can be seen constitutes almost 31.10% of the total words. After cleaning the dataset, we have used the most frequently appearing first 20000 words as the vocabulary, hoping these sufficiently represent the richness of Bengali vocabulary. Along with the 20000 words, the vocabulary contains 3 special tags- <start>, <eof> marking the start and end of a sequence respectively, and <unk> tag to denote words not present in the vocabulary.

5.1.2.1 Tokenization

As the neural network can not associate any meaning to textual data, all the words in the vocabulary are required to be mapped onto a unique integer number. This process is called tokenization. Every sentence in the dataset is transformed into word sequences and each sequence is preceded with the <start> tag and ends with a <eof> tag. These word sequences are then replaced with their corresponding tokens, converting the texts into vectors. Zero is a reserved token for padding.

5.1.2.2 n-gram sequences

To prepare the training data, n-gram sequences are formed from the word sequence vectors. But due to limitations in the available computational resources we have only taken 100,000 training sequences. Each training example $X_i = (x_0, x_1, \ldots, x_t)$ and $Y_i = x_{t+1}$ is produced by incrementing the time marker t by one until the <eof> token is reached for every word sequence vector $(X_0, X_1, \ldots, X_n)$. The Y vector is one-hot encoded for faster training and better prediction.

5.1.2.3 Padding

RNNs can not work with inputs of arbitrary length. Thus sequences with a maximum length of 20, including special tags, are used for this paper. All the sequences in training data shorter than the maximum length are left padded with zeros. Left padding produced significantly better results than padding to the right on our experimentation. And longer sequences are truncated to the maximum length also from the beginning. This procedure results in a total of 1109386 training examples for the algorithm, with X of shape (1109386, 20) and Y of shape (1109386, 2003).

5.2. Prediction and Evaluation

5.2.1. Beam Search During inference, instead of using greedy search to find the most probable outcome at each time step, we use a better strategy called beam search. The idea behind beam search is that there might be some better sequence with more cumulative probability but one or a few of its words might not have the maximum predicted probability. Beam search solves this problem by considering k (beam width) most probable outputs during each prediction step. All these k predictions are combined with the initial input and used in the next step of prediction. The same process is followed until a complete sequence is predicted. The k elements that maximize the scoring function (24) are considered at each step.

$$S(y, c_t) = -\arg\max \frac{1}{K^{\alpha}} \sum_{i}^{K} \log \Pr(y_t | c_t, y_0, y_1, \ldots, y_{t-1}) \tag{24}$$

Where α is a length penalty operator. As Britz et al. showed that larger beam widths are not very effective, we continued with a beam width of 10 and a length penalty of 1.

5.2.2. Evaluation Metric: Human Scoring As there is no standard metric to be able to evaluate the quality and coherency of the generated text, we have collected human evaluations on some sample generated outputs and averaged them. This human evaluation is based on three different criteria each having different weights so that scores are in the range [0, 1]. These criteria and their associated weight is presented in Table 4 and the score for the model is determined using formula (25).

$$Score = \frac{1}{K} \sum_{K} (0.4 \times \varphi_{1,i} + 0.4 \times \varphi_{2,i} + 0.2 \times \varphi_{3,i}) \tag{25}$$

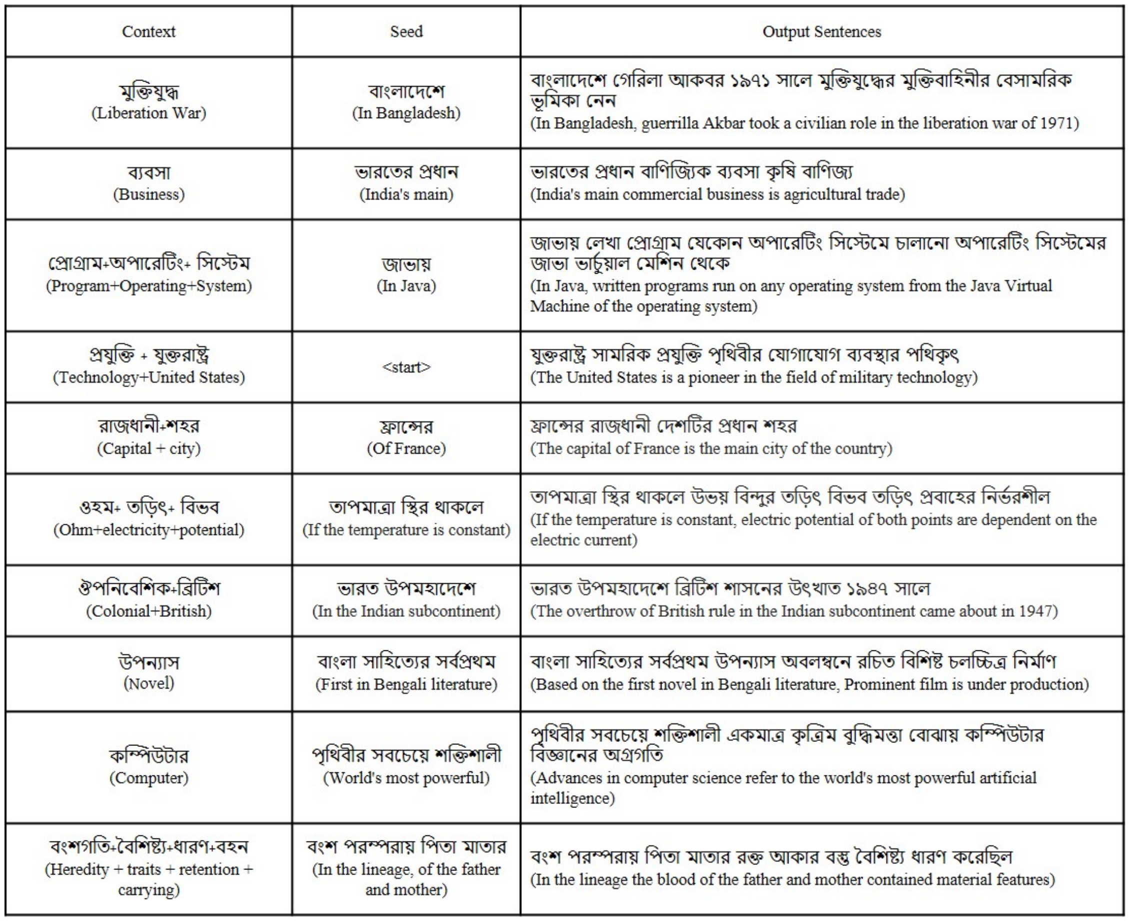
Table 4. Human Evaluation Criteria

Criteria Name	Symbol	Weight
1. Effectiveness of the sentence (whether it is syntactically and semantically correct)	φ_1	0.4
2. The representation of the given context	φ_2	0.4
3. Relevance of the sentence for user application	φ_3	0.2

5.3. Experimental Results

The training model is trained and optimized first with 100 epochs over the full training data. In order to make the model more robust on a larger set of contexts, we generated new context vectors every 10 epochs. Then the weight matrices from the training network are used in the inference model for inferring. The complete output sequence is then produced using the beam search mechanism. The Table 5 presents experimental results for different input word sequences and different combinations of context words.

Table 5. Example outcomes with English Translations



Context	Seed	Output Sentences
মুক্তিযুদ্ধ (Liberation War)	বাংলাদেশে (In Bangladesh)	বাংলাদেশে গেরিলা আকবর ১৯৭১ সালে মুক্তিযুদ্ধের মুক্তিবাহিনীর বেসামরিক ভূমিকা নেন (In Bangladesh, guerrilla Akbar took a civilian role in the liberation war of 1971)
ব্যবসা (Business)	ভারতের প্রধান (India's main)	ভারতের প্রধান বাণিজ্যিক ব্যবসা কৃষি বাণিজ্য (India's main commercial business is agricultural trade)
প্রোগ্রাম+অপারেটিং+ সিস্টেম (Program+Operating+System)	জাভায় (In Java)	জাভায় লেখা প্রোগ্রাম যেকোন অপারেটিং সিস্টেমে চালানো অপারেটিং সিস্টেমের জাভা ভার্চুয়াল মেশিন থেকে (In Java, written programs run on any operating system from the Java Virtual Machine of the operating system)
প্রযুক্তি + যুক্তরাষ্ট্র (Technology+United States)	<start>	যুক্তরাষ্ট্র সামরিক প্রযুক্তি পৃথিবীর যোগাযোগ ব্যবস্থার পথিকৃৎ (The United States is a pioneer in the field of military technology)
রাজধানী+শহর (Capital + city)	ফ্রান্সের (Of France)	ফ্রান্সের রাজধানী দেশটির প্রধান শহর (The capital of France is the main city of the country)
ওহম+ তড়িৎ+ বিভব (Ohm+electricity+potential)	তাপমাত্রা স্থির থাকলে (If the temperature is constant)	তাপমাত্রা স্থির থাকলে উভয় বিন্দুর তড়িৎ বিভব তড়িৎ প্রবাহের নির্ভরশীল (If the temperature is constant, electric potential of both points are dependent on the electric current)
ঔপনিবেশিক+ব্রিটিশ (Colonial+British)	ভারত উপমহাদেশে (In the Indian subcontinent)	ভারত উপমহাদেশে ব্রিটিশ শাসনের উৎখাত ১৯৪৭ সালে (The overthrow of British rule in the Indian subcontinent came about in 1947)
উপন্যাস (Novel)	বাংলা সাহিত্যের সর্বপ্রথম (First in Bengali literature)	বাংলা সাহিত্যের সর্বপ্রথম উপন্যাস অবলম্বনে রচিত বিশিষ্ট চলচ্চিত্র নির্মাণ (Based on the first novel in Bengali literature, Prominent film is under production)
কম্পিউটার (Computer)	পৃথিবীর সবচেয়ে শক্তিশালী (World's most powerful)	পৃথিবীর সবচেয়ে শক্তিশালী একমাত্র কৃত্রিম বুদ্ধিমত্তা বোঝায় কম্পিউটার বিজ্ঞানের অগ্রগতি (Advances in computer science refer to the world's most powerful artificial intelligence)
বংশগতি+বৈশিষ্ট্য+ধারণ+বহন (Heredity + traits + retention + carrying)	বংশ পরম্পরায় পিতা মাতার (In the lineage, of the father and mother)	বংশ পরম্পরায় পিতা মাতার রক্ত আকার বস্তু বৈশিষ্ট্য ধারণ করেছিল (In the lineage the blood of the father and mother contained material features)

From the results, it can be observed that the produced sentences include almost no syntactic or semantic mistakes. Even with shorter training time, the network learns precise sentence representations for the LM. The outputs also carry the provided context accurately, mainly by trying to find all the context words themselves or similar words in the sentence. However, the outcomes are more relevant for more frequently occurring topics.

5.3.1. Comparative Analysis We compared the performance on the dataset for 3 different architectures using the human scoring metric by collecting scores from volunteers on the outputs generated by each architecture. The bidirectional GRU architecture is the baseline model proposed in the paper. The GRU model only replaces the bidirectional layer with unidirectional GRU units. The final architecture substitutes GRU units with LSTM cells for both recurrent layers. The comparative performance is shown in Table 6.

Table 6. Empirical Results for 3 different networks

Architecture	φ_1 Score	φ_2 Score	φ_3 Score	Final Score
Bidirectional GRU	77.86%	73.16%	56.06%	71.62%
GRU	72.55%	70.44%	55.76%	68.34%
LSTM	78.31%	61.04%	75.61%	70.86%

Between the two recurrent units, LSTMs perform a lot better in learning syntax and semantics representation and also produce more useful sentences. However, LSTMs struggle in preserving and portraying the context information through hidden states.

In contrast to the networks with unidirectional recurrent layers, the baseline bidirectional network achieves a better overall score. Bidirectional flow of the hidden state vectors helps significantly in depicting the context as well as in correct sentence formulation.

6. Conclusion and Future Work

We have proposed a context-driven text generation work in this paper that uses the decoder network from the seq2seq model often used in NMT applications. The network is conditioned on the context vectors drawn from the same GloVe language model which is used for embedding the words of each sequence into a vector space. Two separate networks are used for training and inferring. Beam search optimization was used to generate the sequence with the maximum log probability score.

In addition, we have proposed an evaluation technique involving human scoring on different criteria of the outcomes. We have then compared the performance of different recurrent architectures where the baseline model proves to be very effective in producing meaningful outcomes characterized by the target context.

The success of the work presented in the paper, even with short training time and limited data, lays the foundation for numerous other natural language generation based applications where the target sequence is required to hold preferred contextual information. Also, this is the first of a kind work for the Bengali language, which contributes to the very limited literature of the language.

Although the proposed work very effectively meets its objective of containing the target context, it tends to do so by trying to include that word in the generated sentence. So for future work, we will consider other more improved language model representations such as- ELMo [30], which would be able to associate the meaning of the words also while producing contextualized word embeddings.

REFERENCES

1. N. M. Goldman, *Computer generation of natural language from a deep conceptual base*, tech. rep., Stanford University-Computer Science Department, 1974.
2. E. Hovy, G. van Noord, G. Neumann, and J. Bateman, *Language generation*, Survey of the State of the art in Human Language Technology, pp. 131-146, 1996.
3. E. Reiter and R. Dale, *Building natural language generation systems*, Cambridge university press, 2000.
4. I. Sutskever, O. Vinyals, and Q. V. Le, *Sequence to sequence learning with neural networks*, Advances in neural information processing systems, 2014, pp. 3104-3112.
5. Wikipedia, *Bengali language*, https://en.wikipedia.org/wiki/Bengali_language [Online; accessed 20-September-2020].
6. I. Sutskever, J. Martens, and G. E. Hinton, *Generating text with recurrent neural networks*, ICML, 2011
7. M. Sundermeyer, R. Schlter, and H. Ney, *LSTM neural networks for language modeling*, Thirteenth annual conference of the international speech communication association, 2012.
8. X. Zhang and M. Lapata, *Chinese poetry generation with recurrent neural networks*, Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2014, pp. 670-680.
9. K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, *Learning phrase representations using RNN encoder-decoder for statistical machine translation*, arXiv preprint arXiv:1406.1078, 2014.
10. J. Li, M.-T. Luong, and D. Jurafsky, *A hierarchical neural autoencoder for paragraphs and documents*, arXiv preprint arXiv:1506.01057, 2015.
11. T.-H. Wen, M. Gasic, N. Mrksic, P.-H. Su, D. Vandyke, and S. Young, *Semantically conditioned lstm-based natural language generation for spoken dialogue systems*, arXiv preprint arXiv:1508.01745, 2015.
12. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, *Attention is all you need*, arXiv preprint arXiv:1706.03762, 2017.
13. J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *Bert: Pre-training of deep bidirectional transformers for language understanding*, arXiv preprint arXiv:1810.04805, 2018.
14. E. Egonmwan and Y. Chali, *Transformer and seq2seq model for paraphrase generation*, Proceedings of the 3rd Workshop on Neural Generation and Translation, pp. 249-255, 2019.
15. S. Santhanam, *Context based text-generation using lstm networks*, arXiv preprint arXiv:2005.00048, 2020.
16. S. Abujar, A. K. M. Masum, S. M. H. Chowdhury, M. Hasan, and S. A. Hossain, *Bengali Text generation Using Bi-directional RNN*, 2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT), 2019, pp. 1-5.

17. S. Islam, M. F. Sarkar, T. Hussain, M. M. Hasan, D. M. Farid, and S. Shatabda, *Bangla Sentence Correction Using Deep Neural Network Based Sequence to Sequence Learning*, 2018 21st International Conference of Computer and Information Technology (ICCIT), 2018, pp. 1-6.
18. M. S. Islam, S. S. S. Mousumi, S. Abujar, and S. A. Hossain, *Sequence-to-sequence Bangla sentence generation with LSTM recurrent neural networks*, Procedia Computer Science, vol. 152, pp. 51-58, 2019.
19. O. Rakib, S. Akter, M. Khan, A. Das, and K. Habibullah, *Bangla Word Prediction and Sentence Completion Using GRU: An Extended Version of RNN on N-gram Language Model*, 2019 International Conference on Sustainable Technologies for Industry 4.0 (STI), 2019, pp. 1-6, doi: 10.1109/STI47673.2019.9068063.
20. S. Hochreiter and J. Schmidhuber, *Long short-term memory*, Neural computation, vol. 9, no. 8, pp. 1735-1780, 1997.
21. A. Graves, *Generating sequences with recurrent neural networks*, arXiv preprint arXiv:1308.0850, 2013.
22. J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, *Empirical evaluation of gated recurrent neural networks on sequence modeling*, arXiv preprint arXiv:1412.3555, 2014.
23. L. Bottou, *Stochastic gradient descent tricks*, Neural networks: Tricks of the trade, pp. 421-436, Springer, 2012.
24. T. Tieleman and G. Hinton, *Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude*, COURSERA: Neural networks for machine learning, vol. 4, no. 2, pp. 26-31, 2012.
25. J. Duchi, E. Hazan, and Y. Singer, *Adaptive subgradient methods for online learning and stochasticoptimization*, Journal of machine learning research, vol. 12, no. 7, 2011.
26. D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, arXiv preprint arXiv:1412.6980, 2014.
27. J. Pennington, R. Socher, and C. D. Manning, *Glove: Global vectors for word representation*, Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), 2014, pp. 1532-1543.
28. D. Britz, A. Goldie, M.-T. Luong, and Q. Le, *Massive exploration of neural machine translation architectures*, arXiv preprint arXiv:1703.03906, 2017.
29. W. Foundation, *Wikimedia Database backup dumps*, https://dumps.wikimedia.org/bnwiki/latest/ [Online; accessed 12-August-2020].
30. M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, *Deep contextualized word representations*, arXiv preprint arXiv:1802.05365, 2018.