Dzongkha Next Words Prediction Using Bidirectional LSTM

Karma Wangchuk, Tandin Wangchuk and Tenzin Namgyel

DOI: https://doi.org/10.17102/bjrd.rub.se2.038

Abstract

Dzongkha Development Commission of Bhutan (DDC) is trying to computerize Dzongkha. However, the computerization of Dzongkha poses numerous challenges. Currently, the support for Dzongkha in modern technology is limited to printing, typing, and storage. Typewriting a single Dzongkha word requires several keypresses. As a result, typing Dzongkha is tedious. In this paper, the Dzongkha word label prediction was studied. The purpose of the study was to further reduce keystrokes and make Dzongkha typing much faster. The dataset encompasses different genres curated by DDC. The dataset consisted of 10000 sentences and 4820 unique words. Next, 52150 sequences were generated using N-gram methods followed by vectorizing text using embedding techniques. Different RNN-based models were evaluated for the next Dzongkha words prediction. Two Bi-LSTM layers with 512 hidden layer neurons gave the best accuracy of 73.89% with a loss of 1.0722.

Keywords – *Dzongkha word prediction, Machine Learning, Recurrent Neural Network, Long Short-Term Memory, Bidirectional LSTM*

Introduction

Typewriting, exchanging texts, and documentation have been easy with the advancement of technologies. However, Dzongkha typing is difficult (Wangchuk et al., 2021). Dzongkha is the national language of Bhutan. According to Norbu and Namgyel (2019), Dzongkha is native to eight western districts of Bhutan and is now a lingua franca. Dzongkha language has consonants and vowels similar to Tibetan scripts. The *Uchen* and *Ume* scripts were developed by Thonmi Sambhota in the seventh century in Tibet. Furthermore, the *Joyig* script was developed by Demang Tsemang in the eighth century (Norbu & Namgyel, 2019). *Joyig* is used by Bhutanese that is unique from Tibetan scripts. However, the number of consonants and vowels is the same.

Dzongkha is categorized as a Sino-Tibetan language (Chungku et al., 2010). Scripts are written continuously and have no word separator. This poses several challenges in the digitization of the language. Dzongkha has 30 consonants and four vowels (Tshering & Van Driem, 2019). It is a syllabic similar to South East Asian languages such as Hindi and Sanskrit (S. Norbu et al., 2010). According to Norbu and Namgyel (2019), a single syllable can have one to seven characters stacked on one another. Characters such as ད, ས, ག, ར, ཨ, ད, and ས stacked to form བསྒྲུབས (completed). Similarly, a word can have one syllable ཟས་ (food) or multiple syllables དགའ་རྟོག་རྟོ (happy) separated by a dot called 'tsheg'. A single syllable རྒྱལ (win) and ལས (needle) combined together རྒྱལ་ལས (country) can form a new word that has a completely different meaning. Similarly, syllables are concatenated མཛའ་རོགས་དགའ་རོག (lover) with dots to form a word. There is no word delimiter such as space in English. For example: རྫོང་ཁ་སྐད་སྦྱང་གི་དབེ་དེབ་ཀུ་འཛིན་ནི་དང་བར་སྐྱན་འབང་ཞིའི་ངོ་ཤེས་རིག་སྐྱན་ལག་ཏོག་ལ་གོང་འཕེལ་སྟེ་ཚན་ཨིག་ཡང་གཞི་བསྒོགས་གནང་སྟེ་སློབ་གྲྭ་ཁག་ལས་ཚང་རྫོང་ཁ་འདི་སྐད་ཡིག་དང་རྩོམ་རིག་གི་གྲས་ལང་སློབ་སྟོན་གྱི་ཚོས་ཚན་ཨིག་སྟེ་འགོ་བསྒོགས་གནང་ཏུག། As a result, it is difficult to identify words (Dhungyel & Grundspenkis, 2017) and painstaking to develop NLP applications. The absence of natural language processing tools causes a digital divide and may lead to the extinction of the national language.

To facilitate the use of Dzongkha in day-to-day communication and solve the digital divide in the country, it is important to develop technology to process Dzongkha. Not much work has been done to computerize Dzongkha. However, the baseline for the Dzongkha word segmentation was studied using maximal matching followed by bigram techniques (S. Norbu et al., 2010). They prepared text corpora from different genres such as newspaper articles, printed books, history, poetry, songs, and the Dzongkha dictionary. The dictionary-based approach was applied to the corpora for the selection of the segmentation that yielded the minimum number of word tokens from all possible segmentations of the input sentence. Similarly, Dzongkha words are segmented using the maximal matching algorithm (Dhungyel & Grundspenkis, 2017). The probabilistic approach was implemented to select the valid segmented words. The author had used a POS tagged corpus prepared by the Dzongkha Development Commission

of Bhutan (DDC). However, Jamtsho and Muneesawang (2020) studied Dzongkha word segmentation using deep learning. They used the syllable-tagged corpus curated by DDC. Their model achieved the highest F-score of 94.40%, 94.47% precision, and 94.35% recall. Furthermore, Wangchuk et al. (2021) predicted the next Dzongkha syllables using LSTM. The single-layer LSTM model with 128 cells outperformed other models with the best training accuracy of 78.33%. However, the study has some limitations. The proposed system reduces keypresses which makes Dzongkha typing faster and easier but still requires a greater number of syllables selection.

This study aims to propose the next word prediction in Dzongkha using Bi-LSTM which is one of the variants of the Recurrent Neural Network. The paper is organized as follows. The literature reviews have been discussed in Section 2 followed by a material and methods explanation in Section 3. Section 4 explains experimental results followed by a conclusion in Section 5.

Related Works

An artificial Neural Network (ANN) learns any nonlinear function and is known as the universal function approximator (Balkin & Ord, 2000). ANN can be used to solve problems with text data. However, Deep Neural Networks lead to vanishing gradients and exploding gradient problems when weights are updated in the backpropagation (MacDonald et al., 2021). Moreover, ANNs do not capture the temporal information required for the sequential data. The Recurrent Neural Network (RNN) is a variant of a Neural Network which is capable of modeling time-series data or Natural Language Processing (NLP) applications. RNN captures both spatial and temporal information in the sequential data. However, it suffers from long-term dependencies to process longer sequences. The Gated Recurrent Unit (GRU) and LSTM are variants of RNN that are designed to solve long-term dependency issues. These algorithms are used for longer sequential data and the development of NLP applications.

Next word prediction is one of the highly studied topics of the NLP domain. LSTM and Markov's chain were implemented for next-

word prediction in the Ukrainian language (Shakhovska et al., 2021). Stremmel and Singh (2021) used the LSTM-RNN language model implemented by Jing and Xu (2019) for the next word prediction. Sharma et al. (2019) proposed a novel methodology to predict the next word in a Hindi sentence using LSTM and Bi-LSTM. Similarly, LSTM was used for Assamese word prediction (Barman & Boruah, 2018). Furthermore, the RNN language model was used to predict the next character (Shi et al., 2017). In the following sections, RNN, LSTM, and Bi-LSTM are discussed.

Recurrent Neural Network

The traditional neural network assumes that the input and the output are independent. This assumption does not capture timestep information of the sequential data. However, the RNN is the variant of a neural network that deals with time-series data to capture temporal information. The RNN has a memory that remembers previous steps (Pedamkar, 2020). It takes information from prior inputs to affect the current input and output. RNN has a repeating module such as a single *tanh* layer that allows information to persist throughout the networks as shown in Figure 1. The neural network *N* takes previous state output x_{t-1} and the current input x_t to generate output h_t. The loop passes information from one timestep of the network *N* to the next.

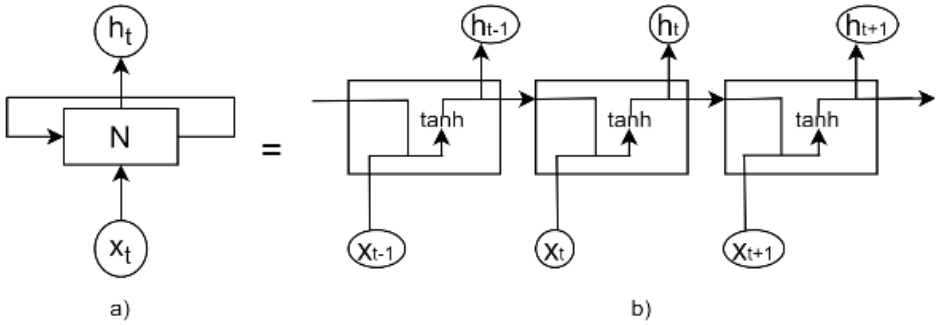

Fig.1 *Recurrent neural network architecture (Olah, 2015): a) Generic RNN architecture, b) An unfolded RNN.*

$$h_t = f(h_{t-1}, x_t) \quad (1)$$

$$h_t = tanh(w_{hh} * h_{t-1} + w_{xh} * x_t) \quad (2)$$

$$Y_t = (w_{hy}, h_t) \quad (3)$$

The current state is calculated by Eq. (1) and passed to the activation function that is defined by Eq. (2) where w_{hh} is the weight of the neuron and w_{xh} is the weight of the input neuron. The output of each time step is defined by Eq. (3) where w_{hy} is the weight at the output layer and Y_t is the output of the RNN.

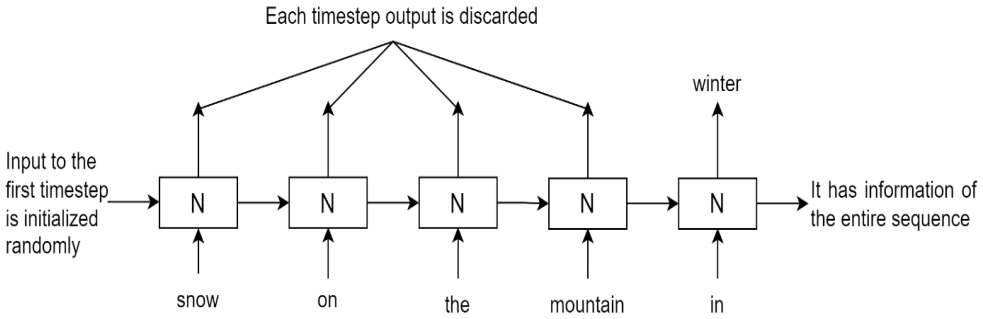

Fig. 2 *Recurrent neural network architecture for word prediction*

Figure 2 shows an unrolled recurrent neural network for the next word prediction. The number of neural networks *N* is linked similar to the chain-like structure that relates to lists and sequences to retain timestep information. The output of the previous step is randomly initialized and fed with the input of the current state to predict the next timestep. To predict the last word in *"snow on the mountain in winter"*, the previous words and context are required. Each word is fed into the RNN network at a different timestep and respective output is generated. However, output from each timestep is discarded and the last output has been considered to predict *"winter"* from the given sequence as shown in Figure 2. The hidden layer *N* of the RNN remembers information about the sequence and stores them in the memory. In theory, RNN can remember any sequence length but in implementation, it suffers from long-term dependencies (Koutnik et al., 2014). The prediction of the next words in a longer sentence required more contextual information. The limitation of the RNN is solve by LSTM.

Long Short-Term Memory

Long Short-Term Memory (LSTM) networks were designed and introduced by Hochreiter & Schmidhuber (1997) to address the short-term memory of the RNN. LSTM is capable of learning long sequences. Over the years, LSTMs were refined, popularized, and have been used for solving various problems (Koutnik et al., 2014; Cho & Merriënboer, 2014; Yao et al., 2015; Jozefowicz et al., 2015). According to Olah (2015), the default behavior of the LSTM is to remember the information for a longer period. LSTM architecture is similar to RNN. However, it has four different repeating layers as shown in Figure 3. The key factor for LSTM's success is the cell state c_{t-1}. The cell state runs straight from left to right (c_{t-1} to c_t) across the entire chain analogous to the conveyor belt (Olah, 2015). The information flows through the cell state unchanged. Moreover, it allows either to add or remove information controlled by gates. There are three gates namely input gate i_t, forget gate f_t, and output gate o_t composed of the *sigmoid* layer. The *sigmoid* layer output is either zero (stop everything) or one (pass everything) deciding on how much information should be added or removed.

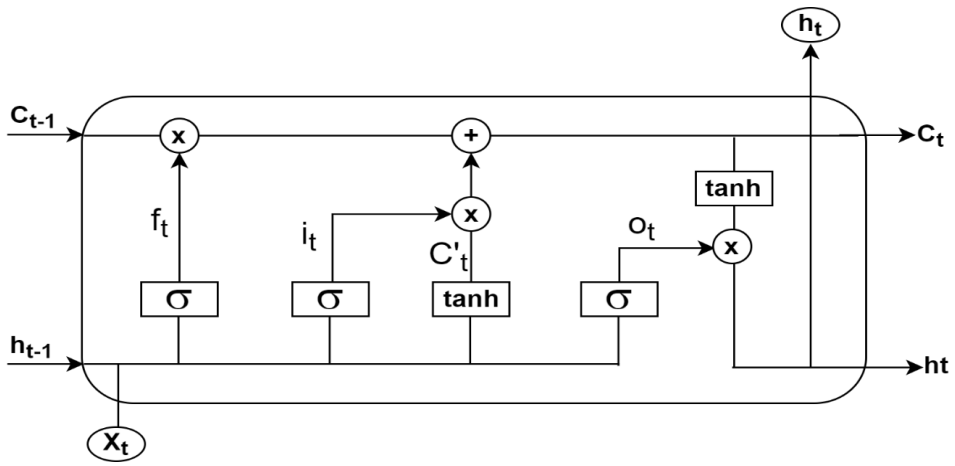

Fig. 3 Long Short-Term Memory layer architecture

$$f_t = \sigma(w_f[h_{t-1}, x_t]+b_f) \quad (4)$$

$$i_t = \sigma(w_i[h_{t-1}, x_t]+b_i) \quad (5)$$

$$c'_t = tanh(w_c[h_{t-1}, x_t]+b_c) \quad (6)$$

$$c_t = f_t * c_{t-1} + i_t * c'_t \quad (7)$$

$$o_t = \sigma(w_o[h_{t-1}, x_t] + b_o) \quad (8)$$

$$h_t = o_t * tanh(c_t) \quad (9)$$

The forget gate decides which information should be thrown away from the cell state. It is defined by Eq. (4) where h_{t-1} is input from the previous state, x_t is current input, w_f is the weight, and b_f is the bias. However, the input gate decides what new information is to be stored in the cell state. It has two phases: the *sigmoid* layer and the *tanh* layer that retain information in the cell state. The *sigmoid* layer decides which value to be updated and is defined by Eq. (5), where w_i is the weight, and b_i is the bias. However, the *tanh* layer creates the value for the new candidate c'_t which could be added to the cell state and is defined by Eq. (6) where w_c and b_c are weight and bias respectively. Equation (7) updates the old cell state c_{t-1} to the new cell state c_t where f_t has the right amount of information to be added to the cell state and $i_t * c'_t$ provides new information that is required to be retained. Similar to the input gate, the output gate has two phases: the *sigmoid* layer that decides what final output should be generated as defined by Eq. (8) and the *tanh* layer to normalize values between -1 to 1 as defined by Eq. (9).

Materials and Methods

The proposed model training pipeline for Dzongkha word prediction was divided into three phases: dataset preparation, sequence generation, and model training as shown in Figure 4.

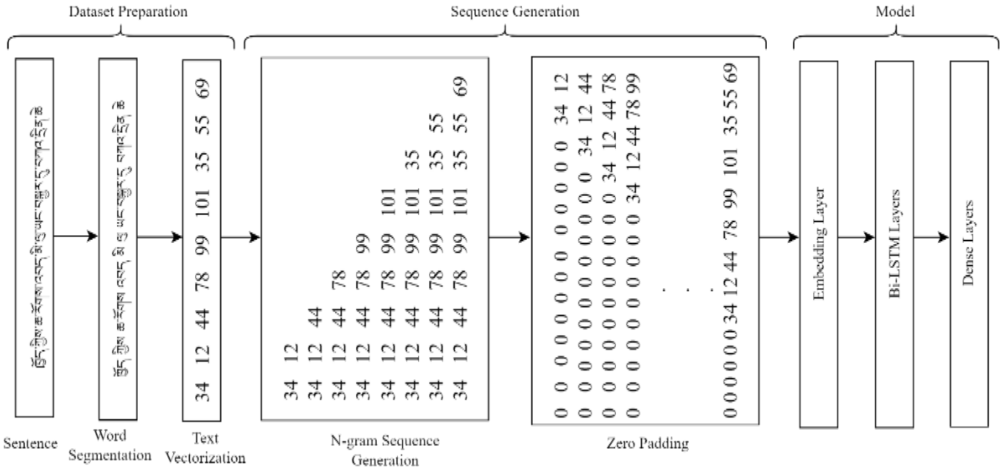

Fig. 4 *Dzongkha next word prediction model pipeline*

Dzongkha Dataset Preparation

Dzongkha sentences were selected from different genres such as stories, songs, poetry, newspapers, and books. Segmented words were validated by linguists from the Dzongkha Development Commission of Bhutan. However, the dataset was further pre-processed. The digits, special symbols, and extra spaces were removed from the sentences. The *tsheg* after a single syllable word was removed to provide a vivid distinction between multiple syllabic words. Furthermore, white space was introduced between words similar to English for easy tokenization.

The dataset consisted of 16080 sentences. However, first 10000 sentences were considered for the training. These sentences were further tokenized into words. The dataset encompassed 4820 unique words and 52150 sequences.

Sequence Generation

A word corpus dictionary was created and every unique word was assigned an integer number. For example, a sentence "ཁྱོད ཀྱིས ཆ་རོགས འབད མི ལུ ཡང་བསྐྱར་དུ བཀའ་དྲིན ཆེ" is equivalent to "*34 12 44 78 99 101 35 55 69*" where ཁྱོད is assigned with integer number 34. Next, using the N-gram method,

the number of sequences was generated starting from bi-gram to N-gram. For example: eight sequences were generated from a sentence of length nine words *("34 12", "34 12 44", "34 12 44 78", "34 12 44 78 99", "34 12 44 78 99 101", "34 12 44 78 99 101 35", "34 12 44 78 99 101 35 55", "34 12 44 78 99 101 35 55 69*") as shown in Figure 4.

Similarly, 52150 sequences were generated from 16080 sentences. The maximum and minimum sentence length was 18 and two respectively. A total of 52139 sequences from 52150 had a sequence length of less than 15. As a result, a sequence length of 15 was considered and pre-padded zero at the beginning of each sequence. The zero-padding makes sequence length equal.

Model Training

The zero-padded sequences were fed into word embedding followed by Bidirectional LSTM for the model training. Text data cannot be given to machine learning models directly. Models take only vectors (numbers). There are several methods to convert text to a number such as one-hot encoding, TF-IDF, word2vec, CBOW, and GloVe. However, one-hot encoding is sparse and inefficient. Word embedding is one of the efficient and dense representation methods to represent text to vector. In word embedding, words with the same meaning have similar word representations. It establishes the semantic and syntactic relationship between every word. The padded sequences were fed into the word embedding layer with 300 embedding dimensions. The dimension of the embedding layer is the hyper-parameters that can be fine-tuned to get the best result from the model training.

The Bidirectional Long Short-Term Memory (Bi-LSTM) is the variant of LSTM. It allows input to flow in two directions as shown in Fig. 5. This behavior of Bi-LSTM preserves past and future information (Panchendrarajan & Amaresan, 2018). To understand a word in language processing, both previous and subsequent words' contexts information are required. Consider two sentences: *"Karma likes python because it takes less time to code"* and *"Karma likes python because they are not poisonous".* How the correct meaning of the word *"python"* is determined? The prior two words *"Karma likes"* in the two sentences

are the same. Therefore, the subsequent words' information is required to determine the correct meaning of the word *"python"*. LSTM only considers pass contexts to predict the current word. However, Bi-LSTM considers past contexts (h_1, h_2) and the future contexts (h_4, h_5, h_6, h_7, h_8) to output h_3 as shown in Figure 5.

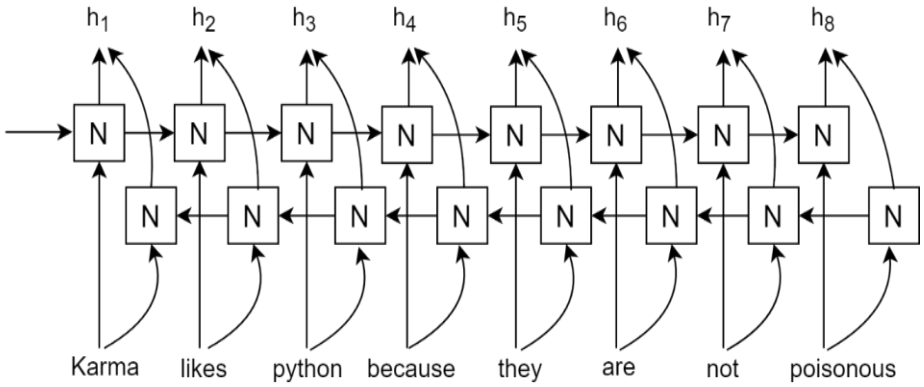

Fig. 5 *Bidirectional LSTM*

Similarly, Dzongkha's next word prediction used both past and future information by implementing the Bi-LSTM algorithm. The vectorized words are fed into two Bi-LSTM layers with 512 neurons and a dropout ratio of 0.25 as shown in Table 1. The two fully-connected dense layers were implemented with *ReLU* and *SoftMax* activation functions. The loss function and the optimizer used were *categorical_crossentropy* and *adam* respectively.

Table 1. *Bi-LSTM Model Summary.*

Layer	Output Shape	Parameters
Embedding	(None, 14, 300)	1446300
Bidirectional_1	(None, 14, 1024)	3330048
Dropout	(None, 14, 1024)	0
Bidirectional_1	(None, 512)	6295552
Dropout	(None, 1024)	0
Dense_1	(None, 2410)	2470250
Dense_2	(None, 4821)	11623431

Results and discussion

This study presents the next word prediction for the Dzongkha language. Different RNN-based models were trained. Furthermore, hybrid models such as Convolutional Neural Network (CNN) and RNN variants models were also evaluated.

Experiments Setting

The models training was conducted on Windows 10 Pro workstation configured with the NVIDIA GeForce RTX 3080 GPU, Intel(R) Core (TM) i7-10700KF CPU @3.80GHz 3.79GHz processor, 1TB storage, and 32 GB RAM. The Jupyter Notebook and Python 3 were used as the front-end tools and the machine learning library Keras with TensorFlow was used as the backend.

A total of 16 experiments were conducted using LSTM. The number of neurons and hidden layers used were 128, 256, 512, and 1024 with one, two, three, and four as shown in Table 2. The embedding dimension of 300 with a dropout ratio of 0.25 with the early stopping of patience three and batch size of 512 was used for training the models. It was observed that the fine-tuning of embedding dimensions and the dropout ratio contribute to the learning. The model learning was vivid from observing accuracy differences. However, changes in batch size do not affect model learning. The accuracy and loss range from 56.63% to 73.79% and 1.0721 to 1.8031 respectively.

Table 2 *LSTM accuracy with different hyper parameters.*

Model	Experiments	Layer	Neurons	Epoch	Time (s)	Loss	Accuracy (%)
LSTM	1	1	128	109	136	1.1214	73.25
	2		256	74	95	1.1060	73.51
	3		512	74	119	1.0721	73.79
	4		1024	52	124	1.0892	73.66
	5	2	128	148	200	1.2195	70.72
	6		256	97	152	1.1649	72.36
	7		512	69	154	1.1358	72.91
	8		1024	78	344	1.0816	73.53
	9	3	128	160	246	1.3665	66.97

	10		256	110	200	1.2877	69.10
	11		512	115	330	1.1875	71.55
	12		1024	108	711	1.1140	73.00
	13	4	128	198	338	1.8031	56.63
	14		256	237	493	1.3451	67.21
	15		512	208	714	1.2598	69.15
	16		1024	189	1632	1.1127	72.61

Table 3 *Results of RNN-based models with different hyper parameters.*

Model	Neurons	Epochs	Training Time(s)	Loss	Accuracy (%)
LSTM	512	74	119	1.0721	73.79
Bi-LSTM	512	71	155	1.0722	73.89
GRU	1024	63	132	1.0798	73.77
CNN+LSTM	1024	75	120	1.1231	73.28
CNN+Bi-LSTM	512	140	180	1.0653	73.86
CNN+GRU	1024	72	60	1.1180	73.25

Similar to the LSTM model as illustrated in Table 2, 16 experiments were conducted each with Bi-LSTM, GRU, CNN+LSTM, CNN+Bi-LSTM, and CNN+GRU. Table 3 summarizes the best accuracies of each model. The increase in the number of hidden layers and neurons does not affect the rate of learning. However, a deep network increases training time. It was observed that the one hidden layer and 512 neurons Bidirectional Long Short-Term Memory (Bi-LSTM) provided the best accuracy of 73.89%. However, the lowest loss of 1.0653 was given by the CNN + Bi-LSTM hybrid model.

Figure 6 shows the accuracy and the loss of the Bi-LSTM model. The accuracy gradually increased from the first epoch to 30 epochs. However, there was not much learning after 60 epochs and the accuracy remained fairly steady. Similarly, the loss plummeted in the initial epochs and continued to decrease over the next 40 epochs. However, the loss remained fairly static after 80 epochs. The s-layered Bi-LSTM model with 512 neurons obtained the highest accuracy of 73.89% which took 71 epochs using an early stopping mechanism.

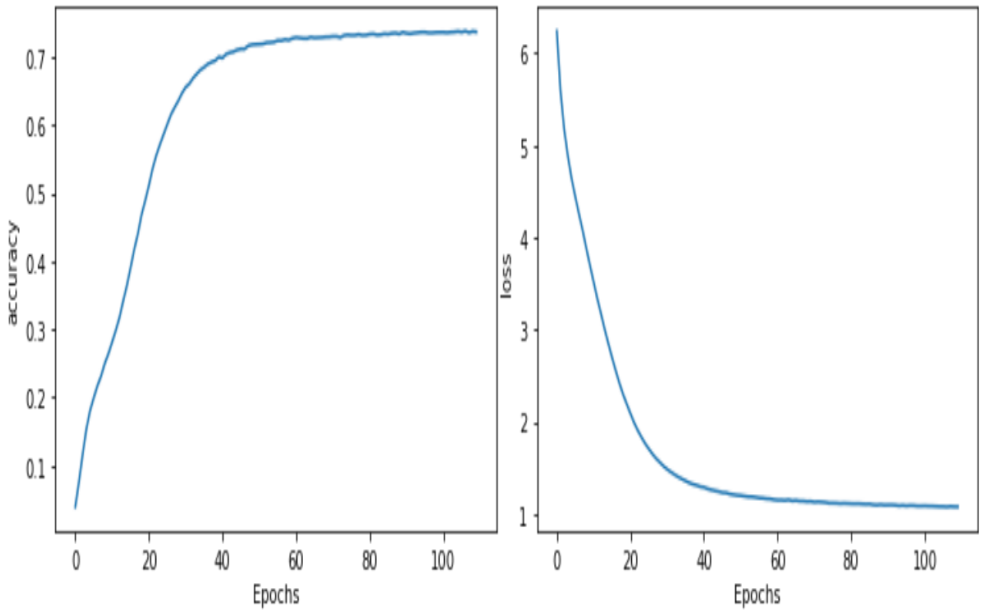

Fig. 6 *Bi-LSTM model accuracy Vs Loss*

Table 4 *A single input word with next predicted words.*

First input word	Next five predicted words				
	1	2	3	4	5
གཅི་སྟེ	མིན་ན	འཁྲབ་སྟོན	བཅོ	ཚོལ	དཀའ་ངལ
གནམ་མེད་ས་མེད	བཀའ་དྲིན	མཁྱེན་པ་མ་བཙུག	གཙང་སྦ	རྒྱུང	དགོད་ཟས་གི
སྨོ་བ	འདི	མིན་འདུག	མེད	བཀྲོན་མ	མ་ཚོ་ཚེག
གནས་སྐབས	ཕྱིན	གཱ་རའི	སོ་སོ	འདི	ཚུ
མེ་དཔལ	འདི	གཱ་ར	འཕྲུལ་ཆས	ཚུ	བཀྲོག་གཏང
བསོད་ནམས་ཅན	དང	ཚུ	ཁོ་རྒྱུས	ལྟ	ང་བཅས
གོང་འཕེལ	སྣ་ཚོགས	ཀྱི	དང	འགྱོཔ	མེད་པ
རྒྱལ་ཁབ	གཱ་རའི	གཱ་ར	ལས	གཉེས	ལེགས་ཤོམ

Table 4 shows the next five predicted words given a single input word. The initial word གནས་སྐབས (idea) predicts ཕྱིན (give), གཱ་རའི (all), སོ་སོ (different), and འདི (this). Similarly, the input word གོང་འཕེལ generates སྣ་ཚོགས, ཀྱི, དང, འགྱོཔ, and མེད་པ. Any one of these words can be chosen depending on the context to write a sentence. For instance, གནམ་མེད་ས་མེད (extremely) can be selected with any of the predicted words as shown in Table 4 to

form sentences: གནམ་མེད་ས་མེད་བཀའ་དྲིན་ཆེ (thank you very much), གནམ་མེད་ས་མེད་མགྱོགས་ཤ་མ་བཚུག (do not run extremely fast), གནམ་མེད་ས་མེད་གཙང་སྦྲ་དགོ (extremely needs cleanliness), གནམ་མེད་ས་མེད་རླུང་འདུག (extremely windy), and གནམ་མེད་ས་མེད་དགོད་ཐུས་ཤི (extremely funny).

Conclusion

In this paper, the next Dzongkha word prediction is presented. The study conducted on Dzongkha next syllables prediction system (Wangchuk et al., 2021) reduced keystrokes for the Dzongkha typing. However, the number of keypresses required is still more. This study further reduces keypress and makes Dzongkha typing faster. The segmented Dzongkha words dataset was collected from the Dzongkha Development Commission of Bhutan. Furthermore, dataset pre-processing was performed. The variant of RNN-based models was evaluated. It is observed that the Bi-LSTM model with two hidden layer and 512 hidden layer neurons achieved the best accuracy of 73.89%. It has been observed that the accuracy does not increase with hyper parameters fine-tuning. In the future, it will be interesting to conduct the model training with an increased dataset with the stop words removed. Furthermore, a graphical user interface can be developed to deploy the trained model

References

Balkin, S. D., & Ord, J. K. (2000). Automatic neural network modeling for univariate time series. *International Journal of Forecasting*, *16*(4), 509–515. https://doi.org/10.1016/S0169-2070(00)00072-8

Barman, P. P., & Boruah, A. (2018). A RNN based Approach for next word prediction in Assamese Phonetic Transcription. *Procedia Computer Science*, *143*, 117–123. https://doi.org/10.1016/J.PROCS.2018.10.359

Cho, K., & Merriënboer, B. Van. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *Arxiv.Org*. https://arxiv.org/abs/1406.1078

Chungku, C., Rabgay, J., & Faaß, G. (2010). Building NLP resources for Dzongkha: a tagset and a tagged corpus. *In Proceedings of the Eighth Workshop on Asian Language*, 21–22. https://www.aclweb.org/anthology/W10-3214.pdf

Dhungyel, P. R., & Grundspenkis, J. (2017). Analysing the Methods of Dzongkha Word Segmentation. *Sciendo.Com*, *21*(1), 61–65. https://doi.org/10.1515/acss-2017-0008

Hochreiter, S., & Schmidhuber, J. (1997). LSTM can solve hard long time lag problems. *Advances in Neural Information Processing Systems*, 473-479.

Jamtsho, Y., & Muneesawang, P. (2020). Dzongkha word segmentation using deep learning. *KST 2020 - 2020 12th International Conference on Knowledge and Smart Technology*, 1–5. https://doi.org/10.1109/KST48564.2020.9059451

Jing, K., & Xu, J. (2019). *A Survey on Neural Network Language Models*. http://arxiv.org/abs/1906.03591

Jozefowicz, R., Zaremba, W., & Sutskever, I. (2015). An empirical exploration of recurrent network architectures. *Proceedings.Mlr.Press*. http://proceedings.mlr.press/v37/jozefowicz15.html

Koutnik, J., Greff, K., Gomez, F., & Schmidhuber, J. (2014). A Clockwork RNN. *Proceedings of the 31st International Conference on Machine Learning*, 1863–1871. http://proceedings.mlr.press/v32/koutnik14.html

MacDonald, G., Godbout, A., Gillcash, B., & Cairns, S. (2021). Volume-preserving Neural Networks. *Proceedings of the International Joint Conference on Neural Networks*, *2021-July*. https://doi.org/10.1109/IJCNN52387.2021.9534112

Norbu, S., Choejey, P., Dendup, T., Hussain, S., & Mauz, A. (2010). Dzongkha word segmentation. *Aclweb.Org*, 21–22. https://www.aclweb.org/anthology/W10-3213.pdf

Norbu, T., & Namgyel, T. (2019). Languages and technology in Bhutan.

Proceedings of the Language Technologies for All, 235–238.

Olah, C. (2015). *Understanding LSTM Networks*. http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Panchendrarajan, R., & Amaresan, A. (2018). Bidirectional LSTM-CRF for named entity recognition. *Aclweb.Org*.

Pedamkar, P. (2020). *Recurrent Neural Networks*. EDUCBA. https://www.educba.com/recurrent-neural-networks-rnn/

Shakhovska, K., Dumyn, I., Kryvinska, N., & Kagita, M. K. (2021). An Approach for a Next-Word Prediction for Ukrainian Language. *Wireless Communications and Mobile Computing*, *2021*. https://doi.org/10.1155/2021/5886119

Sharma, R., Goel, N., Aggarwal, N., Kaur, P., & Prakash, C. (2019). Next Word Prediction in Hindi Using Deep Learning Techniques. *2019 International Conference on Data Science and Engineering, ICDSE 2019*, 55–60. https://doi.org/10.1109/ICDSE47409.2019.8971796

Shi, Z., Shi, M., & Li, C. (2017). The prediction of character based on recurrent neural network language model. *Proceedings - 16th IEEE/ACIS International Conference on Computer and Information Science, ICIS 2017*, 613–616. https://doi.org/10.1109/ICIS.2017.7960065

Stremmel, J., & Singh, A. (2021). Pretraining Federated Text Models for Next Word Prediction. *Advances in Intelligent Systems and Computing*, *1364 AISC*, 477–488. https://doi.org/10.1007/978-3-030-73103-8_34

Tshering, K., & Van Driem, G. (2019). *The Grammar of Dzongkha*. https://doi.org/10.5070/H918144245

Wangchuk, K., Riyamongkol, P., & Waranusast, R. (2021). Next syllables prediction system in Dzongkha using long short-term memory. *Journal of King Saud University - Computer and Information Sciences*. https://doi.org/10.1016/J.JKSUCI.2021.01.001

Yao, K., Cohn, T., Vylomova, K., Duh, K., & Dyer, C. (2015). Depth-Gated Recurrent Neural Networks. *ArXiv Preprint ArXiv:1508.03790 9*.

About the authors

Mr. Karma Wangchuk is an Associate Lecturer at the Information Technology Department, College of Science and Technology (CST), Royal University of Bhutan. He completed his Bachelor of Engineering in Information Technology from the College of Science and Technology, Rinchending, Phuentsholing, Royal University of Bhutan, and Master of Engineering in Computer Engineering from Naresuan University, Phitsanulok, Thailand. His area of interest is in Machine Learning, Computer Vision, IoT, Natural Language Processing, and Big Data. His past publications include next syllables prediction for Dzongkha and Bhutanese Sign Language recognition. He is also one of the recipients of the AURG2021-2022 research grant. His team is currently working on a CST-DDC project titled “English to Dzongkha Translation Using Neural Machine Translation”.

Mr. Tandin Wangchuk is a lecturer at the College of Science and Technology (CST). He is currently leading the Information Technology Department and looking after the smooth operation of the Bachelor of Engineering in Information Technology as the Head of the Department (HoD) and Programme Leader. He has a Master of Information Technology (MIT) in Network Computing from the University of Canberra (2012), Australia, and a Bachelor of Computer Science from the University of New Brunswick, Canada (2007). He has been working at CST since 2008 holding various positions and responsibilities. He is an ardent proponent of promoting programming and motivating students to take a keen interest in technologies.

Mr. Tenzin Namgyel graduated with B.Tech in Computer Science and Engineering from the National Institute of Technology, Warangal, India. He is currently working as a Deputy Chief ICT Officer at Dzongkha Development Commission, Thimphu, Bhutan. He is involved in the digitization of Dzongkha and the promotion of Dzongkha through information technology. He also explores and researches automation in language processing to give easy and equal access to knowledge, information, and services to different sections of the society including those marginalized ones in the era of digital technology.