International Journal of Computer Applications (0975 – 8887)
Volume 178 – No. 31, July 2019

Nepali Speech Recognition using RNN-CTC Model

Paribesh Regmi
Electronics and Communication Engineer
Institute of Engineering Tribhuvan University

Arjun Dahal
Electronics and Communication Engineer
Institute of Engineering Tribhuvan University

Basanta Joshi
Assistant Professor Department of Electronics and Computer Engineering
Institute of Engineering Tribhuvan University

ABSTRACT
This paper presents a Neural Network based Nepali Speech Recognition model. RNN (Recurrent Neural Networks) is used for processing sequential audio data. CTC (Connectionist Temporal Classification) [1] technique is applied allowing RNN to train over audio data. CTC is a probabilistic approach of maximizing the occurrence probability of the desired labels from RNN output. After processing through RNN and CTC layers, Nepali text is obtained as output. This paper also defines a character set of 67 Nepali characters required for transcription of Nepali speech to text.

Keywords
Artificial Intelligence, Machine Learning, Automatic Speech Recognition, Recurrent Neural Network, Connectionist Temporal Classification, Softmax, Hidden Markov Model, Nepali Speech Recognition, Long-Short Term Memory (LSTM), Backpropagation, Character Error Rate

1. INTRODUCTION
With the advancement in Machine Learning models, machines are being made capable of doing tasks that otherwise required human intelligence. Automatic Speech Recognition (ASR) is a challenging as well as interesting problem in the field of Artificial Intelligence today. ASR is a process of mapping an acoustic waveform into text in which the text should have a meaning equivalent to the spoken words. It can change the way humans interact with computers, which happens mostly through text today. A lot of existing systems can be automated with Interactive Voice Response systems with speech recognition technology.

The goal of this paper is to present an end to end Nepali Speech Recognition model with the use of recurrent neural networks. A neural network is an artificial self-learning structure modeled to resemble human brain. RNN is a form of neural network that is capable of learning sequential data i.e. data represented as a time-sequence. Long sequence of audio data is divided into small frames and fed in one after another to RNN, which is capable of learning patterns in audio sequences. Training machine learning models such as RNN generally requires pre-labelled data (audio with its corresponding text). Since audio data are unsegmented (framewise labelling is not possible), CTC layer is applied over RNN for training which eliminates the need for framewise labelling. CTC layer takes in target text from data and maximizes the occurrence probability of the text from RNN output.

2. RELATED WORK
Speech Recognition has been practiced widely with variety of models. In early days, HMM (Hidden Markov Model) was used for speech recognition. HMM is a probabilistic model that is capable of learning patterns in time-sequence data. However, the performance of HMM was only good for limited vocabulary within a limited context. With the development of efficient training methods of Neural Networks [2][3], speech recognition was carried out with neural networks and the results were improved. Neural Network based models were able handle large vocabularies [4]. Speech recognition systems were also able to identify speech from a wide range of context.

Towards End-to-End Speech Recognition with Recurrent Neural Networks [5] presents how speech recognition tasks can be carried out using Recurrent Neural Networks. It was shown that RNNs were much efficient in handling speech data compared to previous existing models. The Neural network had LSTM cell that could manipulate long range sequences. The experiment was carried out on a wall street journal for English corpus. For English corpus, a total of 43 characters built up the language. CTC layer was used above RNN for handling unsegmented data.

Automatic Speech Recognition for Nepali Language [6] project is the outcome of implementation of HMM, Java speech grammar format in the acoustic model of Sphinx-4. Sphinx-4 is a state-of-the-art speech recognition system written entirely in the Java programming language. Training of the model composed of Gaussian Acoustic Model for filtering phones of a word. Acoustic model consists of numeric transformation of each phone in matrix form. So, when speaker speaks a word, it first portioned in phones and numeric transformation of phones is compared with acoustic model.

HMM Based Isolated Word Nepali Speech Recognition [7] project was carried using HMM. The project was built to identify 10 distinct Nepali words.

3. ASR MODEL
The proposed model is illustrated in Figure 1. First layer of the network takes audio features as input. Audio features are then processed by layers of Bidirectional RNN. LSTM cells are used for RNN for their capability of learning long-time sequence patterns. Output form Bidirectional RNN is processed by dense layers. The output vector from dense layer is the same size as the total number of characters in the defined Nepali character set plus an extra blank character (discussed in Section 3.4 and 3.5). The output is forwarded to Softmax layer which assigns occurrence probability to each character. This probability vector is forwarded to CTC layer for CTC loss calculation and the network is backpropagated for minimization of this loss.

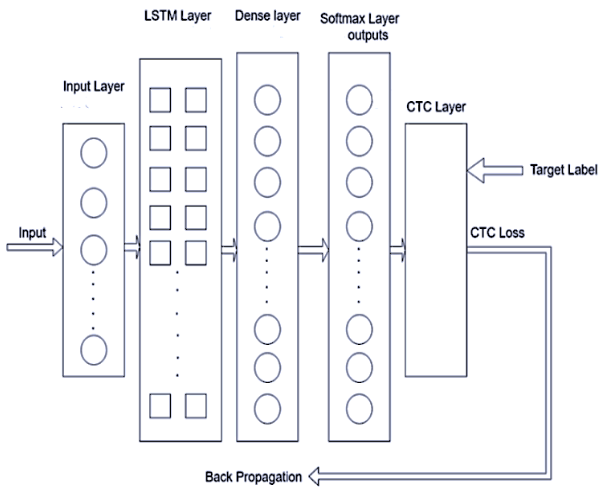

Fig 1: Network Architecture for Nepali Speech Recognition

3.1 Long-Short Term Memory [8]

Standard RNN cells are incapable of learning long time-dependent patterns i.e. it suffers from Vanishing Gradient Problem [9]. Long-Short Term Memory cells are widely used in RNN as they overcome this issue.

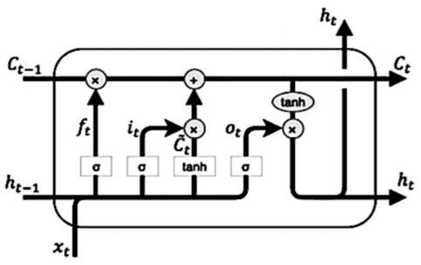

Fig 2: An LSTM Cell

The output of an LSTM cell is given by the following set of equations where B term denote biases, W term denote weight matrices (e.g. $[W_{hi}]$ is the hidden-input weight matrix), f is the forget gate, i is the input gate, o is the output gate, x is the input vector, σ and $tanh$ are activation functions. '*' denotes element-wise multiplication.

$$[f_t] = \sigma([x_t][W_{if}] + [h_{t-1}][W_{hf}] + [B_f]) \quad (1)$$
$$[i_t] = \sigma([x_t][W_{ii}] + [h_{t-1}][W_{hi}] + [B_i]) \quad (2)$$
$$[o_t] = \sigma([x_t][W_{io}] + [h_{t-1}][W_{ho}] + [B_o]) \quad (3)$$
$$[\tilde{C}_t] = tanh([x_t][W_{i\tilde{c}}] + [h_{t-1}][W_{h\tilde{c}}]) \quad (4)$$
$$[C_t] = [f_t] * [C_{t-1}] + [i_t] * [\tilde{C}_t] \quad (5)$$
$$[h_t] = [o_t] * \tanh([C_t]) \quad (6)$$

$[h_t]$ is emitted as an output from LSTM layer.

3.2 Bidirectional RNN [10]

In regular recurrent neural networks, the output obtained at a current time step is the function of its current and past history of inputs. The input information to the network can be increased if the network also considers the inputs occurring in future time steps. The concept of BRNN is to have another recurrent layer, in addition to the forward running layer, that runs backward in time. The backward running layer first feeds in the input from last time step and preceding inputs successively. The two layers run independently and do not have any connection between them. Bidirectional RNN are found to outperform unidirectional RNNs in speech recognition tasks [11].

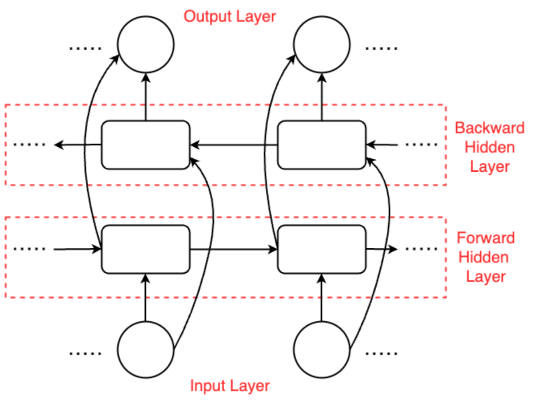

Fig 3: Structure of a Bidirectional RNN

3.3 Connectionist Temporal Classification

Generally, time-sequence data are trained as frame-wise classifiers in which the training dataset has target label for every frame. This is not applicable in speech recognition problem since the number of frames and total number of label characters are not equal. For example, an audio clip of 5 seconds has 500 numbers of frames, but the text output might only be of 30 characters. Also, same text can be read fast or slow, creating varying number of frames for same text.

Creating framewise alignment of labels is a difficult task. CTC provides a way to resolve this problem. CTC loss is an error function that allows RNN to be trained over time-sequence data that doesn't have frame-wise alignment between input and target sequences.

3.4 Training with CTC

There are 67 characters in Nepali language that can represent sound in the form of text.

L = {'क', 'ख', 'ग', 'घ', 'ङ', 'च', 'छ', 'ज', 'झ', 'ञ', 'ट', 'ठ', 'ड', 'ढ', 'ण', 'त', 'थ', 'द', 'ध', 'न','प', 'फ', 'ब', 'भ', 'म', 'य','र', 'ल', 'व','श', 'ष', 'स', 'ह','अ', 'आ', 'इ', 'ई', 'उ','ऊ', 'ए', 'ऐ', 'ओ', 'औ', 'ा', 'ि', 'ी', 'ु', 'ू', 'ृ', 'े', 'ै', 'ो', 'ौ', '्', 'ँ', '०', '१', '२', '३', '४', '५', '६', '७', '८', '९','।', ' '}

The character set is indexed from 0 to 66 starting from 'क' as 0 and increasing accordingly.

The output from the dense layer is first fed to a Softmax layer where a number associated with each output character is assigned a probability value for the occurrence of the character.

$$softmax(y_i) = \frac{\exp(y_i)}{\sum_i \exp(y_i)} \quad (7)$$

The activations of first $|L|$ units are interpreted as the probabilities of observing corresponding labels at particular times. The activation of one extra label is the probability of observing a blank or no label. Together these outputs define the probabilities of all possible ways of aligning all possible label sequences with the input sequence. A target sequence can be obtained by condensing different alignments of characters at different time steps. These different alignments are called paths.

Let us assume for a sequence x of T time frames, we are to find probability of a label sequence l:

l = 'कमल'

Then, we can have many sets of paths from output that can converge to this label. Some of them are ('-' is the blank label):

$paths = \{--$ ककक $-----$ मम $----$ लल $--$,

$---$ कक $---$ म $--$ ललल $------$,

$-----$ कक $------$ मल $------$,

$-----$ क $------$ म $---$ लल $---$,

$.....etc\}$

All of the above paths correspond to target label l when decoded and defined such that

$$D(p) = l, \quad for\ p\ in\ paths \tag{8}$$

The decoding rule is that, first successively repeated similar characters are merged to one single character and then blank labels are discarded to get the desired label. Let, n^{th}character in path p be defined as $p(n)$ and π_k^t be the probability of occurrence of character k at time t. Then, for each path p in paths, the probability of its occurrence can be calculated from Softmax output as:

$$P(p|x) = \prod_{t=1}^{T} \pi_{p(t)}^t \tag{9}$$

given that the RNN outputs at different times are independent of each other. Hence, the total probability of occurrence of the label l is the total sum of probabilities of occurrence of each path p in paths. i.e.

$$P(l|x) = \sum_p P(p|x), \quad for\ p\ in\ paths \tag{10}$$

$$P(l|x) = \sum_{\substack{p \\ D(p)=l}} \prod_{t=1}^{T} \pi_{p(t)}^t \tag{11}$$

The CTC loss function is now defined as:

$$CTC(x) = -log(P(l|x)) \tag{12}$$

This is our objective function which is to be minimized while training. Minimizing this objective function simply means maximizing the probability of occurrence of each path in paths which ultimately maximizes the probability of occurrence of the label l.

3.5 The CTC Forward-Backward Algorithm

From equation (12), it seems difficult to recognize every path and costly to calculate probability for each one. Forward-Backward Algorithm is a recursive approach for calculation of $P(l|x)$.

Let, $l_{1:s}$ denote the first s characters of a label l and forward variable $\alpha_l^t(s)$ be the probability of occurrence of $l_{1:s}$ at time t.

$$\alpha_l^t(s) = \sum_{\substack{p \\ D(p)=l_{1:s}}} \prod_{t'=1}^{t} \pi_{p(t')}^{t'} \tag{13}$$

Minimizing $CTC(x)$ by maximum likelihood, we can assume that most of the time the output from Softmax layer will have a high probability of observing blank character (-), which doesn't change the output at all. From a well-trained model, we expect blank character in between successive characters in the character set. So, it is better we train our model with modified label l' obtained by adding blank (-) at the beginning and end and in between every character in l.

Now, network firing a blank will correspond to this blank between characters. By training with l', all labels start with a blank. We can initialize the $\alpha_{l'}^t(s)$ variable as:

$\alpha_{l'}^1(1) = \pi_b^1$ and $\alpha_{l'}^1(2) = \pi_{l'_2}^1$

$$\alpha_{l'}^1(s) = 0, \forall s > 2 \tag{14}$$

where, b is the blank label and l'_2 is the second character in l' i.e. the beginning character in l. The second initialization is so because more than one character in l cannot be obtained in a single time step.

According to our decoding rule defined earlier that, first successively repeated similar characters are merged to one single character and then blank labels are discarded to get the desired label, prefix can only be increased when the last character in the prefix and the incoming character are different. The recursive equations can be written as:

$\alpha_{l'}^t(s) = [\alpha_{l'}^{t-1}(s) + \alpha_{l'}^{t-1}(s-1)].\pi_{l'_s}^t \ \ if\ l'_s = b\ or\ l'_{s-2} = l'_s$

$$otherwise,\ \alpha_{l'}^t(s) = [\alpha_{l'}^{t-1}(s) + \alpha_{l'}^{t-1}(s-1) + \alpha_{l'}^{t-1}(s-2)].\pi_{l'_s}^t \tag{15}$$

When l'_s is a blank, we can see the prefix $l'_{1:s}$ at time t if we'd already seen this prefix at time $t-1$ or if we'd seen prefix $l'_{1:s-1}$ at time $t-1$. When l'_s is a non-blank character and different from previous non-blank l'_{s-2}, according to our decoding rule, we can avoid the blank in between and make transition two steps forward directly from $\alpha_{l'}^{t-1}(s-2)$ to $\alpha_{l'}^t(s)$.

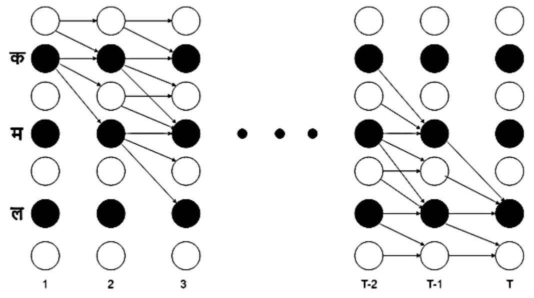

Fig 4: Showing label transitions for Forward Backward algorithm applied to label 'कमल' [1]

In figure 4, white circles and black circles represent blank labels and non-blank labels respectively. For this case l' = _क_म_ल_. Arrows specify the allowed transitions at every time step. The transition between black to black and white to white on same horizontal line is for retaining same prefix on coming from previous to current time step. One step downward transition is for extending prefix in the current time step. Downward transition by two steps can also be seen between distinct non-blank labels which are for avoiding blank between distinct non-blanks to make a two-step transition.

The probability $P(l|x)$ is the sum of total probabilities of l' with and without final blank label at time T as given in equation (16).

$$P(l|x) = \alpha_{l'}^T(|l|) + \alpha_{l'}^T(|l|-1) \tag{16}$$

Similarly, we can define a backward variable $\beta_l^t(s)$ as the probability of occurrence of $l_{s:|l|}$ at time t to calculate $P(l|x)$

4. DECODING

After the model is trained, the implementation steps are as shown in Fig 5.

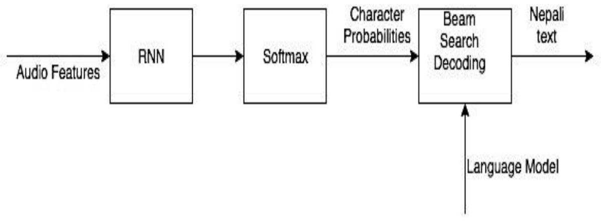

Fig 5: Implementation of a trained model

On implementing a trained model, audio features are processed by RNN and Softmax layer successively. The output from Softmax layer is the occurrence probabilities of different characters at different time steps. The task of decoding is to find a label with maximum occurrence probability.

One straightforward method is Best Path Decoding, to pick the most probable character at each time step. In this case, y is the label corresponding to most probable path in the output.

$$y = D\left(argmax_a \prod_{t=1}^{T} \pi_{a(t)}^t\right) \qquad (17)$$

Best Path decoding in equation (17) is time and space efficient to compute, however it does not guarantee the most probable labelling.

Second method is the Prefix Search Decoding algorithm. It is carried out by expanding prefixes at every time step and storing them with their probabilities. At the end, the prefix with highest probability is the desired label. Given enough time, Prefix Search Decoding always finds the most probable labeling. However, it is computationally infeasible if the input sequence length is large because the prefixes grow exponentially with time.

4.1 Beam Search Decoding

For practical implementation of Prefix Search Decoding, beam width is defined which keeps only specified number of most probable prefixes at each time step. Also threshold probability is set which discards the characters at each time step from softmax layer with probability not exceeding a threshold. Prefix Search Decoding with beam width defined is also known as Beam Search Decoding.

Let us define beam width W and threshold probability P_t. $S(c,t)$ be the softmax output for character c at time step t and $Lm(x)$ be the probability from language model for sequence x. $P_b(x,t)$, $P_{nb}(x,t)$ are the probabilities of sequence x at time t ending in blank and non-blank characters respectively. Let, w_s be white space (space character) and lastly define l_e as the last character in l.

Algorithm 1: Beam Search Decoding

```
Initialize $\boldsymbol{L} \leftarrow \{\Phi\}$, $\boldsymbol{P_b(\Phi, 0) \leftarrow 1}$
for $\boldsymbol{t = 1\ to\ T}$ do
            $\boldsymbol{L' \leftarrow W\ most\ probable\ sequence\ in\ L}$
                        $\boldsymbol{L} \leftarrow \{\}$
$\boldsymbol{C_t} \leftarrow \{\boldsymbol{c : S(c,t) > P_t}\}$
for $\boldsymbol{l \in L'}$ do
        for $\boldsymbol{c \in C_t}$ do
            if $\boldsymbol{c} = \Phi$ then
                $\boldsymbol{P_b(l,t) \leftarrow S(c,t) * [\, P_b(l,t-1) + P_{nb}(l,t-1)]}$
            if $\boldsymbol{c = l_e}$ then
                $\boldsymbol{P_{nb}(l+c,t) \leftarrow S(c,t) * P_b(l,t-1)}$
                $\boldsymbol{P_{nb}(l,t) \leftarrow S(c,t) * P_{nb}(l,t-1)}$
            if $\boldsymbol{c = w_s}$ then
              $\boldsymbol{P_{nb}(l+c,t) \leftarrow Lm(l+c) * S(c,t) * \begin{bmatrix} P_b(l,t-1) \\ + P_{nb}(l,t-1) \end{bmatrix}}$
            else
              $\boldsymbol{P_{nb}(l+c,t) \leftarrow S(c,t) * [\, P_b(l,t-1) + P_{nb}(l,t-1)]}$
$\boldsymbol{L} \leftarrow \{\boldsymbol{x : P_{nb}(x,t) > 0}\} \cup \{\boldsymbol{y : P_b(y,t) > 0}\}$
return $\boldsymbol{argmax_{l \in L}\, [P_b(l,T) + P_{nb}(l,T)]}$
```

At the beginning L is initialized to null and expanded as prefixes are grown. The basis for prefix expansion in the current time step is the list of prefixes obtained in previous time step. Each prefix obtained in previous time step is grown with characters in current time step. At last, the sequence in L with highest total probability is returned as the most probable labeling for given input sequence.

4.2 Language Model

Transcription from RNN-CTC network may not always be grammatically correct or may not contain a valid word or phrase. It is the task of language model to check the validity of a word or sentence. It can be defined as a conditional probability distribution of occurrence of next word/character in a sequence given the series of previous words/characters. Language model is integrated in beam search decoding algorithm (Algorithm 1). Every time a prefix is extended with a space character, the probability of occurrence of last word in that sequence is calculated.

N gram modelling is the concept of looking back to N previous elements in the sequence. In N gram modelling, probability of next element is calculated as in equation (18):

$$P\left(w_i | w_{i-(N-1)} \ldots\ldots w_{i-1}\right) \qquad (18)$$

Let us assume the case of simple one-gram model, we need to check the probability of occurrence of a sequence:

$$L = < w_1\, w_2\, w_3 \ldots w_n >$$

$$< and > \ are\ start\ and\ end\ characters$$

Probability of occurrence of L can be split into conditional probabilities as in equation (19):

$$P(L) = P(w1 \mid <) * \left[\prod_{p=2}^{n} P\left(w_p | w_{p-1}\right)\right] * P(> | w_n) \qquad (19)$$

The individual conditional probabilities occurring above are based on frequency count of words in the large dataset. If $C(w_1)$ is the total number of occurrences of the word/character w_1 and $C(w_1\, w_2)$ is the total occurrence of sequence $w_1\, w_2$, the conditional probability is given as:

$$P(w_2 | w_1) = \frac{C(w_1\, w_2)}{C(w_1)} \qquad (20)$$

5. ERROR MEASUREMENT

There should be a mathematical measure of how accurate our model is in speech to text transcription tasks. Edit distance (ED) is a measure of error in sequence transcription tasks. If, $< a1, a2, a3, \ldots \ldots an >$ is the most probable sequence obtained from the model and $< b1, b2, b3, \ldots \ldots bn >$ is the target sequence, then edit distance between two sequences is the total number of operations (deletions, insertions and substitutions) required to transform obtained sequence to target sequence.

Character Error Rate (CER) is defined as the average of edit distance over the total number of samples. For test dataset of total N characters over L number of samples, the CER is defined as:

$$CER = \frac{1}{N} \sum_{i=0}^{L} ED(x_i) \tag{21}$$

Where $ED(x_i)$ is the edit distance between obtained sequence for sample x_i and its target sequence.

6. EXPERIMENT

Each sentence was encoded with respective index of characters. The text was encoded as in table 1, forming integer array for each sentence. The array is a target sequence to be used in training.

Table 1: Text encoding example

विराटनगर ५ विकेटले पराजित	[28, 44, 26, 43, 10, 19, 2, 26, 66, 60, 66, 28, 44, 0, 49, 10,27, 49, 66, 20, 26, 43, 7, 44, 15]

The scrapped text had 1320 words with 617 distinct words and since the dataset was small, one-gram language modeling was carried out.

The obtained text was recorded with three different male speakers. The entire text was recorded twice by each speaker which made it a total of two hours of continuous speaking. Recording was carried out in audacity on which noise removal was carried out. The voice was windowed into frames and 15 MFCC features [12] were extracted from each frame which was the input to the network. The data was trained in the model described in figure 1, coded in Python programming language [13] using Tensorflow [14] platform. One advantage of Tensorflow was once we compute the objective function, it automatically computed gradients for backpropagation. Final size of the model had two layers of LSTM cells and a dense layer each having 300 units.

Table 2 shows CER for data predicted by model in different cases. Test data was made from the same speaker (one who was involved in training set preparation) and different speakers. The error calculated in the two cases were different.

Table 2: CER for different cases

Without Language Modeling (Same Speaker)	0.42
With Language Modeling (Same Speaker)	0.34
With language Modeling (Different Speaker)	0.52

Figure 6 shows the character probabilities emitted by a trained model. The dotted line in the graph represents probability of a blank label which is maximum in most of the frames. Probabilities for different characters are plotted in different colors. The spikes occurring for short times represent sudden increase in non-blank character probabilities at those frames and this is where blank probability drops down. The characters represented by those spikes are at the top of the figure lying above the corresponding spike. '__' represents probability for space character giving spikes at the beginning, end and in between words. It can be seen that labels 'दु','वि','के' 'लि' occurs very close together suggesting that the network learned them as single sounds although they are multiple characters. Smaller spikes occurring elsewhere are errors. Some of the outputs obtained from the trained model is shown in Table 3.

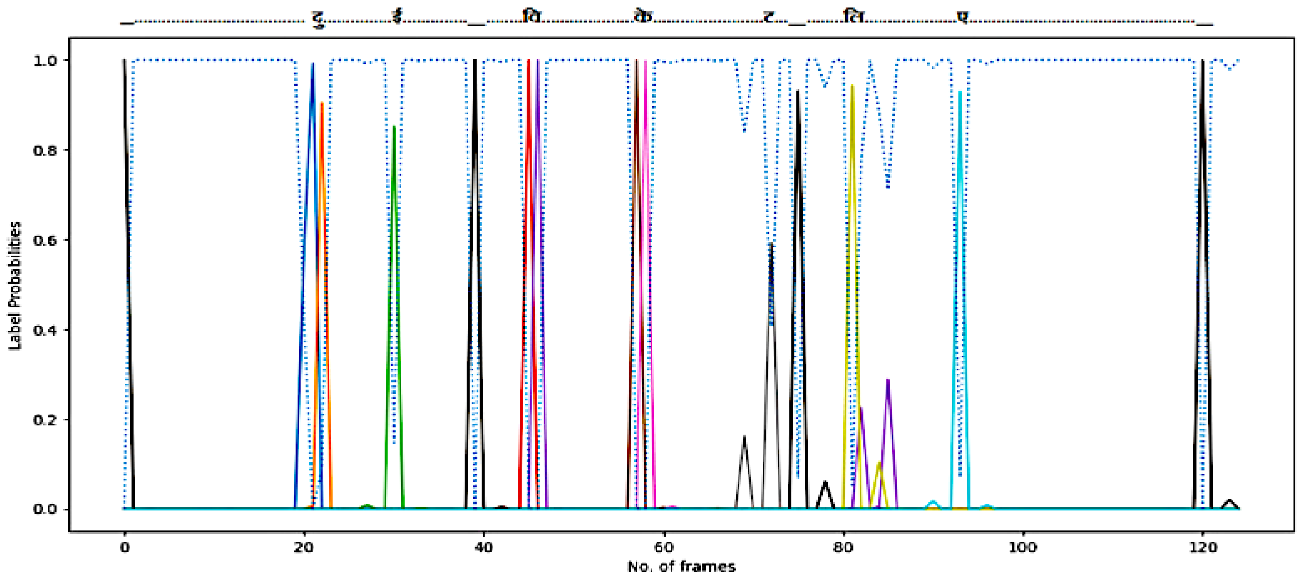

Fig 6: Showing character probabilities emitted by the softmax layer plotted against frame number for a target label 'दुई विकेट लिए'.

Table 3: Outputs from model after training

Target Text	Obtained Text
रासिद खानले	राद ािनले
दुई विकेट लिए	दुइ विके लिए
युवा तथा खेलकुद मन्त्रालयको	युव था खेलद मनतालय
सन्दीपले बलिङमा दिनदिनै	सन्दीपले बलिमा दिनसनै
सुधार गर्दै गईरहेको छ	ुबार गर्दै गईएको छ

7. DISCUSSION

Speech Recognition is one of the challenging tasks in Artificial Intelligence. RNN in this case has to learn the phonetic combinations of mixed sound patterns. Figure 6 clearly illustrates how CTC algorithm trains RNN to learn these sound patterns. However, it requires huge amount of data for such models to accurately transcribe speech to text. The major limitation for this case is the unavailability of data. Also, network identified the voice it was trained with giving lower error rate and voice from different speaker was transcribed with higher error rate. This is straightforward to explain that the model was well adapted to the training pattern and it could identify patterns from same speaker more accurately.

8. CONCLUSION

This paper has demonstrated that Nepali Speech Recognition can be carried out with RNN without having to explicitly make phonetic alignments between speech and audio sequences. We can conclude that CTC algorithm can be well-used for Speech Recognition with Nepali language. Furthermore, the set of 67 Nepali characters defined in the paper is sufficient to transcribe Nepali speech to text.

In the future, the model can be trained with large context-independent dataset which is expected to make recognition task accurate enough for use in automated Nepali-language systems.

9. ACKNOWLEDGEMENT

We would like to thank Department of Electronics and Computer Engineering, IOE for helping us undertake this research and allowing us to access the computing resources we used for training the model.

10. REFERENCES

[1] A. Graves, S. Fernandez, F. Gomez and J. Schmidhuber. 2006. Connectionist Temporal Classification: Labelling Unsegmented data with Recurrent Neural Networks. In ICML '06 Proc. of the Int. Conf. on International Conference on Machine Learning, Pittsburgh Pennsylvania USA

[2] E Hinton, Geoffrey & Osindero, Simon & Teh, Yee-Whye. 2006. A Fast Learning Algorithm for Deep Belief Nets. Neural computation, 18, pp. 1527-54.

[3] Bourlard, Herve A. and Morgan, Nelson. Connectionist Speech Recognition: A Hybrid Approach. Kluwer Academic Publishers, Norwell, MA, USA, 1993.

[4] G. E. Dahl, D. Yu, L. Deng and A. Acero. 2012. Context-Dependent Pre-Trained Deep Neural Networks for Large Vocabulary Speech Recognition. In Proc. IEEE Transactions on Audio, Speech and Language Processing, 20, pp. 30-42.

[5] A. Graves and N. Jaitly. 2014. Towards End-to-End Speech Recognition with Recurrent Neural Networks. In ICML 14 Proc. of the Int. Conf. on International Conference on Machine Learning, Beijing China

[6] A. Kalakheti, K. P. Bhattarari, S. Kuwar and S. Adhikari, Automatic Speech Recognition for Nepali Language. Tribhuvan University, Nepal

[7] B. Joshi, A. Gajurel, A. Pokhrel and M. K. Sharma. 2017. HMM Based Isolated Word Nepali Speech Recognition. In Intern. Conf. on Machine Learning and Cybernetics. Ningbo, China.

[8] S. Hochreiter and J. Schmidhuber. 1997. Long Short-Term Memory. Neural Computation, 9(8), pp. 1735-1780

[9] Hochreiter, Sepp. (1998). The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems. 6. 107-116.

[10] M. Schuster and K. K. Paliwal. 1997. Bidirectional Recurrent Neural Networks. IEEE Transactions on Signal Processing, 45.

[11] A. Graves, S. Fernandez and J. Schmidhuber. 2005. Bidirectional LSTM networks for improved phoneme classification and recognition. In Proceedings of the 2005 International Conference on Artificial Neural Networks. Warsaw, Poland.

[12] S. Magre, P. Janse, and R. Deshmukh. 2014. A Review on Feature Extraction and Noise Reduction Technique. International Journal of Advanced Research in Computer Science and Software Engineering

[13] The Python Tutorial, https://docs.python.org/3/tutorial/index.html

[14] Tensorflow, https://www.tensorflow.org