

Overview of Incorporating Nonlinear Functions into Recurrent Neural Network Models



Farzaneh Nikbakhtsarvestani

Abstract This paper describes training recurrent neural networks (RNNs) which are able to learn features and long-range dependencies from sequential data. Although training RNNs is mostly plagued by the vanishing and exploding gradient problems, but there is a RNN architecture so called long short-term memory (LSTM) to address these issues. To demonstrate the impact of nonlinearity of activation functions on training recurrent neural networks, the limitations of the LSTM algorithm on speech recognition technology are presented in this study.

1 Introduction

Recurrent neural networks (RNNs) which have attracted great attention have been widely studied from 1986 and were based on David Rumelhart's work for modeling time series. These networks are used for different machine learning tasks which can modelize sequential data.

Recurrent neural networks have many applications, especially when the input and output have variable lengths such as handwriting recognition, speech recognition, and image to text.

The layers of connected units called artificial neurons make artificial neural networks (ANNs). The shallow network of ANN includes an input layer, an output layer, and at most a hidden layer without a recurrent connection. Recurrent connections in ANNs make them recurrent neural networks (RNNs), and the increase of complexity of network depends on the number of layers. More number of layers or recurrent connections generally increases the depth of the network and empowers it to provide various levels of data representation and feature extraction, referred to as deep learning [1]. The difference between these networks and higher layer ones is related to their units. The structure of hidden states causes RNNs to store, remember, and process past complex signals for long time periods and works as the memory of the network and state of the hidden layer at a time which is conditioned on its previous

F. Nikbakhtsarvestani (✉)

Department of Science, University of Ontario Institute of Technology, Oshawa, ON, Canada
e-mail: farzaneh.nikbakht@ontariotechu.ca

state [2]. Mapping the input sequence at the current timestep to the output sequence which is prediction the sequence in the next timestep is the another capability of RNNs. Artificial neurons and the feedback loops which are recurrent cycles over time or sequence make RNNs as a class of supervised machine learning models [3]. In this model, for training the RNNs, the dataset of input-target pairs needs to be trained, and the goal is minimizing the difference between output and target pairs via optimizing the weights of the network. This work is based on training recurrent neural network [1] and error bound for approximations [4].

Definition 1.1 (*Neural Network*) Let $d, L \in \mathbb{N}$. A neural network U with input dimension d and L layers is a sequence of matrix-vector tuples

$$U = ((A_1, b_1), (A_2, b_2), \dots, (A_L, b_L))$$

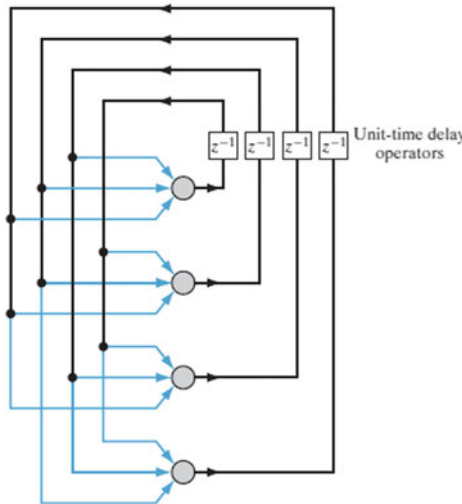
where $N_0 = d$ and $N_1, \dots, N_L \in \mathbb{N}$ and where

$$A_L := N_L \times \sum_{k=0}^{L-1} N_k,$$

and $b_L \in \mathbb{R}^{N_L}$. (Note that N_L is the definition of the output layer of U)

2 A Simple Recurrent Neural Network

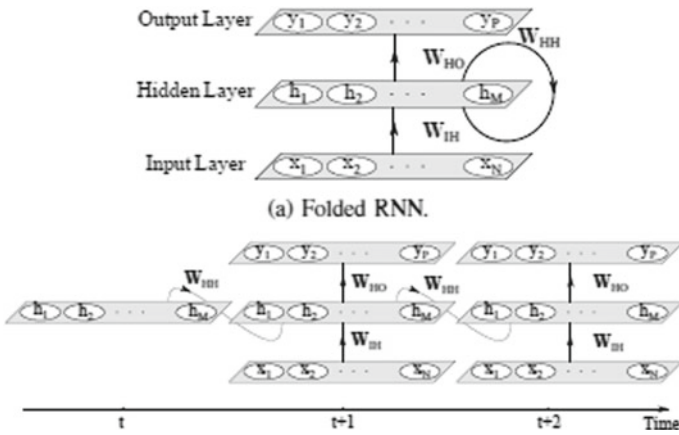
Recurrent cycles over time are called feedback loops. In the neural network literature, artificial neurons with one or more global feedback are referred to as recurrent networks [1]. RNNs are a class of supervised machine learning models. Learning capability of the RNNs and its performance depends on the amount of feedback loops.



Also where the output of a neuron is fed back into its own input, the network has self-feedback. Moreover in the situation that neural network contains nonlinear units, feedback loops include particular branches unit time delay operators (denoted by Z^{-1} in the figure).

2.1 Model Architecture

Recurrent neural network architectures can have many different forms. The layer of input, recurrent hidden, and layer of output are three layers in simple RNNs.



A sequence of vectors through time $\{\dots, x_{t-1}, x_t, x_{t+1}, \dots\}$ such that $x_t = \{x_1, x_2, \dots, x_N\}$ makes the input of input layer.

In a fully connected RNN, the input units are connected to the hidden units. In this layer, a weight matrix W_{IH} defines the connections. The hidden units $h_t = \{h_1, h_2, \dots, h_N\}$ cause the hidden layer to connect with recurrent connections through time. The stability and performance of the network depend on the initialization of the hidden units with small elements. The state space (memory) of the system is defined by hidden layer as

$$h_t = f_H(O_t) \tag{2.1}$$

such that $f_H(\cdot)$ is the activation function and

$$O_t = W_{IH}x_t + W_{HH}h_{t-1} + b_h$$

In the hidden units, b_h is the bias vector. In the third layer which is output layer, the units are computed as

$$y_t = f_O(W_{HO}h_t + b_0)$$

where $f_O(\cdot)$ is activation function and b_0 is the bias in this layer. Weighted connections W_{HO} connect the hidden layer to the output layer. A set of values which summarizes all the unique necessary information of the previous states of the network through

the time is hidden state of a RNN. These hidden states make accurate predictions at the output layer according to input vector. If a simple RNN is trained well, the network will be capable for modeling rich dynamics; however in every units, simple activation function is used [1].

2.2 Activation Function

In the output layer for training a classification model, an activation function is applied. The activation function must be continuous in order to meet differentiability requirements. Sigmoidal nonlinear functions are examples of a continuously differentiable nonlinear activation functions which are used in multi-layer perceptrons.

Some popular activation functions are rectified linear unit (ReLU) , “tanh”, and “sigmoid”. These functions (“tanh” and “sigmoid”) are two forms of sigmoidal nonlinear functions.

Both the nature of the machine learning problem and the training dataset are two important factors for choosing the proper activation function. The activation function typically takes the form of a hyperbolic tangent function which is defined as

$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$

or logistic(sigmoid) function which is known as common choice activation function

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

The domain of this function is real numbers, and its range is $[0, 1]$. These two activation functions which fastly saturate the neuron and cause the gradient to be vanished are related as

$$\sigma(x) = \frac{\tanh\left(\frac{x}{2}\right) + 1}{2}$$

Obviously, the scaled version of sigmoid activation function is tanh. The ReLU activation function which works on positive input values is defined as

$$\rho(x) = \max(x, 0)$$

Although comparison between ReLU activation function and another two activation functions indicates that in ReLU activation function, the acceleration of the convergence of stochastic gradient descent (SGD) is greater than tanh and sigmoid, but due to the lack of resistance of ReLU function against growing the weight matrix and the large gradient, neuron may be inactive by using of this type of activation function during training.

3 Training Recurrent Neural Network

Training the RNN in which the training loss being minimized is a main issue in such networks. Optimizing the algorithm in order to tune the weights and instantiating them are the main approaches used for minimizing the training loss. The main focus in optimizing the machine learning algorithm is on the convergence and reducing the complexity of training section of the algorithm which needs a large number of iterations. There are many approaches for training RNNs. In this paper, we study activation functions in gradient-based machine learning algorithms and their modified forms.

3.1 Gradient-Based Learning Methods

One of the most common approaches to optimize neural network is gradient descent (GD). Although this method causes the total loss to minimized, but for large datasets, this method is computationally expensive and is not appropriate for training the models as inputs arrive (i.e., online training). Basically in this way by computing the error function derivative with respect to each member of the weight matrices, the weights of the model are set. Assuming that the activation function is nonlinear and differentiable, in order to minimize the total loss, the gradient descent alters at each weight. In GD, each iteration of optimization for doing an update follows of this formula:

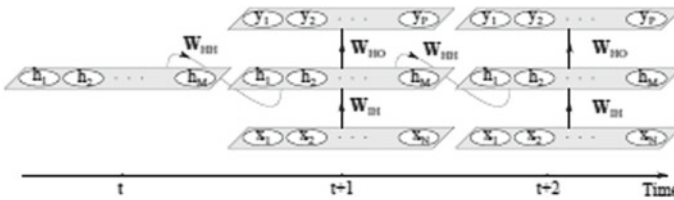
$$\theta_{t+1} = \theta_t - \frac{\lambda}{U} \sum_{k=1}^U \frac{\partial L_k}{\partial \theta}$$

where λ is the rate of training and U is the extent of training set and θ is the set of parameters. The gradient for whole dataset is computed by GD so GD is considered as batch GD. In other words, by GD method, we follow the direction of the slope of the surface created by the objective function downhill until we reach a valley [5]. Since the time is not considered in GD and RNNs include recurrent cycles over time, so this method does not work properly for training the network. For solving this problem, an extended version of GD through time is needed. It is called backpropagation through time (BPTT). Basically, backpropagation is a specific technique for implementing gradient descent in the weight space for a multi-layer perception [6]. In RNN, the connections between parameters and the dynamics are unstable, this makes computing error-derivatives through time complicated, and GD method is thus inadequate. Another shortcoming of GD is due to the difficulty in recognizing the dependencies as they increase in magnitude. The only parameter which is considered by loss function derivative with respect to weights is the distance between the updated output and its consistent target as the history of weights information is not applied. The vanishing gradient is another deficiency of applying GD method for training RNN. The exponential decay of backpropagated gradient

causes RNNs not to learn long-term temporal dependencies. In a reverse situation, GD method may lead to explode gradient issue which is due to exponentially blow-up of backpropagated gradient. This result is in unstable learning process. We are going to discuss these challenges and provide an architecture for solving these problems.

3.1.1 Backpropagation Through Time (BPTT)

The simplest type of neural network is the feedforward neural network. This network has no loop, and the information moves from input nodes, via hidden ones and finally output nodes (i.e., in only one direction). For feedforward networks, the method ‘‘BPTT’’ is used to train the network. BPTT is the generalization of backpropagation. In this method, the idea is making the network unfolded in time, and the signals of error propagate backwards through time [2].



In this network, the parameters can be considered as the set

$$\theta = \{W_{HH}, W_{IH}, W_{HO}, b_h, b_l, b_o\} \tag{3.1}$$

These parameters affect the loss function in the previous timesteps. The gradients of loss function with respect to this set are

$$\frac{\partial L}{\partial \theta} = \sum_{t=1}^T \frac{\partial L_T}{\partial \theta}$$

where for the loss function L , we have

$$\frac{\partial L_t}{\partial \theta} = \sum_{k=1}^t \left(\frac{\partial L_t}{\partial h_t} \cdot \frac{\partial h_t}{\partial h_k} \cdot \frac{\partial h_k^+}{\partial \theta} \right),$$

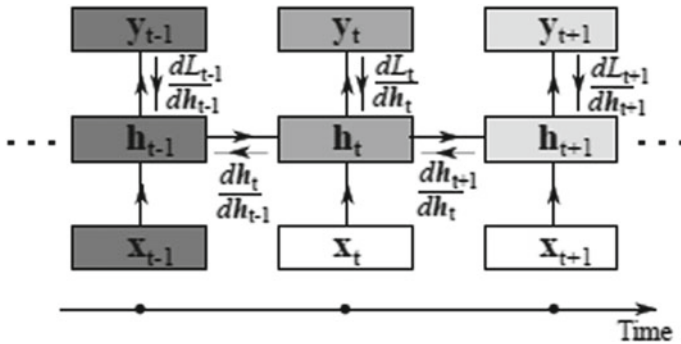
where h_t is the hidden state of network at time t and $\frac{\partial h_k^+}{\partial \theta}$ is the ‘‘immediate’’ partial derivative. For propagating the error signals backward in timesteps t and k which is $k < t$, we have

$$\frac{\partial h_t}{\partial h_k} = \prod_{i=k+1}^t \frac{\partial h_i}{\partial h_{i-1}},$$

and so according to Eq. (2.1) and using the Jacobian matrix for the hidden state we have:

$$\prod_{i=k+1}^t \frac{\partial h_i}{\partial h_{i-1}} = \prod_{i=k+1}^t W_{HH}^T \text{diag}|f'_H(h_{i-1})|.$$

As we see the participation of the hidden states in the network through time is obvious. In terms of the contribution of inputs and corresponding hidden states over time, two types of hidden state contribution are recognizable, such as long-term contribution (for time $k \ll t$) and short time contribution for another time.



By considering the above figure, it is evident that when the new input are admitted to the network, the sensitivity of units vanishes (decreasing the contribution of the inputs x_{t-1} through time), the activation in hidden units is overwritten by BPTT (increasing the contribution of the loss function value L_{t+1} w.r.t h_{t+1} in BPTT trough time) [1].

3.1.2 Vanishing Gradient Problem

The vanishing gradient problem causes some defects for RNNs. This is because of strong nonlinearity which is used for making complex pattern of data. When the gradient propagates back through time, its magnitude decreases exponentially. Subsequently, the long-term correlations are neglected by the network, which causes an issue in learning process of dependencies among distant events. There are two possible explanations for this:

1. The gradient of nonlinear functions which is close to zero.
2. While the gradient propagates back through time, recurrent matrix increases the gradient magnitude.

For the less than one eigenvalues of the recurrent matrix, after five to ten times of running backpropagation algorithm, the rate of convergence of gradient increases. In the RNNs learning process with extended sequences and small weights, the gradient shrinks as well. Long-term components explode in the recurrent weight matrix W_{HH}

when its spectral radius becomes more than 1 and $t \rightarrow \infty$. Because product of matrices can lead to shrinkage/explosion along several directions. In order to generalize this to nonlinear function $f'_h(\cdot)$ in Eq. (2.1), we can bound it with $\gamma \in \mathbb{R}$ s.t

$$\|\text{diag}(f'_H(h_k))\| \leq \gamma$$

And since we have

$$\prod_{i=k+1}^t \frac{\partial h_i}{\partial h_{i-1}} = \prod_{i=k+1}^t W_{\text{HH}}^T \text{diag}|f'_H(h_{i-1})|$$

so

$$\left\| \frac{\partial h_{k+1}}{\partial h_k} \right\| \leq \|W_{\text{HH}}^T\| \cdot \|\text{diag}(f'_H(h_k))\| \leq 1$$

Now for $\delta \in \mathbb{R}$, if we consider $\|\frac{\partial h_{k+1}}{\partial h_k}\| \leq \delta < 1$ for loss function component, we have

$$\left\| \frac{\partial L_t}{\partial h_t} \left(\prod_{i=1}^{t-1} \frac{\partial h_{i+1}}{\partial h_i} \right) \right\| \leq \delta^{t-k} \left\| \frac{\partial L_t}{\partial h_t} \right\|$$

in different timesteps. Since $\delta < 1$, increasing $t - k$ leads to vanishing gradient problem. Generally in recurrent matrix W_{HH} , if for the largest singular value λ_1 we have $\lambda_1 < \frac{1}{\gamma}$, the gradient vanishing problem happens.

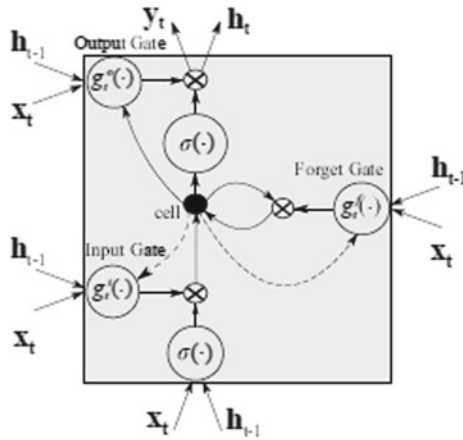
3.1.3 Exploding Gradient Problem

As its mentioned, the process of training RNNs with BPTT may be exposed by exploding problem. In training recurrent neural networks on long sequences, increasing the weights causes the norm of gradient to increase and gradients are subsequently exploded. This is a different necessary condition in comparison to the vanishing gradient problem in which for the largest singular value of recurrent matrix W_{HH} (i.e., λ_1), we have $\lambda_1 > \frac{1}{\gamma}$.

4 Long Short-term Memory

As mentioned before, the main shortcoming of BPTT method pertains to error signals flowing backwards in time. This causes gradients to vanish or explode through time which turns out to more difficulties in learning long-term dependencies. To tackle this problem, some methods have been proposed. One of the most successful techniques to strengthen the long-term dependencies is known to be the long short-term memory

(LSTM). In this method, the sigmoid or tanh hidden units are replaced with “memory cell”. This change leads to more controlled behavior of backpropagated gradients. In this approach, the input and output values of the cell memory are controlled by gates. Each cell is also matched with a forget gate that controls the decay rate of its stored values [7]. In this way, the memory cell holds its stored values during the periods that the input and output gates are off, and the forget gate is not causing decay [1]. Therefore, the gradient of the error with respect to its stored value, when backpropagated over those periods, stays constant [8]. Depending on the training application, there are varieties of LSTM structures developed by many researchers. In the following chapter, standard LSTM approach will be illustrated, and then, we will focus on bidirectional LSTM in particular to suit our applications.



4.1 Standard LSTM

As its shown in above figure, each typical memory cell has its own input, output, and forget gate and a cell activation component that provide continuous analogs of write, read, and reset operations for the cells. More precisely, the input, forget, and output gate are trainable to learn, respectively, what information to store in the memory, how long to store it, and when to read it out [9]. The activation of the cell is controlled by the designed multipliers. The input to the cells is multiplied by the activation of the input gate, the output to the net is multiplied by that of the output gate, and the previous cell values are multiplied by the forget gate. The net can only interact with the cells via the gates [10]. The input gate of LSTM is defined as

$$g_t^{\text{in}} = \sigma(W_{I_g^{\text{in}}}x_t + W_{H_g^{\text{in}}}h_{t-1} + W_{g_c^{\text{in}}}g_{t-1}^c + b_{g^{\text{in}}}) \tag{4.1}$$

where W_{\dots} is the weight matrix as below

- $W_{I_g^{\text{in}}}$: input layer \rightarrow input gate

- $W_{\text{Hg}^{\text{in}}}$: hidden state \rightarrow input gate
- $W_{\text{g}^{\text{c}}\text{g}^{\text{in}}}$: cell activation \rightarrow input gate
- $b_{\text{g}^{\text{in}}}$: bias of the input gate
- forget gate :

$$g_t^{\text{f}} = \sigma(W_{\text{Ig}^{\text{f}}}x_t + W_{\text{Hg}^{\text{f}}}h_{t-1} + W_{\text{g}^{\text{c}}\text{g}^{\text{f}}}g_{t-1}^{\text{c}} + b_{\text{g}^{\text{f}}}) \quad (4.2)$$

where

- $W_{\text{Ig}^{\text{f}}}$: input layer \rightarrow forget gate
- $W_{\text{Hg}^{\text{f}}}$: hidden state \rightarrow forget gate
- $W_{\text{g}^{\text{c}}\text{g}^{\text{f}}}$: cell activation \rightarrow forget gate
- $b_{\text{g}^{\text{f}}}$: bias of the forget gate
- cell gate:

$$g_t^{\text{c}} = g_t^{\text{in}} \tanh(W_{\text{Ig}^{\text{c}}}x_t + W_{\text{Hg}^{\text{c}}}h_{t-1} + b_{\text{g}^{\text{c}}}) + g_t^{\text{f}} g_{t-1}^{\text{c}} \quad (4.3)$$

where

- $W_{\text{Ig}^{\text{c}}}$: input layer \rightarrow cell gate
- $W_{\text{Hg}^{\text{c}}}$: hidden state \rightarrow cell gate
- $b_{\text{g}^{\text{c}}}$: bias of the cell gate
- output gate:

$$g_t^{\text{out}} = \sigma(W_{\text{Ig}^{\text{out}}}x_t + W_{\text{Hg}^{\text{out}}}h_{t-1} + W_{\text{g}^{\text{c}}\text{g}^{\text{out}}}g_t^{\text{c}} + b_{\text{g}^{\text{out}}}) \quad (4.4)$$

where

- $W_{\text{Ig}^{\text{out}}}$: input layer \rightarrow output gate
- $W_{\text{Hg}^{\text{out}}}$: hidden state \rightarrow output gate
- $W_{\text{g}^{\text{c}}\text{g}^{\text{out}}}$: cell activation \rightarrow output gate
- $b_{\text{g}^{\text{out}}}$: bias of the output gate
- hidden state:

$$h_t = g_t^{\text{out}} \tanh(g_t^{\text{c}}) \quad (4.5)$$

The LSTM gates can prevent the rest of the network from modifying the contents of the memory cells for multiple timesteps [1].

4.2 Bidirectional LSTM

In order to train data, looking at the previous context and future context is important and has many applications such as speech recognition Bidirectional RNN (BRNN) considers all available input sequence in both the past and future for estimation of the output vector [11]. To enhance the capability of BRNNs through stacking hidden layers of LSTM cells in space, deep bidirectional LSTM (BLSTM) can be applied. BLSTM networks are more powerful than unidirectional LSTM networks [1]. This

means that the bidirectional nets and the LSTM nets did not take significantly more time to train per epoch than the unidirectional or RNN [10]. During computation, BLSTM includes all information of input sequences. Like BRNN, BLSTM model can solve the vanishing gradient problem and extend the model. But biggest difference between BRNN and BLSTM is related to their training time. The convergence of BRNN is more than eight times as long respect to BLSTM.

We consider an extended LSTM layer in multi-layer net, and the pseudocode for the forward pass is described.

4.2.1 Notation

- S input sequence
- τ time
- $x_k(\tau)$ network input to unit k at the time τ
- $y_k(\tau)$ the activation of the network input
- $E(\tau)$ output error at the time τ
- $t_k(\tau)$ training target at output unit k at time τ
- N set of all units (input units, bias units)
- W_{ij} weight from unit i to unit j
- i input gate
- ϕ forget gate
- ω output gate
- c elements of the set of cells C
- s_c state value of cell c
- f is a function of gate
- g cell input function
- h output function

Note that for each memory block, the LSTM equations are written, and these calculations can be repeated for each block. The error gradient is calculated with online BPTT, i.e., after every sequence BPTT shrink to input sequence length with the weight updates [10].

4.2.2 Forward Pass

- Re-adapt the activation to 0,
- Feed in the inputs and update the activation functions. All hidden layer and output activation functions at every timestep need to be stored,
- The activation functions are updated as:
Input Gates

$$x_t = \sum_{j \in N} \omega_{ij} y_j(\tau - 1) + \sum_{c \in C} \omega_{ic} s_c(\tau - 1)$$

$$y_t = f(x_t)$$

Forget Gates

$$x_\phi = \sum_{j \in N} \omega_{\phi j} y_j(\tau - 1) + \sum_{c \in C} \omega_{\phi c} s_c(\tau - 1)$$

5 Application of LSTM in Speech Recognition

5.1 Speech Recognition

Since RNNs are the structure through time and the signals of speech and audio change continuously over time, so RNNs can be an ideal model to learn features. Also speech recognition prediction uses the past and future sequential data, so BRNN is suitable in this field. Later applications of the connectionist temporal classification (CTC) function contributed to promote the RNNs in speech recognition. Connectionist temporal classification (CTC) is an objective function that allows an RNN to be trained for sequence transcription tasks without requiring any prior alignment between the input and target sequences [12]. CTC model has iterations like the sequence transducer and neural transducer. This property enables a second RNN to perform as a language model. This eventually leads to do the task such as online speech recognition. So by these argumentation, based on linguistic feature and prior transcriptions, the model can make the prediction.

5.2 Speech Emotion

Another application of RNNs is speech emotion. In this field, the segment of speech is organized as an emotion. Since in speech emotion recognition, the progress proceeds from the same way as that of speech recognition so the speech emotion recognition is much the same to speech recognition. Several methods have been proposed in speech application such as hidden Markov model (HMMs) and Gaussian mixture models (GMMs). With RNNs establishment, the trend of learning has improved. Because the networks were able to learn the features on their own. So RNN models have been applied for performing speech emotion recognition. LSTM-RNN has been successfully applied to speech recognition. Because in LSTM network, long-range dependencies are modeled better in order to capture the emotions. Also deep bidirectional LSTMs can capture more data through taking them in large number of frames.

5.3 *Speech Synthetic*

Another type of speech application is speech synthetic. In this field, long-term sequence learning is needed as well. HMM-based models and deep MLP neural networks can synthesize speech. However, these models have some problems. For example, in HMM-based models, statistical averaging during the training phase leads to overly smooth trajectories so the sound is not natural or MLP neural network takes each frame as an independent entity from its neighbors and fails to take into account the sequential nature of speech [13]. Introducing the RNNs in speech synthesis collaborates to leverage the sequential dependencies.

Speech synthesis also requires long-term sequence learning. HMM-based models can often produce synthesized speech, which does not sound natural. This is due to the overly smooth trajectories produced by the model, as a result of statistical averaging during the training phase [13].

Following that, LSTM performs better than RNNs. Also the ability of BLSTM model to integrate the relationship with neighboring frames in both future and past time steps [14, 15] make this model very effective in learning long-term sequential dependencies.

Acknowledgements The author would like to thank the University of Ontario Institute of Technology and the University of Manitoba for their support with this study.

References

1. Salehinejad, H., Sankar, S., Barfett, J., Colak, E., Valaee, S.: Recent Advances in Recurrent Neural Networks. [arXiv:1801.01078v3](https://arxiv.org/abs/1801.01078v3) [cs.NE, 22 Feb 2018]
2. Mikolov, T., Joulin, A., Chopra, S., Mathieu, M., Ranzato, M.: Learning longer memory in recurrent neural networks. [arXiv:1412.7753](https://arxiv.org/abs/1412.7753) (2014)
3. Haykin, S.: *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR (1994)
4. Yarotsky, D.: Error bounds for approximations with deep ReLU networks. *Neural Netw.* **94**, 103–114 (2017)
5. Ruder, S.: An overview of gradient descent optimization algorithms. [arXiv:1609.04747](https://arxiv.org/abs/1609.04747) (2016)
6. Sutskever, I., Martens, J., Dahl, G., Hinton, G.: On the importance of initialization and momentum in deep learning. In: *International Conference on Machine Learning*, pp. 1139–1147 (2013)
7. Gers, F.A., Schmidhuber, J., Cummins, F.: Learning to forget: continual prediction with LSTM (1999)
8. Le, V., Jaitly, N., Hinton, G.E.: A simple way to initialize recurrent networks of rectified linear units. [arXiv:1504.00941](https://arxiv.org/abs/1504.00941) (2015)
9. Graves, A., Schmidhuber, J.: Framewise phoneme classification with bidirectional LSTM and other neural network architectures. In: *IJCNN 2005 Conference Proceedings*. Published Under the IEEE Copyright (2014)
10. Graves, A., Schmidhuber, J.: Framewise phoneme classification with bidirectional LSTM and other neural network architectures *Neural Networks*, vol. 18, no. 5, pp. 602–610 (2005)
11. Schuster, M., Paliwal, K.K.: Bidirectional recurrent neural networks. **45**(11), 2673–2681 (1997)
12. Graves, A., Fernandez, S., Gomez, F., Schmidhuber, J.: Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. in: *Proceedings of the 23rd International Conference on Machine Learning*. ACM, pp. 369–376 (2006)

13. Fan, K., Wang, Z., Beck, J., Kwok, J., Heller, K.A.: Fast second order stochastic backpropagation for variational inference. In: *Advances in Neural Information Processing Systems*, pp. 1387–1395 (2015)
14. Fan, Y., Qian, Y., Xie, F.-L., Soong, F.K.: TTs synthesis with bidirectional LSTM based recurrent neural networks. In: *Fifteenth Annual Conference of the International Speech Communication Association* (2014)
15. Fernandez, R., Rendel, A., Ramabhadran, B., Hoory, R.: Prosody contour prediction with long short-term memory, bi-directional, deep recurrent neural networks. In: *Interspeech*, pp. 2268–2272 (2014)