Introduction to Sequence Learning Models: RNN, LSTM, GRU

Sakib Ashraf Zargar

Department of Mechanical and Aerospace Engineering, North Carolina State University, Raleigh, North Carolina 27606, USA

Table of Contents

1. Introduction to Sequence Learning

Sequences consist of data points that can be meaningfully ordered such that observations at one location in the sequence provide useful information about observations at other location(s). Countless supervised learning tasks require dealing with sequence data and depending on the nature of the input and output, one of the following three situations can arise in a sequence learning problem:

1. The input is a sequence while the target output is a single data point – Examples include video activity recognition, sentiment classification, stock price prediction, etc.
2. The input is a single data point while the target output is a sequence – Examples include image captioning, music generation, speech synthesis, etc.
3. Both the input and the target output are sequences – Examples include speech recognition, natural language translation, DNA sequence analysis, name entity recognition, etc. It is worth mentioning that the input and output sequences may be of same or different lengths.

In the most general case (when both the input and output are sequences), a training example can be represented as follows

$$(\boldsymbol{x}, \boldsymbol{y}) = \left(\left[\boldsymbol{x}^{<1>}, \boldsymbol{x}^{<2>}, \cdots, \boldsymbol{x}^{<t>}, \cdots, \boldsymbol{x}^{<T_x>} \right], \left[\boldsymbol{y}^{<1>}, \boldsymbol{y}^{<2>}, \cdots, \boldsymbol{y}^{<t>}, \cdots, \boldsymbol{y}^{<T_y>} \right] \right)$$

where $\boldsymbol{x}$ and $\boldsymbol{y}$ denote the input and output/target sequence respectively. Each data point $\boldsymbol{x}^{<t>}$ in the input sequence can be a scaler (discrete values in a measured sensor signal, daily value in a stock price history, etc.) or a real valued vector (frames in a video, words in a sentence, etc.). Similarly, each data point $\boldsymbol{y}^{<t>}$ in the output sequence can be a scalar or a real valued vector. The length of

the input and the output sequence can be same ($T_x = T_x$) or different ($T_x \neq T_x$). Standard feed-forward neural networks/multi-layer perceptrons (MLPs) have two main limitations which make them unsuitable for handling such learning tasks [1]:

1. Standard feed-forward neural networks rely on the assumption of independence among data points. If each data point is generated independently, this presents no problem, however, if the data points are related in time (frames in a video) or space (words in a sentence), this becomes unacceptable. In other words, this simply means that standard feed-forward neural networks are sequence-agnostic and have no explicit way of capturing context in sequences.
2. Standard feed-forward neural networks generally rely on examples (training/test) being vectors of fixed length which is not usually the case with sequence learning problems. As an example, for natural language translation (say English to French), it is highly unreasonable to expect all the sentences to be of the same length.

Just as convolutional neural networks (CNNs) are designed specifically for processing a grid of values (such as an image), recurrent neural networks (RNNs) are designed for processing sequences. RNNs are connectionist models that capture the dynamics of sequences via cycles in the network of nodes. Unlike standard feed-forward neural networks/MLPs, recurrent networks retain a state that can represent information from an arbitrarily long context window. As a result, while an MLP can only map from input to output vectors, an RNN can in principle map from the entire history of previous inputs to each output. In fact, the equivalent result to the universal approximation theory for MLPs is that an RNN with a sufficient number of hidden units can approximate any measurable sequence-to-sequence mapping to arbitrary accuracy [2]. The key point is that the recurrent connections allow a memory of previous inputs to persist in the network's internal state, and thereby influence the network output [3].

The origin of RNNs dates back to the 1980s. Hopfield [4], in 1982, introduced a family of recurrent neural networks that had pattern recognition capabilities. Hopfield networks are useful for recovering a stored pattern from a corrupted version and are the forerunners of Boltzmann machines and auto-encoders. In 1986, Jordan [5] used a feedforward neural network with a single hidden layer extended with special units called state units for supervised learning on sequences. Output node values are fed to the special units, which then feed these values to the hidden nodes at the following time step. This allows the network to remember actions taken at previous time steps. The architecture introduced by Elman [6] in 1990 was simpler than the earlier Jordan architecture. Associated with each unit in the hidden layer is a context unit. Each such unit takes as input the state of the corresponding hidden node at the previous time step, along an edge of fixed weight. This value then feeds back into the same hidden node along a standard edge. This architecture is equivalent to a simple RNN in which each hidden node has a single self-connected recurrent edge. The idea of fixed-weight recurrent edges that make hidden nodes self-connected is fundamental in subsequent work on LSTM networks.

This chapter starts by introducing the basic RNN architecture along with its limitations. Next, long short-term memory (LSTM), gated recurrent unit (GRU), and bidirectional recurrent neural

network (BRNN) are introduced which are variants of the basic RNN designed specifically for addressing these limitations and represent the state-of-the-art for sequence modelling.

2. Recurrent Neural Network (RNN)

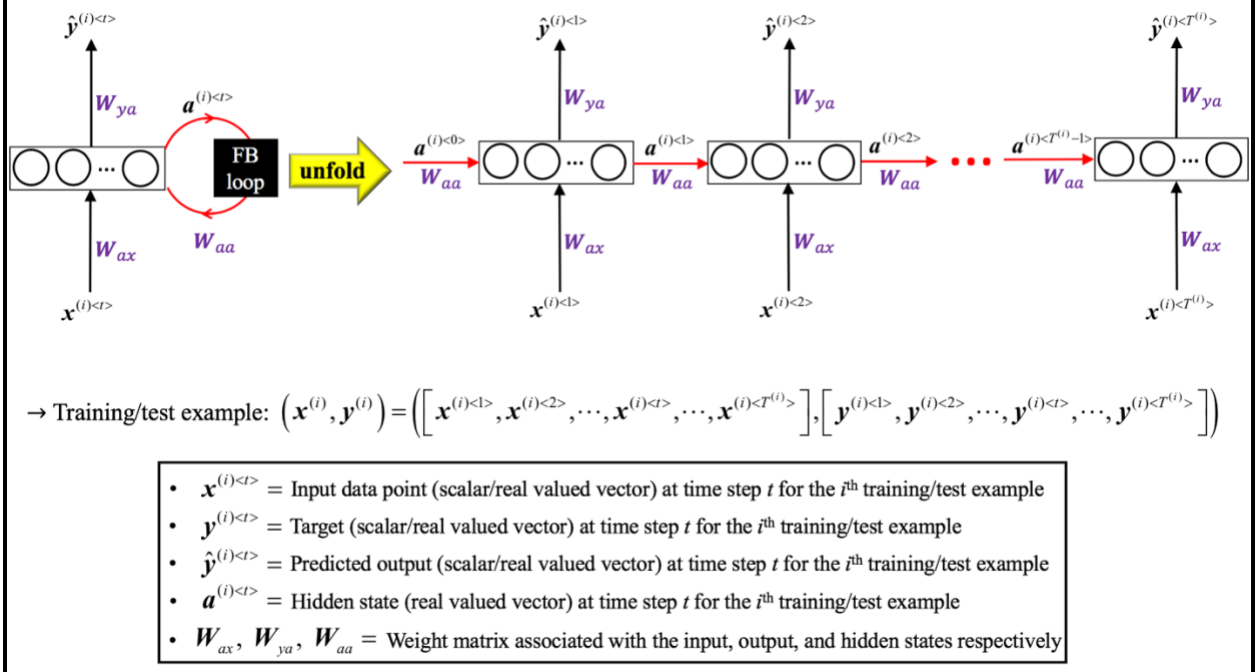


Figure 1: A standard recurrent neural network (RNN) having a single hidden layer in its compact and unfolded form.

Figure 1 shows a recurrent neural network (RNN) with a single hidden layer in its basic form. It consists of a single set of input, hidden, and output units, with the hidden units feeding into themselves through feedback (FB) loops. In simple words, as is clear from its unfolded version, an RNN can be thought of as multiple copies (in time) of the same network, each passing a message to its successor. The trainable parameters for an RNN include the weights and biases that are shared between all time-steps. Eqs. 1 and 2 specify all the computations at each time step during the forward pass of an RNN

$$\begin{aligned} \boldsymbol{a}^{<t>} &= \psi_1\left(\boldsymbol{W}_{ax}\boldsymbol{x}^{<t>} + \boldsymbol{W}_{aa}\boldsymbol{a}^{<t-1>} + \boldsymbol{b}_a\right) \\ &= \psi_1\left(\begin{bmatrix}\boldsymbol{W}_{ax} & \boldsymbol{W}_{aa}\end{bmatrix}\begin{bmatrix}\boldsymbol{x}^{<t>} \\ \boldsymbol{a}^{<t-1>}\end{bmatrix} + \boldsymbol{b}_a\right) \\ &\equiv \psi_1\left(\boldsymbol{W}_a\left[\boldsymbol{x}^{<t>}; \boldsymbol{a}^{<t-1>}\right] + \boldsymbol{b}_a\right) \end{aligned} \tag{1}$$

$$\hat{\boldsymbol{y}}^{<t>} = \psi_2\left(\boldsymbol{W}_{ya}a^{<t)} + \boldsymbol{b}_y\right) \equiv \psi_2\left(\boldsymbol{W}_y a^{<t)} + \boldsymbol{b}_y\right) \tag{2}$$

where the hidden state $\boldsymbol{a}^{<t>}$ computed at time step t, of which the predicted output $\hat{\boldsymbol{y}}^{<t>}$ is a filtered version, utilizes the current data point $\boldsymbol{x}^{<t>}$ in the input sequence as well as the hidden state $\boldsymbol{a}^{<t-1>}$ from the previous time step. It simply means that the predicted output at a particular time step depends not only on the current input in the sequence, but also on the past information. $\boldsymbol{W}_{ax}$, $\boldsymbol{W}_{aa}$, $\boldsymbol{W}_{ya}$ are the weights associated with the input, hidden, and output states respectively. For ease of notation, $\boldsymbol{W}_{ax}$ and $\boldsymbol{W}_{aa}$ are concatenated into a single matrix $\boldsymbol{W}_a$ and the subscript a in $\boldsymbol{W}_{ya}$ is dropped. $\boldsymbol{b}_a$ and $\boldsymbol{b}_y$ are the biases associated with the computation of $\boldsymbol{a}^{<t>}$ and $\hat{\boldsymbol{y}}^{<t>}$ respectively. It is common to use the activation function *tanh*() for ψ_1, however the choice of ψ_2 depends on the nature of the output (*sigmoid*, *softmax*, *relu*).

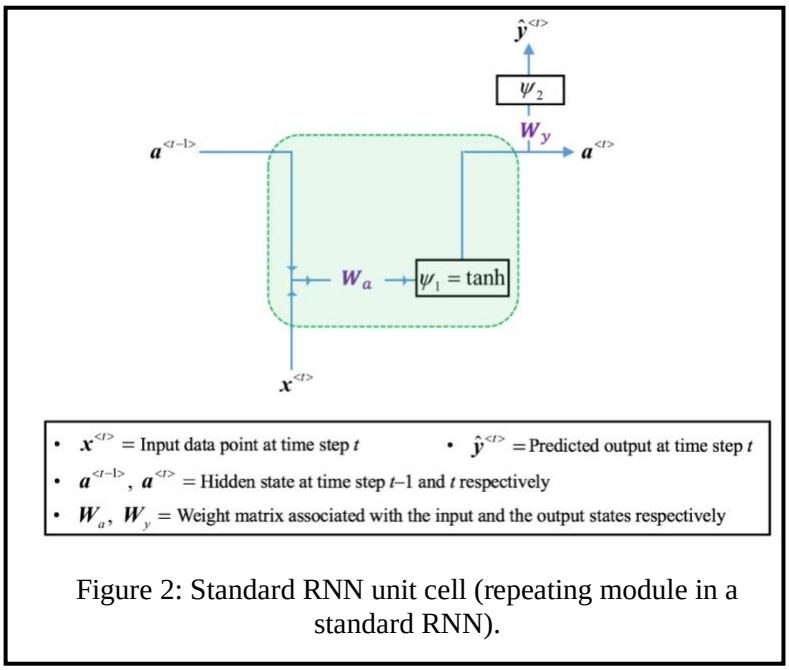


Figure 2: Standard RNN unit cell (repeating module in a standard RNN).

Figure 2 shows the computations at a single time step in an RNN in a pictorial manner. This will henceforth be referred to as a basic RNN unit cell. The unfolded version of the RNN is then simply obtained by stacking the individual unit cells together along the time axis as shown in Figure 3. Figures 1 and 3 represent an RNN architecture that is suited for problems where the length of the input and output sequences is the same. For a lot of sequence problems, this, however, may not be the case. Figure 4 gives a few network architectures that cover such cases.

For training an RNN, backpropagation through time (BPTT), introduced by Werbos [8] in 1990, is employed which essentially is the ordinary backpropagation algorithm applied on the unfolded network with weight sharing taken into account. However, when backpropagating errors across multiple time steps, RNNs generally suffer from the problem of vanishing and exploding gradients due to which they have a hard time learning long-term dependencies [9][10]. Intuitively, the phenomenon can be explained as follows: During backpropagation, the gradients have to go through continuous matrix multiplications (because of the chain rule) while they are being propagated back in time. As against deep feed-forward neural networks, this is problematic in RNNs as each layer essentially has the same weights (imagine multiplying a scaler weight w by itself many times, the product will either vanish or explode depending on the magnitude of w. For non-recurrent, feed-forward networks, different layers have different weights associated with them

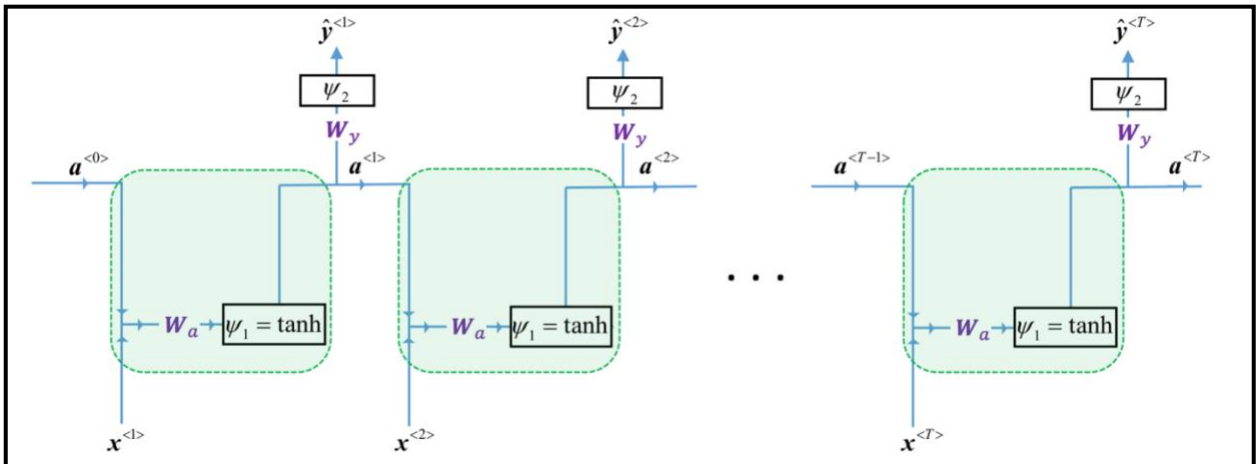
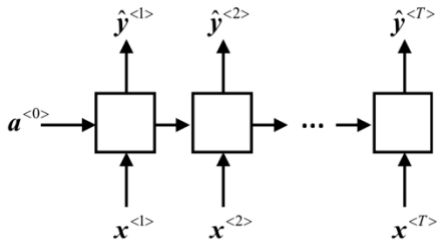
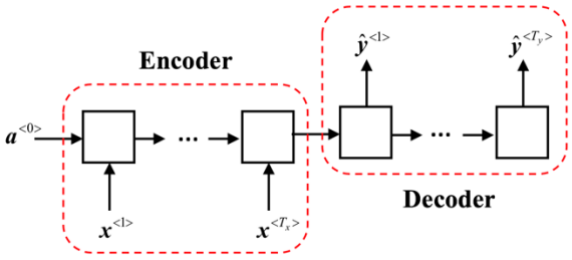


Figure 3: Unfolded version of a standard RNN with one hidden layer obtained by stacking individual RNN unit cells together along the time axis.



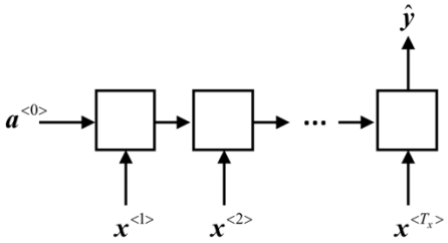
Many-to-many architecture ($T_x = T_y$)

(DNA sequence analysis, name entity recognition)



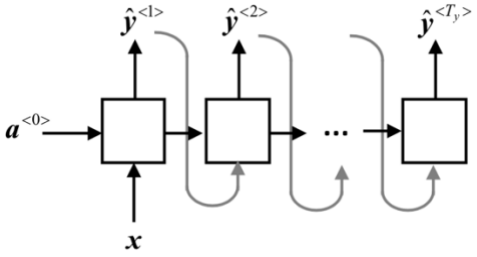
Many-to-many architecture ($T_x \neq T_y$)

(speech recognition, natural language translation)



Many-to-one architecture

(video activity recognition, sentiment classification)



One-to-many architecture

(image captioning, music generation)

Figure 4: Different RNN architectures that cater to different types of sequence problems.

which makes the situation different). While exploding gradients can lead to the crashing of the model, a direct consequence of vanishing gradients is that the information at earlier time steps cannot be effectively utilized while making predictions at later stages, or in other words, information at earlier time steps cannot be retained effectively over long term. It does not mean

that it is impossible to learn, but that it might take a very long time to learn long-term dependencies as the gradient of a long term interaction has exponentially smaller magnitude than the gradient of a short term interaction, and as such, the signal about these dependencies will tend to be hidden by the smallest fluctuations arising from short-term dependencies [11].

The exploding gradient problem can be circumvented by techniques such as gradient clipping [12] or by specifying some maximum number of time steps along which the error is propagated – an algorithm referred to as truncated backpropagation through time (TBPTT) [13]. The vanishing gradients problem is however more serious and requires fundamental changes to the basic RNN architecture. Next, long short-term memory (LSTM) and gated recurrent unit (GRU) networks are introduced which are variants of the basic RNN and are designed specifically for addressing the issue of vanishing gradients or in other words for making it easy to remember information over long periods of time (until it is needed).

3. Long Short-Term Memory (LSTM)

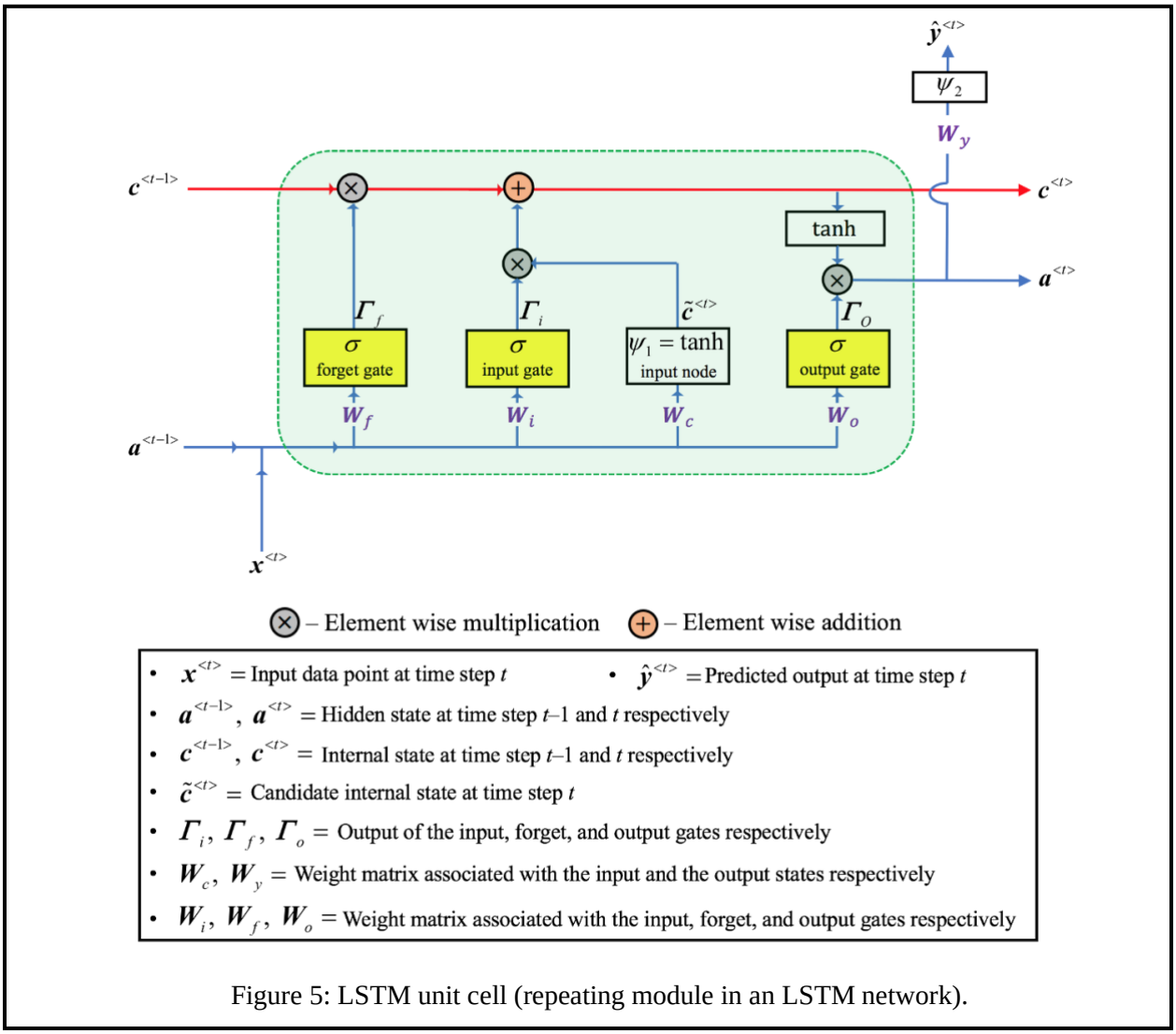


Figure 5: LSTM unit cell (repeating module in an LSTM network).

In order to overcome the problem of vanishing gradients, Hochreiter and Schmidhuber [14] in 1997 introduced the LSTM network. It resembles a standard RNN, however, the ordinary RNN unit cell is replaced by a memory cell as shown in Figure 5. Since the original LSTM network was introduced in 1997, several variations have been proposed [15][16]. What is presented here is the most commonly used LSTM architecture. At the heart of it, an LSTM unit cell has two distinct attributes: (1) the internal state (2) the gates.

The internal state $\boldsymbol{c}$ (also referred to as the cell state) can be thought of as the memory of the LSTM and is associated with an edge that spans adjacent time steps with a fixed unit weight. As the weight remains constant, error can flow across multiple time steps without vanishing or exploding. Intuitively, the internal state is like a conveyor belt that runs straight down the entire chain with only minor linear interactions which makes it is feasible for information to flow unchanged across multiple time steps [17]. The value of the internal state is regulated through two gates (input and forget). At each time step *t*, the input node ψ_1 takes the current data point $\boldsymbol{x}^{<t>}$ and the hidden state $\boldsymbol{a}^{<t-1>}$ from the previous time step as input and generates a candidate internal state $\tilde{\boldsymbol{c}}^{<t>}$ according to Eq. 3

$$\tilde{\boldsymbol{c}}^{<t>} = \tanh\left(\boldsymbol{W}_c\left[\boldsymbol{x}^{<t>};\boldsymbol{a}^{<t-1>}\right]+\boldsymbol{b}_c\right) \tag{3}$$

where $\boldsymbol{W}_c$ and $\boldsymbol{b}_c$ are the weights and biases associated with the input node. Typically, *tanh*() is the activation function used for ψ_1, although in the original LSTM paper [14], the activation function used was *sigmoid*. The influence of the computed candidate internal state $\tilde{\boldsymbol{c}}^{<t>}$ on the current state $\boldsymbol{c}^{<t-1>}$ is regulated through the gates (input and forget) and depends the significance of the current data point $\boldsymbol{x}^{<t>}$ in the sequence for long-term learning.

The function of the gates in an LSTM is to control the flow of information thereby regulating the internal state and the output of the memory cell. The gates (input, forget, and output) are sigmoid units that take as input the current data point $\boldsymbol{x}^{<t>}$ and the hidden state $\boldsymbol{a}^{<t-1>}$ from the previous time step and output a vector with values ranging between zero and one. Eq. 4 gives the computations for each gate at time step *t*

$$\begin{aligned}
\boldsymbol{\Gamma}_i &= \sigma\left(\boldsymbol{W}_i\left[\boldsymbol{x}^{<t>};\boldsymbol{a}^{<t-1>}\right]+\boldsymbol{b}_i\right)\\
\boldsymbol{\Gamma}_f &= \sigma\left(\boldsymbol{W}_f\left[\boldsymbol{x}^{<t>};\boldsymbol{a}^{<t-1>}\right]+\boldsymbol{b}_f\right)\\
\boldsymbol{\Gamma}_o &= \sigma\left(\boldsymbol{W}_o\left[\boldsymbol{x}^{<t>};\boldsymbol{a}^{<t-1>}\right]+\boldsymbol{b}_o\right)
\end{aligned} \tag{4}$$

where $\boldsymbol{W}_i$, $\boldsymbol{W}_f$, and $\boldsymbol{W}_o$ are the weights and $\boldsymbol{b}_i$, $\boldsymbol{b}_f$, and $\boldsymbol{b}_o$ are the biases associated with the input, forget, and output gates respectively. It should be noted that the dimensions of the internal state $\boldsymbol{c}^{<t-1>}$, the generated candidate internal state $\tilde{\boldsymbol{c}}^{<t>}$ and the gate outputs $\boldsymbol{\Gamma}$ are the same. A gate is so-called because its value is used to multiply the value of some other node: if its value is zero, then flow from the other node is cut off (reset) and if the value of the gate is one, all flow is passed through (retained). Through training, the gates can learn which data in the sequence is important

for long-term learning and thus should influence the internal state of the memory cell (or simply which data points should be stored in the memory of the LSTM).

The updated internal state $\boldsymbol{c}^{<t>}$ is then computed according to the following equation

$$\boldsymbol{c}^{<t>} = \left(\boldsymbol{\Gamma}_i * \tilde{\boldsymbol{c}}^{<t>}\right) + \left(\boldsymbol{\Gamma}_f * \boldsymbol{c}^{<t-1>}\right) \tag{5}$$

where $\boldsymbol{\Gamma}_i$ and $\boldsymbol{\Gamma}_f$ are the outputs of the input and forget gates respectively and $*$ denotes the element wise multiplication. At time step *t*, a value of 1 (vector of ones) for $\boldsymbol{\Gamma}_i$ and a value of 0 (vector of zeros) for $\boldsymbol{\Gamma}_f$ indicates that the cell state $\boldsymbol{c}^{<t-1>}$ will be updated to assume the value $\tilde{\boldsymbol{c}}^{<t>}$. This means that the current data point $\boldsymbol{x}^{<t>}$ is important for long-term learning and needs to be retained in the LSTM memory. On the other hand, a value of 0 (vector of zeros) for $\boldsymbol{\Gamma}_i$ and a value of 1 (vector of ones) for $\boldsymbol{\Gamma}_f$ indicates that the cell state $\boldsymbol{c}^{<t-1>}$ remains unchanged. This means that the current data point $\boldsymbol{x}^{<t>}$ is not important for long-term learning and does not need to be retained across multiple time steps.

The hidden state and the output of the memory cell at time step *t* are filtered versions of the internal state and are computed as follows

$$\begin{aligned} \boldsymbol{a}^{<t>} &= \boldsymbol{\Gamma}_o * \tanh\left(\tilde{\boldsymbol{c}}^{<t>}\right) \\ \hat{\boldsymbol{y}}^{<t>} &= \psi_2\left(\boldsymbol{W}_y \boldsymbol{a}^{<t>} + \boldsymbol{b}_y\right) \end{aligned} \tag{6}$$

where $\boldsymbol{W}_y$ and $\boldsymbol{b}_y$ are the weights and biases associated with the output. Most state-of-the-art results for sequence modelling are obtained using LSTM networks and their variants. Next, a gated recurrent unit (GRU) network is presented which is one such variant of LSTM and has been gaining a lot of popularity in the recent years. In recent years, LSTMs have shown a lot of promise across a wide variety of disciplines including structural health monitoring (SHM) [18][19][20][21][22][23][24][25][26].

4. Gated Recurrent Unit (GRU)

The GRU is a simpler variant of the LSTM and was introduced in 2014 by Cho et al. [27] and Chung et al. [28]. The internal structure of GRU is simpler as compared to the LSTM and is easier to train as fewer computations are involved. These simplifications are achieved primarily in two ways: (1) The input and the forget gates are combined into a single gate referred to as the update gate (2) The internal/cell state and the hidden state are merged. Figure 6 shows the GRU unit cell and the computations involved as given in Eq. 7.

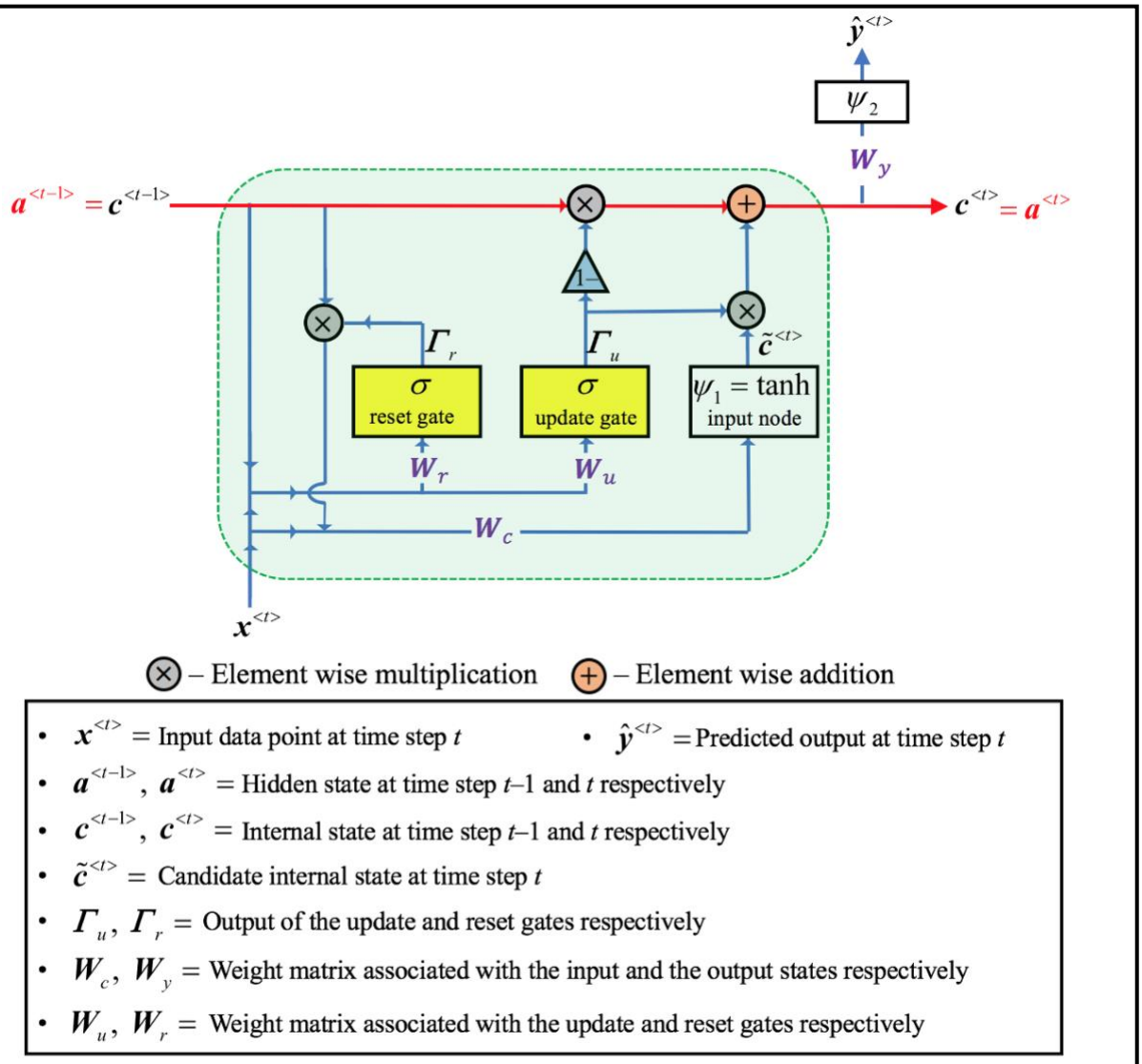


Figure 6: GRU unit cell (repeating module in a GRU network).

$$
\begin{aligned}
\tilde{\boldsymbol{c}}^{<t>} &= \tanh\left(\boldsymbol{W}_c\left[\boldsymbol{x}^{<t>}; \boldsymbol{\Gamma}_r * \boldsymbol{c}^{<t-1>}\right] + \boldsymbol{b}_c\right) \\
\boldsymbol{\Gamma}_u &= \sigma\left(\boldsymbol{W}_u\left[\boldsymbol{x}^{<t>}; \boldsymbol{c}^{<t-1>}\right] + \boldsymbol{b}_u\right) \\
\boldsymbol{\Gamma}_r &= \sigma\left(\boldsymbol{W}_r\left[\boldsymbol{x}^{<t>}; \boldsymbol{c}^{<t-1>}\right] + \boldsymbol{b}_r\right) \\
\boldsymbol{c}^{<t>} &= \left(\boldsymbol{\Gamma}_u * \tilde{\boldsymbol{c}}^{<t>}\right) + \left(1 - \boldsymbol{\Gamma}_u\right) * \boldsymbol{c}^{<t-1>} \\
\hat{\boldsymbol{y}}^{<t>} &= \psi_2\left(\boldsymbol{W}_y \boldsymbol{c}^{<t>} + \boldsymbol{b}_y\right)
\end{aligned}
\tag{7}
$$

To consolidate the discussion so far, Figure 7 shows the block diagram of a sequence learning model in the unfolded form (the input and output sequences are of the same length). Each cell in the figure can be a standard RNN, an LSTM, or a GRU unit cell. One important limitation of such a network is that it is unidirectional i.e., apart from the current input, the output at a particular time step depends only the past information in the input sequence. In certain situations, however, it may be beneficial to look at not only the past, but also the future to make the predictions. This can be accomplished using a bidirectional network which is presented next.

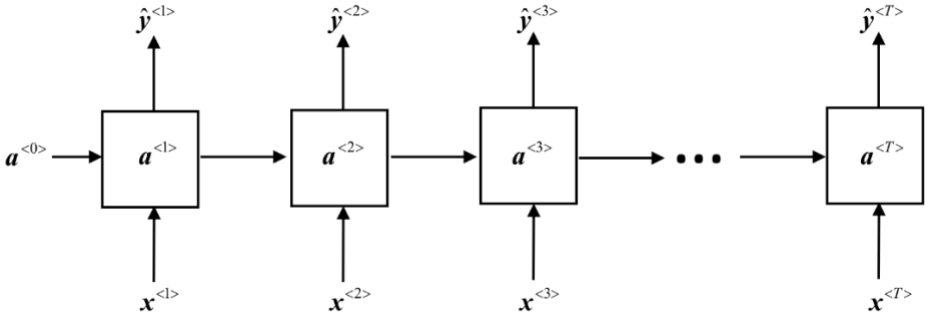


Figure 7: A unidirectional sequence learning model with one hidden layer in the unfolded form.

5. Bidirectional Sequence Learning Model

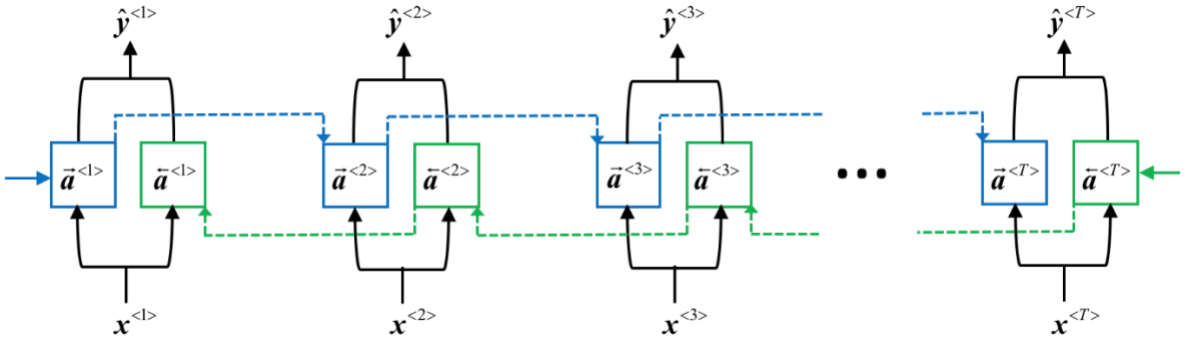


Figure 8: A bidirectional sequence learning model with one hidden layer in the unfolded form.

To overcome the limitations of a regular (unidirectional) RNN, a bidirectional recurrent neural network (BRNN) was proposed by Schuster and Paliwal [29] in 1997. Apart from the current input, the output at a particular time step utilizes information both from the past and the future. This is accomplished by training the network simultaneously in the positive and negative direction. For that, the neurons of a regular RNN are split into a part that is responsible for the positive direction and a part for the negative direction. Outputs from positive neurons are not connected to negative neurons and vice versa. This leads to the general structure that can be seen in Figure 8. The computations involved are given in Eq. 8.

$$\vec{\boldsymbol{a}}^{<t>} = \psi_1\left(\boldsymbol{W}_{\vec{a}}\left[\boldsymbol{x}^{<t>}; \boldsymbol{a}^{<t-1>}\right] + \boldsymbol{b}_{\vec{a}}\right)$$
$$\overleftarrow{\boldsymbol{a}}^{<t>} = \psi_1\left(\boldsymbol{W}_{\overleftarrow{a}}\left[\boldsymbol{x}^{<t>}; \boldsymbol{a}^{<t+1>}\right] + \boldsymbol{b}_{\overleftarrow{a}}\right) \quad (8)$$
$$\hat{\boldsymbol{y}}^{<t>} = \psi_2\left(\boldsymbol{W}_y\left[\vec{\boldsymbol{a}}^{<t>}; \overleftarrow{\boldsymbol{a}}^{<t>}\right] + \boldsymbol{b}_y\right)$$

where $\vec{\boldsymbol{a}}^{<t>}$ and $\overleftarrow{\boldsymbol{a}}^{<t>}$ are the activations calculated during the positive and negative directions respectively. As with the unidirectional model, each cell in Figure 8 can be a standard RNN, an LSTM, or a GRU unit cell. One limitation of the bidirectional sequence model is that the entire input sequence must be known before predictions could be made.

6. Deep RNN/LSTM/GRU

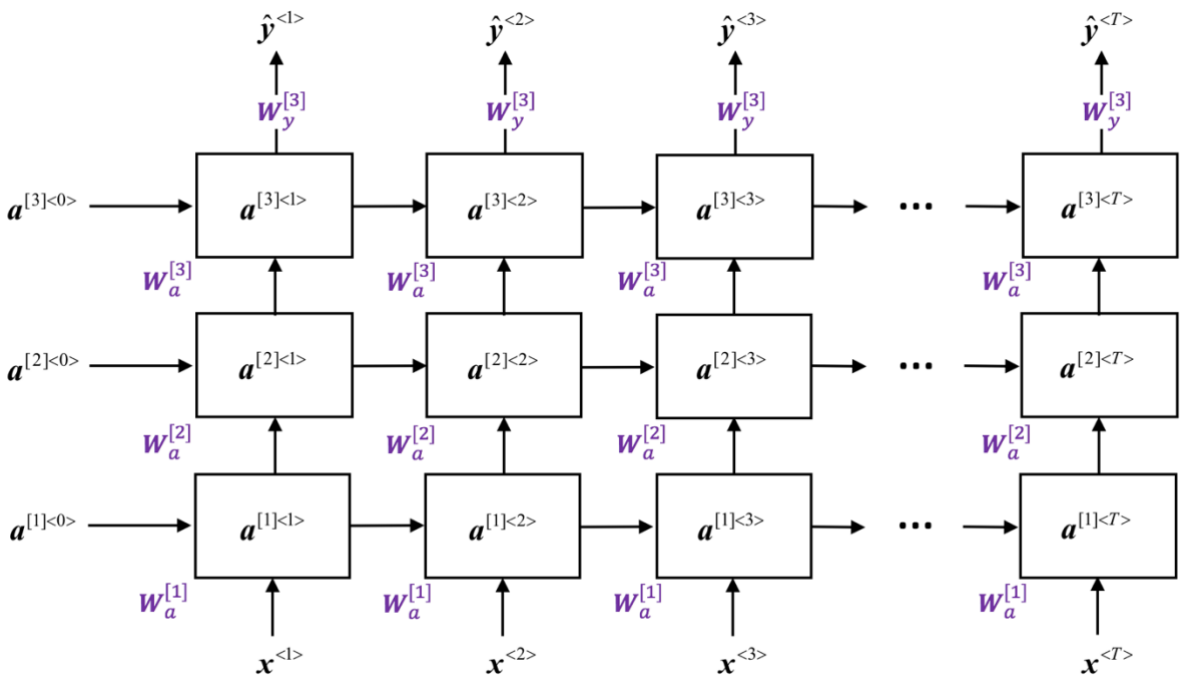


Figure 9: A deep unidirectional sequence learning model with three hidden layers in the unfolded form.

Experimental evidence suggests that for learning complex mappings, it is useful to stack multiple layers together to get deeper versions of these models [30]. Figure 9 shows a unidirectional sequence learning model with three hidden layers in the unfolded form. As before, each cell in the figure can be a standard RNN, an LSTM, or a GRU unit cell and the overall network can be bidirectional as well. It should however be noted that, sequence models are inherently deep in time as their hidden state is a function of all previous hidden states. As such, while in the case of standard feed-forward neural networks, it is not uncommon to have very deep networks, having more than 2-3 layers stacked up in not very common in sequence models.

References

[1] Lipton, Z. C., Berkowitz, J., & Elkan, C. (2015). A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*.

[2] Hammer, B. (2000). On the approximation capability of recurrent neural networks. *Neurocomputing*, *31*(1-4), 107-123.

[3] Graves, A. (2012). Supervised sequence labelling. In *Supervised sequence labelling with recurrent neural networks* (pp. 5-13). Springer, Berlin, Heidelberg.

[4] Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, *79*(8), 2554-2558.

[5] Jordan, M. I. (1986). *Serial order: a parallel distributed processing approach. Technical report, June 1985-March 1986* (No. AD-A-173989/5/XAB; ICS-8604). California Univ., San Diego, La Jolla (USA). Inst. for Cognitive Science.

[6] Elman, J. L. (1990). Finding structure in time. *Cognitive science*, *14*(2), 179-211.

[7] Karpathy, A. (2015). The unreasonable effectiveness of recurrent neural networks. *Andrej Karpathy blog*, *21*.

[8] Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, *78*(10), 1550-1560.

[9] Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, *5*(2), 157-166.

[10] Hochreiter, S., Bengio, Y., Frasconi, P., & Schmidhuber, J. (2001). Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.

[11] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press.

[12] Pascanu, R., Mikolov, T., & Bengio, Y. (2013, February). On the difficulty of training recurrent neural networks. In *International conference on machine learning* (pp. 1310-1318).

[13] Williams, R. J., & Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, *1*(2), 270-280.

[14] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735-1780.

[15] Gers, F. A., Schmidhuber, J., & Cummins, F. (1999). Learning to forget: Continual prediction with LSTM.

[16] Gers, F. A., & Schmidhuber, J. (2000, July). Recurrent nets that time and count. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium* (Vol. 3, pp. 189-194). IEEE.

[17] Olah, C. (2015). Understanding lstm networks.

[18] Zargar, S. A., & Yuan, F. G. (2020). Impact diagnosis in stiffened structural panels using a deep learning approach. *Structural Health Monitoring*, 1475921720925044.

[19] ZARGAR, S. A., & YUAN, F. G. (2019). A deep learning approach for impact diagnosis. *Structural Health Monitoring 2019*.

[20] Yuan, F. G., Zargar, S. A., Chen, Q., & Wang, S. (2020, April). Machine learning for structural health monitoring: challenges and opportunities. In *Sensors and Smart Structures Technologies for Civil, Mechanical, and Aerospace Systems 2020* (Vol. 11379, p. 1137903). International Society for Optics and Photonics.

[21] Wang, S., Zargar, S. A., & Yuan, F. G. (2020). Augmented reality for enhanced visual inspection through knowledge-based deep learning. *Structural Health Monitoring*, 1475921720976986.

[22] WANG, S., ZARGAR, S. A., XU, C., & YUAN, F. G. (2019). An efficient augmented reality (AR) system for enhanced visual inspection. *Structural Health Monitoring 2019*.

[23] Khajwal, A. B., & Noshadravan, A. (2020). Probabilistic hurricane wind-induced loss model for risk assessment on a regional scale. *ASCE-ASME Journal of Risk and Uncertainty in Engineering Systems, Part A: Civil Engineering*, *6*(2), 04020020.

[24] Khajwal, A. B., & Noshadravan, A. (2021). An uncertainty-aware framework for reliable disaster damage assessment via crowdsourcing. *International Journal of Disaster Risk Reduction*, *55*, 102110.

[25] Lyathakula, K. R., & Yuan, F. G. (2021, March). Fatigue damage prognosis of adhesively bonded joints via a surrogate model. In *Sensors and Smart Structures Technologies for Civil, Mechanical, and Aerospace Systems 2021* (Vol. 11591, p. 115910K). International Society for Optics and Photonics.

[26] Lyathakula, K. R., & Yuan, F. G. (2021, March). Probabilistic fatigue life prediction for adhesively bonded joints via surrogate model. In *Sensors and Smart Structures Technologies for Civil, Mechanical, and Aerospace Systems 2021* (Vol. 11591, p. 115910S). International Society for Optics and Photonics.

[27] Cho, K., Van Merriënboer, B., Bahdanau, D., & Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*.

[28] Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

[29] Schuster, M., & Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, *45*(11), 2673-2681.

[30] Graves, A., Mohamed, A. R., & Hinton, G. (2013, May). Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing* (pp. 6645-6649). IEEE.