Bridge Weigh-In-Motion through Bidirectional Recurrent Neural Network with Long Short-Term Memory and Attention Mechanism

Zhichao Wang[1,a], Yang Wang[*1,2]

[1]*School of Civil and Environmental Engineering, Georgia Institute of Technology, Atlanta, GA, USA*
[2]*School of Electrical and Computing Engineering, Georgia Institute of Technology, Atlanta, GA, USA*

(Received July 7, 2020, Revised September 15, 2020, Accepted October 10, 2020)

Abstract In bridge weigh-in-motion (BWIM), dynamic bridge response is measured during traffic and used to identify overloaded vehicles. Most past studies of BWIM use mechanics-based algorithms to estimate axle weights. This research instead investigates deep learning, specifically the recurrent neural network (RNN), toward BWIM. In order to acquire the large data volume to train a RNN network that uses bridge response to estimate axle weights, a finite element bridge model is built through the commercial software package LS-DYNA. To mimic everyday traffic scenarios, tens of thousands of randomized vehicle formations are simulated, with different combinations of vehicle types, spacings, speeds, axle weights, axle distances, etc. Dynamic response from each of the randomized traffic scenarios is recorded for training the RNN. In this paper we propose a 3-stage Bidirectional RNN toward BWIM. Long short-term memory (LSTM) and attention mechanism are embedded in the BRNN to further improve the network performance. Additional test data indicates that the BRNN network achieves high accuracy in estimating axle weights, in comparison with a conventional moving force identification (MFI) method.

Keywords: Bridge weigh-in-motion, deep learning, bidirectional recurrent neural network, attention mechanism, long short-term memory

1. Introduction

While highway transportation is essential for flow of goods and people in a modern society, overloaded trucks have become a growing concern in bridge infrastructure maintenance (Fu and Hag-Elsafi, 2000). Highway pavement sustains the most direct consequences from overloaded trucks, while occasionally bridge structures can be endangered. Over the past four decades, many researchers have proposed different methods for detecting illegally overloaded trucks passing through a bridge from the bridge response measurements. Among those techniques is bridge weigh-in-motion (BWIM), which measures the dynamic response of a bridge structure and uses the response data to back derive the weight of vehicles driving over the bridge (Lydon et al., 2016; Yu et al., 2016). The installation of under-deck BWIM systems usually does not require traffic closure and is relatively low cost. Compared with pavement weigh-in-motion, another obvious benefit of BWIM is that the bridge response data can also be utilized to monitor the condition of the bridge structure itself (Skokandic et al., 2017).

Despite its advantages, the main challenge of BWIM lies in the robust numerical algorithm that can accurately estimate vehicle weight using dynamic bridge response data. Such data often contains strain or acceleration response of the bridge girders or deck (Zhu and Law, 2015; Zhu and Law, 2016). To this end, Moses' algorithm is acknowledged as the first widely adopted BWIM algorithm (Moses, 1979). The

[*] Corresponding author, Professor, E-mail: yang.wang@ce.gatech.edu
[a] Graduate student, E-mail: zcwang0201@gatech.edu

algorithm uses static influence lines to predict the response of a bridge girder and derives axle weights by minimizing the difference between predicted and measured girder responses. Despite its early popularity, the algorithm has some major limitations because the statically derived influence lines cannot consider bridge vibration dynamics. In addition, the Moses' algorithm omits the effect of transverse position of a vehicle on the bridge deck. Quilligan (2003) later proposed an influence area method to include the effect of transverse position, though the computational cost increased significantly, and the method still cannot consider the bridge vibration dynamics.

In order to achieve more accurate truck weight estimation using dynamic bridge response data, in the past few decades, most BWIM efforts have been made toward the consideration of bridge vibration dynamics. Relying on either an analytical mechanics model of a girder, or a finite element structural model of the bridge superstructure, many of these methods attempt to identify the time history of the contact force between a vehicle axle and the bridge deck; these methods are collectively referred as MFI (moving force identification) or MLI (moving load identification). Among earlier examples of these methods, an analytical model is needed to obtain bridge dynamic response. Then, an optimization problem with the contact force as an objective function variable is solved to minimize the difference between the analytical response and sensor measurement. Since the problems are often ill-posed, Tikhonov regularization (Tikhonov and Arsenin, 1977) is usually adopted for this group of methods. For example, Law et al. (1997) proposed a time domain method that showed acceptable accuracy in numerical simulation and experimental validation, though the method was sensitive to modeling inaccuracy and sensor noise. The time domain method was later extended into the frequency domain, while experimental study found the method computationally expensive and still sensitive to the measurement locations (Law et al., 1999). In addition, an interpretive method was put forward utilizing numerical simulation to "interpret" the moving load; the method is less computationally demanding while achieving lower accuracy (Chan et al., 1999). Recently, a method for moving force identification in state space with Hamilton's principle and modal superposition was formulated and validated using simulation and field test (Zhu et al., 2006). Though adequate BWIM accuracy was achieved with a large quantity of sensors deployed, the performance was found to deteriorate quickly with the reduction of sensors or with noisy data. Besides, all the experimental validations only investigated the scenario of a single two-axle vehicle on the bridge; no results were provided for scenarios with either a multi-axle vehicle or multiple vehicles passing through the bridge. These more complex yet realistic scenarios present critical challenges for the practical application of BWIM in the field.

Other examples in this group of MFI/MLI methods require a finite element model of the bridge superstructure. O'Connor and Chan (1988) put forward a method that modeled the bridge superstructure with lumped masses by connected by massless beam elements. The bridge response was then predicted, and the dynamic load could be inferred from measurement. Due to over simplification of the bridge model, this method can result in low accuracy in practical deployment. An optimal state estimation approach was then proposed to reduce the fluctuations of identified forces at the beginning and end of the time history, though the identification errors were still large in general (Law and Fang, 2001). Wu and Shi (2006) approximated the structural response and excitation by wavelets to reduce the computation cost, but the method required many sensor measurements, and the validation was only in simulation. In summary, MFI/MLI problems are ill-conditioned inverse problems that require careful regularization. When a more complex and accurate structural model is used, the time required for obtaining the moving force increases dramatically for each identification process.

The BWIM literature summarized above can be categorized into mechanics-based (or physics-based) BWIM algorithms. Meanwhile, in the broad fields of engineering and computer science, the past decade has seen unprecedented interest in deep learning, a topic in artificial intelligence that is data driven or based on data analytics. Using artificial neural networks (ANN), deep learning has particularly made significant strides in the past decade, transforming a number of engineering fields including image recognition and natural language processing (LeCun et al., 2015; Goodfellow et al., 2016). A classical deep learning neural network usually has perceptrons as building blocks (Minsky and Papert, 1969). A perceptron consists of an (usually nonlinear) activation function that maps the sum of a bias and the multiplication between weightings and inputs to an output. The sigmoid function, $\sigma(x) = e^x/(1 + e^x)$, is among the most widely

used activation functions. For example, $\hat{y} = \sigma\left(\sum_{i=1}^{3} w_i x_i + b\right)$ is a perceptron that takes three input scalars (x_1, x_2 and x_3) and generates one output $\hat{y}$; the perceptron has three weightings (w_1, w_2 and w_3) and one bias b shown in Fig. 1.

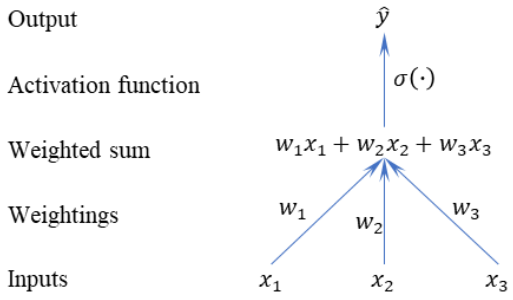


Fig. 1 A perceptron with three input and one output variable

For a neural network to accurately generate the correct/desired output from certain input data, the weightings and biases are the most important function parameters whose values need to be fine-tuned. The process of automatically fine-tuning these parameter values is called the training of a neural network, which is usually performed through supervised learning. Take image recognition for example – to construct a neural network that can correctly recognize images, supervised learning starts with a lot of training data, which are essentially many pictures (input to the network) with known labels (correct/desired output from the network). Consisting of many layers of neural network, with many function parameters to be trained, deep learning techniques automatically find appropriate parameter values through mathematical optimization. This training process essentially identifies/learns the oftentimes highly nonlinear relationship which maps input (an image) to output (a label). To detect overfitting and choose hyperparameters, performance of the trained network can be simultaneously verified using (cross) validation data, which are additional known input-output pairs that were not among the training data. When the training process ends, test data can be utilized to assess the performance of the trained neural network. If overfitting occurs, techniques such as dropout (Srivastava et al., 2014) and regularization (Goodfellow, et al., 2016) can be applied to reduce overfitting and improve performance with the validation and test sets.

Early research in multi-layer neural network training started in the 1950's (Rosenblatt, 1957; Selfridge, 1959). However, it was the past decade that has seen the most significant improvement in deep learning. These are mainly attributed to two reasons. The first reason is the recent development of innovative and sophisticated neural network architectures that can adapt to complex problems. The second reason is the increasing availability of data and the leap in computing power, which enables the training of much larger neural networks. Besides, the widespread availability of GPUs (graphics processing units) provides another significant boost to computing power by efficiently distributing the computation load toward deep learning (Krizhevsky et al., 2012; Das and Deka, 2016). In a typical deep learning application today, there can be tens of millions function parameters to be learned/trained in a sophisticated network, which would have been impossible only a decade ago. For example, convolutional neural networks (ConvNets or CNN) made the biggest breakthrough in image recognition, where the input data comes in the form of multiple arrays. In particular, a color image usually consists of three 2D arrays with picture intensities of three color channels (R/G/B). In a CNN, convolutional layers, pooling layers and fully-connected layers are frequently utilized. The convolutional layer plays the role of a feature detector/filter that slides through the pixel blocks in two dimensions, a process mathematically similar to discrete convolution (LeCun et al., 1998). Pooling layers then reduce the dimension of the intermediate feature results without introducing any new parameters to be learned (Yang et al., 2009). Finally, the fully-connected layers produce estimated outputs using the information extracted from the earlier convolutional layers or pooling layers (Krizhevsky, et al., 2012).

Toward BWIM applications, two-layered neural networks were first proposed to output gross vehicle weight and axle weight distribution factors, which were then utilized to calculate each axle weight (Kim et

al., 2009). The network had a primitive architecture that is fully connected. In addition, due to the limitation in computing power from a decade ago, the network only accepted peak strain data from a small number of strain sensors, instead of dynamic time histories from heterogenous sensors. Input data size to the network is around ten (whereas today, input data size of tens of thousands are commonplace). As a result, the network had mediocre performance; some test cases showed more than ±20% errors, which are far from acceptable for practical applications. Furthermore, the network could only handle the situation when only one vehicle travels over the bridge, which prevents its practical adoption. In another BWIM study, a convolutional neural network was constructed to estimate vehicle existence and speed using strain data as input (Kawakatsu et al., 2019). However, the vehicle weight estimation is still through traditional Moses' algorithm, which is subject to the same inaccuracies of early conventional mechanics-based BWIM algorithms summarized earlier. Zhang et al. (2010) proposed a neural network approach for pavement weigh-in-motion. The study used measurement from sensors embedded in road pavement, instead of response from a bridge structure.

Besides CNN, another category of deep learning has made fascinating progress in natural language processing. In natural language processing, a number of innovative neural network architectures have been studied. The recurrent neural network (RNN) was first proposed to deal with time series data, a ubiquitous feature of natural languages (Rumelhart et al., 1986). RNN was further advanced to BRNN (bidirectional recurrent neural network) by including both forward transformation and backward transformation (Schuster and Paliwal, 1997). Later, the benefits brought by the increased depth of neural network were discussed (Graves et al., 2013). Two other methods had also been proposed, which greatly improved the performance of natural language processing. The first one made the process of training more robust, especially for a long input sequence, using LSTM (long short-term memory) (Hochreiter and Schmidhuber, 1997) or GRU (gated recurrent unit) (Chung et al., 2014). GRU can be viewed as a simplified version of LSTM for dealing with long input sequences, while LSTM's robustness has been more widely verified. Both GRU and LSTM techniques create one more intermediate variable than RNN, which changes slowly through the sequence and achieves the goal of memorizing input information a long time ago. The other method is the attention mechanism which could deal with different lengths of inputs and outputs, and improve the performance when the input consists of long sequences (Chorowski et al., 2014). The attention mechanism is embedded in an encoder-decoder architecture, where one RNN encodes the raw data and another RNN decodes for final output, and attention mechanism connects the encoder RNN and decoder RNN through appropriate weightings.

To our best knowledge, the authors have not seen the latest deep learning techniques, such as BRNN and LSTM being adopted toward BWIM applications. For practical application, significant challenge also exists in obtaining the large amount (e.g. hundreds of thousands) of training data sets from field measurements. Such training data should contain not only bridge response, but also the true weights of hundreds of thousands of various passing vehicles. The latter can be hardly feasible to obtain in practice. Nevertheless, it is envisioned that for an actual bridge application, finite element model updating of the bridge can be performed first to achieve high fidelity under different environmental scenarios. The updating usually has moderate demand in field measurement data, and the calibrated model(s) can then be used to generate the hundreds of thousands of training data sets. As a preliminary study in the topic area, we use training data sets generated from high-fidelity simulations. To simulate dynamic bridge response with a moving vehicle load, a commercial software package LS-DYNA is used to build the bridge finite element model. For the relatively short bridge span in consideration, three different scenarios are considered, i.e. one truck, one truck and one car, and two trucks. In each example, different axle numbers, axle weights, axle distances, vehicle distances and vehicle speeds are combined. The neural network takes strain histories as input, and outputs the estimation for axle weights. A sophisticated network architecture is proposed in order to exploit both BRNN and RNN to extract useful information concealed in the strain measurements. LSTM is adopted to enhance the capability of the neural network in dealing with long input sequences, i.e. multiple strain measurements at a high sampling rate. As the final stage of the proposed network architecture, the attention mechanism is adopted to finally provide axle weights. In comparison with MFI/MLI, the proposed method does not suffer from ill conditioning. Besides, the neural network can deal

with high noise levels so long as it is trained with data that has high noise levels. Lastly, after network training, the actual process of deriving axle weights is instantaneous and appropriate for practical applications.

The remainder of the paper is organized as follows. In Section 2, classical RNN neural networks and the architecture of a neural network model for BWIM are reviewed. The section introduces the concepts of bidirectional RNN, LSTM, and attention mechanism. Section 3 discusses the randomized numerical simulation of the bridge dynamic response, including the testbed bridge information, measurement scheme, finite element modeling, and combination of different scenarios. The BWIM accuracy and comparison with previous MFI/MLI methods are finally presented in Section 4.

2. Bidirectional recurrent neural network

This section describes the bidirectional recurrent neural network (BRNN) developed for BWIM. Classical recurrent neural network (RNN) is first summarized. Then, some useful techniques for improving the network performance, like BRNN are introduced. Lastly, the architecture of the BRNN model for BWIM is described.

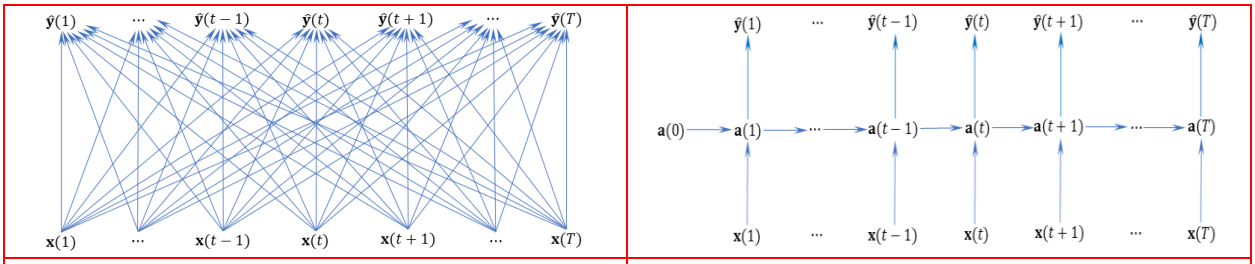
2.1. Recurrent neural network (RNN)

2.1.1 Traditional recurrent neural network with the same length in input and output sequences

Both the input and the output of a neural network can be viewed as a sequence of vectors. The length of input sequence (i.e. the number of input vectors) is denoted as T_x, while the length of output sequence is denoted as T_y. Let's first consider the scenario where the length of the input sequence equals the length of the output sequence, i.e. $T_x = T_y = T$. At each sequence step, the dimension of the input vector is denoted as n_x, while the dimension of each output vector is denoted n_y. We do not require n_x and n_y to be equal. The input to a neural network at sequence step t is denoted as $\mathbf{x}(t) \in \mathbb{R}^{n_x}$, $t = 1, 2, \dots, T$. The network output at step t is $\hat{\mathbf{y}}(t) \in \mathbb{R}^{n_y}$, where the hat symbol carries the meaning that (after training) the network attempts to estimate the output $\mathbf{y}$ using the input $\mathbf{x}$.

Fig. 2(a) shows a conventional fully-connected neural network, where the computation of each output estimation $\hat{\mathbf{y}}(t)$ requires input vectors at all steps. When estimating $\hat{\mathbf{y}}(t)$ for a current step t, not only input from past time steps $1,2, \dots, t-1$, but also input from future steps $t+1, t+2, \dots, T$ are utilized. For an application with a large number of input/output vectors at high dimensional space, this conventional architecture requires a significant amount of computational capability that is usually unaffordable even with state-of-the-art of hardware. The network training quickly becomes highly inefficient, if not impossible.

In comparison, Fig. 2(b) shows the architecture of a recurrent neural network (RNN) that is particularly efficient in dealing with time history data (of a long sequence). Instead of using all $\mathbf{x}(t)$, $t = 1, 2, \dots, T$ to decide the output at one time step, intermediate vector variables $\mathbf{a}(t)$ are introduced. The intermediate variable recurrently accumulates information from past input data and uses the information toward estimating future output.



(a) Fully-connected neural network (b) Recurrent neural network (RNN)

Fig. 2 Flow diagram and architecture of neural networks

In the RNN demonstrated in Fig. 2(b), $\mathbf{a}(t) \in \mathbb{R}^{n_a}$ is an intermediate (hidden) vector variable at step t. To initiate the RNN process, $\mathbf{a}(0)$ is usually set as a zero vector $\mathbf{0}_{n_a}$. The value of the intermediate (hidden) variable at step t, $\mathbf{a}(t)$, is calculated from the intermediate (hidden) variable at previous step $t-1$, $\mathbf{a}(t-1) \in \mathbb{R}^{n_a}$, and the input at step t, $\mathbf{x}(t) \in \mathbb{R}^{n_x}$. The mapping from $\mathbf{a}(t-1)$ and $\mathbf{x}(t)$ to $\mathbf{a}(t)$ is realized through a nonlinear activation function. In this study, an entry-wise hyperbolic tangent function $\tanh(x) = (e^x - e^{-x})/(e^x + e^{-x})$ is adopted as the activation function.

$$\mathbf{a}(t) = \tanh(\mathbf{W}_{aa}\mathbf{a}(t-1) + \mathbf{W}_{ax}\mathbf{x}(t) + \mathbf{b}_a) \tag{1}$$

Here $\mathbf{W}_{aa} \in \mathbb{R}^{n_a \times n_a}$ and $\mathbf{W}_{ax} \in \mathbb{R}^{n_a \times n_x}$ are the weighting matrices, and $\mathbf{b}_a \in \mathbb{R}^{n_a}$ is the bias vector. Consequently, $\mathbf{a}(t)$ is used to generate output $\hat{\mathbf{y}}(t) \in \mathbb{R}^{n_y}$, the estimation of $\mathbf{y}(t) \in \mathbb{R}^{n_y}$. The mapping from $\mathbf{a}(t)$ to $\hat{\mathbf{y}}(t)$ uses a weighting matrix $\mathbf{W}_{ya} \in \mathbb{R}^{n_y \times n_a}$ and a bias vector $\mathbf{b}_y \in \mathbb{R}^{n_y}$.

$$\hat{\mathbf{y}}(t) = \tanh\left(\mathbf{W}_{ya}\mathbf{a}(t) + \mathbf{b}_y\right) \tag{2}$$

In neural network jargon, label refers to the known output $\mathbf{y}(t)$ that corresponds to some known input $\mathbf{x}(t)$. The labels, together with the corresponding inputs, are used to train the network by finding the optimal values of the constant variables, including the weighting matrices and bias vectors. With $\mathbf{W}_{aa}$, $\mathbf{W}_{ax}$, $\mathbf{b}_a$, $\mathbf{W}_{ya}$, and $\mathbf{b}_y$ as the optimization variables, the value of a non-negative cost function $J \in \mathbb{R}^+$ can be defined based on the difference between the known labels and estimated output. The training of a neural network is essentially a mathematical optimization process to find the best weighting matrixes and bias vectors that can minimize the cost function J. Gradient descent is frequently utilized to solve this often very large optimization problem.

$$J = \sum_{i=1}^{m} \sum_{j=1}^{T_y} \left\| \mathbf{y}^i(j) - \hat{\mathbf{y}}^i(j) \right\|_2^2 \tag{3}$$

where $m \in \mathbb{R}$ represents the total number of training data sets, T_y is the length of output sequence, $\mathbf{y}^i(j) \in \mathbb{R}^{n_y}$ is the label for the j-th step output of the i-th data set, $\hat{\mathbf{y}}^i(j) \in \mathbb{R}^{n_y}$ is the j-th step output of the i-th training input data set, and $\|\cdot\|_2$ represents the vector $\mathcal{L}2$-norm.

$$\mathbf{W}(k+1) = \mathbf{W}(k) - \lambda(k) \frac{\partial J}{\partial \mathbf{W}} \left(\mathbf{W}(k), \mathbf{b}(k)\right) \tag{4}$$

$$\mathbf{b}(k+1) = \mathbf{b}(k) - \lambda(k) \frac{\partial J}{\partial \mathbf{b}} \left(\mathbf{W}(k), \mathbf{b}(k)\right) \tag{5}$$

where k represents the k-th iteration step in the optimization process, $\mathbf{W}$ represents all weighting matrices, $\mathbf{b}$ represents all bias vectors and $\lambda(k) \in \mathbb{R}^+$ is the learning rate at the k-th step. The learning rate is one of the most important parameters to be tuned in the network training, and it can be set as a constant in most applications.

One mistake to avoid in network training is overfitting, which means the trained neural network provides good performance with the training set, while the performance deteriorates with the validation set or test set. To prevent overfitting, we use a regularization technique in this study (Goodfellow, et al., 2016). The technique adds a new penalty term to Eq. (3), so that the weightings do not grow too large. Here, μ is a parameter to be tuned, and $\|.\|_F$ represents the matrix Frobenius norm.

$$J = \sum_{i=1}^{m} \sum_{j=1}^{T_y} \left\| \mathbf{y}^i(j) - \hat{\mathbf{y}}^i(j) \right\|_2^2 + \mu \|\mathbf{W}\|_F^2 \tag{6}$$

2.1.2. LSTM (long short-term memory) Unit

When network size is large, the performance of RNN shown in Fig. 2(b) deteriorates with the increase of input data volume. Besides, in the process of training a RNN with a long input sequence, vanishing gradient and gradient explosion are among the difficulties commonly encountered. Vanishing gradient means that the gradient $\partial J/\partial \mathbf{W}$ or $\partial J/\partial \mathbf{b}$ (in Eqs. (4) and (5)) approaches zero, resulting in very slow convergence. Gradient explosion means that the gradient becomes very large, leading to failure in optimization. In practice, gradient explosion is usually dealt with by setting a maximum threshold limit for the gradient. A vanishing gradient is usually more difficult to deal with. To this end, methods like LSTM (Hochreiter and Schmidhuber, 1997) and GRU (Chung, et al., 2014) have been proposed. In this study, LSTM is adopted due to its widely accepted stability.

Recall from Eq. (1) that in a normal RNN unit, the intermediate (hidden) variable at step t, $\mathbf{a}(t)$, is calculated using the intermediate (hidden) variable at step $t-1$, $\mathbf{a}(t-1)$, and the input at step t, $\mathbf{x}(t)$. After many steps of propagation, $\mathbf{a}(t-1)$ alone usually cannot contain input information much earlier than current step t. To maintain a longer term memory, Fig. 3 demonstrates an alternative approach for calculating the intermediate variable $\mathbf{a}(t)$ that provides long short-term memory (LSTM).

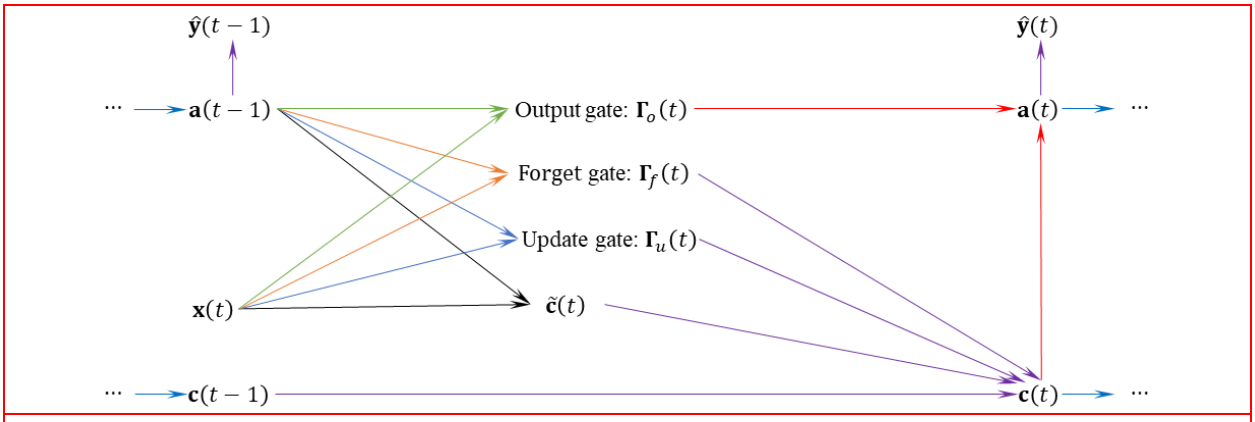


Fig. 3 Flow diagram providing LSTM (long short-term memory)

Similar as a normal RNN unit, a candidate memory cell $\tilde{\mathbf{c}}(t) \in \mathbb{R}^{n_a}$ is first calculated at step t:

$$\tilde{\mathbf{c}}(t) = \tanh(\mathbf{W}_{ca}\mathbf{a}(t-1) + \mathbf{W}_{cx}\mathbf{x}(t) + \mathbf{b}_c) \tag{7}$$

where $tanh(\cdot)$ is the elementwise hyperbolic tangent function, $\mathbf{W}_{ca} \in \mathbb{R}^{n_a \times n_a}$ and $\mathbf{W}_{cx} \in \mathbb{R}^{n_a \times n_x}$ are the weighting matrices, and $\mathbf{b}_c \in \mathbb{R}^{n_a}$ is the bias vector. In addition, also using $\mathbf{a}(t-1)$ and $\mathbf{x}(t)$, three new intermediate variables are introduced: $\mathbf{\Gamma}_u(t) \in \mathbb{R}^{n_a}$ as the update gate, $\mathbf{\Gamma}_f(t) \in \mathbb{R}^{n_a}$ as the forget gate, and $\mathbf{\Gamma}_o(t) \in \mathbb{R}^{n_a}$ as the output gate. Entries in these gate variables are restricted to be in the interval [0,1], in order to control the percentage of information to forget or to remember. Therefore, an entry-wise version of the sigmoid function, $\sigma(x) = e^x/(1+e^x)$, is used to calculate all these gate vectors.

$$\boldsymbol{\Gamma}_u(t) = \sigma(\mathbf{W}_{ua}\mathbf{a}(t-1) + \mathbf{W}_{ux}\mathbf{x}(t) + \mathbf{b}_u) \tag{8}$$

$$\boldsymbol{\Gamma}_f(t) = \sigma\big(\mathbf{W}_{fa}\mathbf{a}(t-1) + \mathbf{W}_{fx}\mathbf{x}(t) + \mathbf{b}_f\big) \tag{9}$$

$$\boldsymbol{\Gamma}_o(t) = \sigma(\mathbf{W}_{oa}\mathbf{a}(t-1) + \mathbf{W}_{ox}\mathbf{x}(t) + \mathbf{b}_o) \tag{10}$$

Here the weighting matrices are $\mathbf{W}_{ua} \in \mathbb{R}^{n_a \times n_a}$, $\mathbf{W}_{ux} \in \mathbb{R}^{n_a \times n_x}$, $\mathbf{W}_{fa} \in \mathbb{R}^{n_a \times n_a}$, $\mathbf{W}_{fx} \in \mathbb{R}^{n_a \times n_x}$, $\mathbf{W}_{oa} \in \mathbb{R}^{n_a \times n_a}$, and $\mathbf{W}_{ox} \in \mathbb{R}^{n_a \times n_x}$; the bias vectors include $\mathbf{b}_u \in \mathbb{R}^{n_a}$, $\mathbf{b}_f \in \mathbb{R}^{n_a}$, and $\mathbf{b}_o \in \mathbb{R}^{n_a}$.

The memory cell at step t, $\mathbf{c}(t) \in \mathbb{R}^{n_a}$, is calculated as a weighted sum between the candidate memory cell $\tilde{\mathbf{c}}(t)$ and the memory cell at previous step $t-1$, $\mathbf{c}(t-1)$. The two weightings are the update gate $\boldsymbol{\Gamma}_u(t)$ and forget gate $\boldsymbol{\Gamma}_f(t)$, respectively. As a result, the memory cell achieves a balance between the input information from the most recent step and from past steps.

$$\mathbf{c}(t) = \boldsymbol{\Gamma}_u(t) * \tilde{\mathbf{c}}(t) + \boldsymbol{\Gamma}_f(t) * \mathbf{c}(t-1) \tag{11}$$

Here, the star symbol "$*$" means elementwise multiplication between two vectors of the same shape. In general, the update of $\mathbf{c}(t)$ is slow in order to memorize early information.

Finally, as shown in Fig. 3, the intermediate variable $\mathbf{a}(t) \in \mathbb{R}^{n_a}$ is calculated using output gate $\boldsymbol{\Gamma}_o(t)$ as the weighting, and memory cell $\mathbf{c}(t)$ is activated through the elementwise hyperbolic tangent function.

$$\mathbf{a}(t) = \boldsymbol{\Gamma}_o(t) * \tanh\big(\mathbf{c}(t)\big) \tag{12}$$

The mapping from $\mathbf{a}(t)$ to $\hat{\mathbf{y}}(t)$ uses a weighting matrix $\mathbf{W}_{ya} \in \mathbb{R}^{n_y \times n_a}$ and a bias vector $\mathbf{b}_y \in \mathbb{R}^{n_y}$.

$$\hat{\mathbf{y}}(t) = \tanh\big(\mathbf{W}_{ya}\mathbf{a}(t) + \mathbf{b}_y\big) \tag{13}$$

In summary, LSTM utilizes $\mathbf{a}(t-1)$, $\mathbf{c}(t-1)$ and input $\mathbf{x}(t)$ to calculate $\mathbf{a}(t)$, $\mathbf{c}(t)$ and output $\hat{\mathbf{y}}(t)$. Memory cell $\mathbf{c}(t)$ generally changes slower in comparison with $\mathbf{a}(t)$. In this way, $\mathbf{a}(t)$ can capture the rapid input changes, while $\mathbf{c}(t)$ can help retain more information from past time steps. Lastly, similar to the general process described in Eqs. (3)~(6), all the weighting matrices and bias vectors in LSTM are to be determined through network training. The training finds optimal matrices and vectors such that an estimation error index, i.e. cost function *J*, is minimized.

2.2. Bidirectional recurrent neural network (BRNN) with the same length of inputs and outputs

One drawback of RNN is that when estimating $\hat{\mathbf{y}}(t)$, the network only uses input data from previous steps (before t) and not any future steps (after t). Bidirectional recurrent neural network (BRNN) is a recently developed network architecture which uses input data both before and after t. Again, for the scenario where the length of input sequence equals the length of output sequence ($T_x = T_y = T$), Fig. 4 shows the flow diagram of a BRNN.

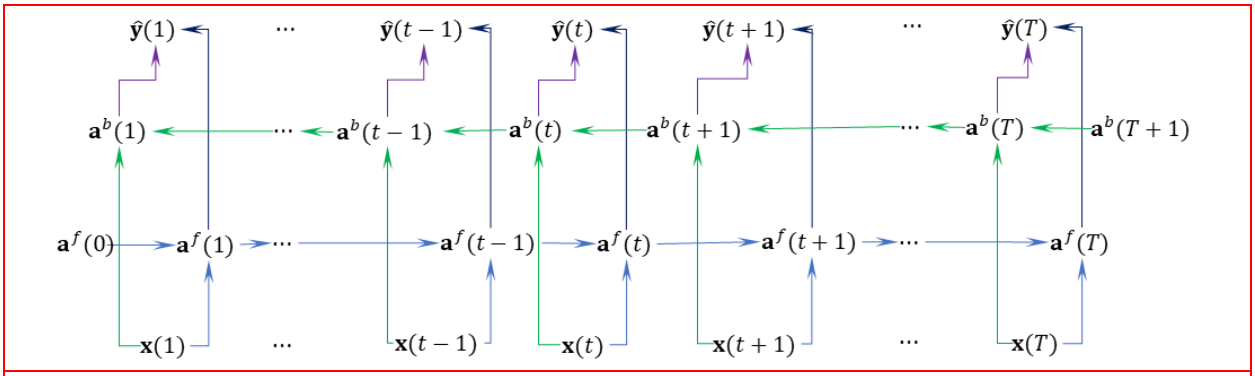


Fig. 4 Flow diagram of a bidirectional recurrent neural network (BRNN)

Recall that at step t, RNN uses an intermediate (hidden) variable $\mathbf{a}(t)$. In comparison, BRNN introduces a forward intermediate variable $\mathbf{a}^f(t) \in \mathbb{R}^{n_a}$ and a backward intermediate variable $\mathbf{a}^b(t) \in \mathbb{R}^{n_a}$. The forward variable $\mathbf{a}^f(t)$ depends on the input $\mathbf{x}(t)$ and the forward variable at previous step, $\mathbf{a}^f(t-1)$. This forward layer utilizes a hyperbolic tangent activation function with two weighting matrices $\mathbf{W}_{aa}^f \in \mathbb{R}^{n_a \times n_a}$ and $\mathbf{W}_{ax}^f \in \mathbb{R}^{n_a \times n_x}$, as well as a bias vector $\mathbf{b}_a^f \in \mathbb{R}^{n_a}$.

$$\mathbf{a}^f(t) = \tanh\left(\mathbf{W}_{ax}^f \mathbf{x}(t) + \mathbf{W}_{aa}^f \mathbf{a}^f(t-1) + \mathbf{b}_a^f\right) \tag{14}$$

On the other hand, the backward intermediate (hidden) variable $\mathbf{a}^b(t)$ depends on the input $\mathbf{x}(t)$ and the backward intermediate (hidden) variable at the next step, $\mathbf{a}^b(t+1)$. This backward activation allows the use of input from future steps through a hyperbolic tangent activation function with weighting matrices $\mathbf{W}_{aa}^b \in \mathbb{R}^{n_a \times n_a}$ and $\mathbf{W}_{ax}^b \in \mathbb{R}^{n_a \times n_x}$, as well as a bias vector $\mathbf{b}_a^b \in \mathbb{R}^{n_a}$.

$$\mathbf{a}^b(t) = \tanh\left(\mathbf{W}_{ax}^b \mathbf{x}(t) + \mathbf{W}_{aa}^b \mathbf{a}^b(t+1) + \mathbf{b}_a^b\right) \tag{15}$$

As shown in Fig. 4, both $\mathbf{a}^f(0)$ and $\mathbf{a}^b(T+1)$ are needed to initialize the bidirectional recurrent neural network. They are set as zero vectors, i.e. $\mathbf{a}^f(0) = \mathbf{a}^b(T+1) = \mathbf{0}_{n_a}$. The last step in the BRNN is to combine the information of $\mathbf{a}^f(t)$ and $\mathbf{a}^b(t)$ to estimate $\hat{\mathbf{y}}(t) \in \mathbb{R}^{n_y}$. The estimation is accomplished through the hyperbolic tangent function with weighting matrices $\mathbf{W}_{ya}^f \in \mathbb{R}^{n_y \times n_a}$ and $\mathbf{W}_{ya}^b \in \mathbb{R}^{n_y \times n_a}$, and bias vector $\mathbf{b}_y \in \mathbb{R}^{n_y}$.

$$\hat{\mathbf{y}}(t) = \tanh\left(\mathbf{W}_{ya}^f \mathbf{a}^f(t) + \mathbf{W}_{ya}^b \mathbf{a}^b(t) + \mathbf{b}_y\right) \tag{16}$$

For clarity, the BRNN illustration in Fig. 4 does not include LSTM shown in Fig. 3, while the combination of BRNN and LSTM modules is frequently utilized. To this end, at the t-th step in the forward RNN, the inputs to the LSTM module include forward variable $\mathbf{a}^f(t-1)$, $\mathbf{x}(t)$ and forward memory cell $\mathbf{c}^f(t-1)$. Forward candidate memory cell $\tilde{\mathbf{c}}^f(t)$ is calculated using Eq. (7), and different gates in the forward RNN, $\mathbf{\Gamma}_o^f(t)$, $\mathbf{\Gamma}_u^f(t)$ and $\mathbf{\Gamma}_f^f(t)$, are calculated using Eqs. (8), (9) and (10). Finally, $\mathbf{c}^f(t)$ and $\mathbf{a}^f(t)$ are derived using Eq. (11) and (12), respectively. Similarly, the backward RNN integrates LSTM, using corresponding backward variables $\mathbf{a}^b(t+1)$, $\mathbf{c}^b(t+1)$, $\tilde{\mathbf{c}}^b(t)$, $\mathbf{\Gamma}_o^b(t)$, $\mathbf{\Gamma}_u^b(t)$, $\mathbf{\Gamma}_f^b(t)$, $\mathbf{c}^b(t)$ and $\mathbf{a}^b(t)$.

2.3. Attention mechanism to deal with long input sequence and with different lengths of inputs and outputs

Both RNN and BRNN described above require the input and output lengths to be the same: $T_x = T_y = T$. However, in many applications, the lengths of input sequence and output sequence are different, i.e.

$T_x \neq T_y$. Taking BWIM (bridge weigh-in-motion) for example, the length of input sequence depends on data collection time and sampling frequency, while the length of output sequence is simply the number of truck axle weights to be estimated. To this end, we can use an encoder-decoder architecture that contains an encoder RNN to extract information from input and a decoder RNN to generate the output (Sutskever et al., 2014; Sutskever et al., 2014). However, the performance of such an architecture deteriorates as the length of input sequence increases, because it is challenging for the neural network to memorize all the inputs when the length of input sequence is large. In order to improve the performance when dealing with long input sequence with a large T_x, the attention mechanism was recently proposed (Chorowski, et al., 2014).

When $T_x \neq T_y$, suppose t_y represents the sequence step in the output ($t_y = 1, 2, \cdots, T_y$), and t_x represents the sequence step in the input ($t_x = 1, 2, \cdots, T_x$). Fig. 5 shows the proposed neural network that incorporates attention mechanism and can be used for BWIM. The network contains three main stages: (1) pre-attention BRNN with LSTM, (2) attention mechanism and (3) post-attention RNN with LSTM. For clarity, LSTM is not illustrated in the figure.

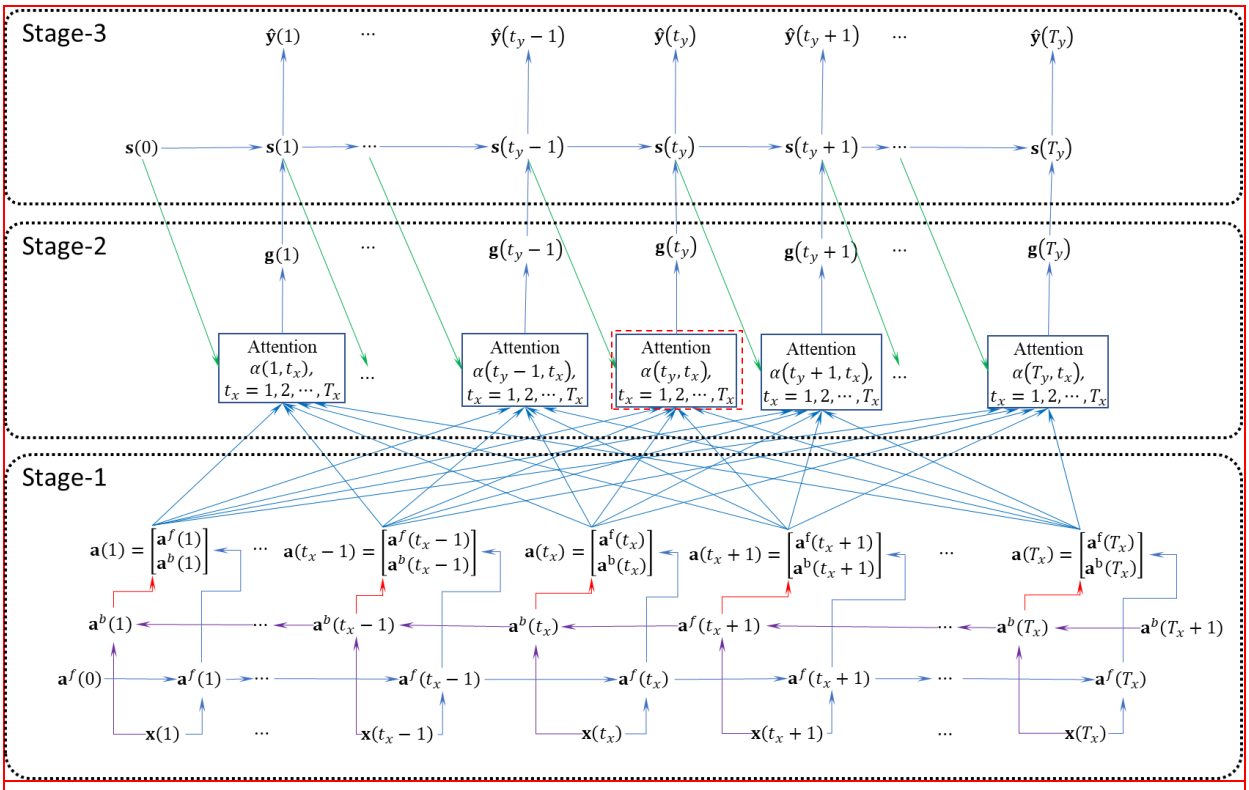


Fig. 5 Proposed 3-stage neural network architecture for BWIM

In detail, the pre-attention BRNN outputs intermediate variable $\mathbf{a}(t_x) \in \mathbb{R}^{2n_a}$. Together with intermediate variable $\mathbf{s}(t_y - 1)$ from the post-attention RNN, $\mathbf{a}(t_x)$ can be used to decide the attention factors $\alpha(t_y, t_x) \in (0,1)$ through a fully-connected network. With a value between 0 and 1, the attention factor determines how much each pre-attention BRNN output $\mathbf{a}(t_x)$ contributes into context variable $\mathbf{g}(t_y) \in \mathbb{R}^{2n_a}$, which is among the inputs to the post-attention RNN finally generating $\hat{\mathbf{y}}(t_y)$.

$$\mathbf{g}(t_y) = \sum_{t_x=1}^{T_x} \alpha(t_y, t_x) \cdot \mathbf{a}(t_x) \tag{17}$$

The attention factor $\alpha(t_y, t_x)$ is calculated from intermediate variable $\mathbf{a}(t_x)$ from the pre-attention BRNN, and intermediate variable $\mathbf{s}(t_y - 1)$ fed backward from the post-attention RNN. The inclusion of $\mathbf{s}(t_y - 1)$ allows the decision for later output to consider previous information. The details of the attention mechanism are omitted in Fig. 5 for clarity, and provided in Fig. 6 instead. In the attention mechanism, intermediate variables $\mathbf{a}(t_x)$ and $\mathbf{s}(t_y - 1)$ are fed into a sigmoid function to first calculate energy score $e(t_y, t_x) \in \mathbb{R}$ with weightings $\mathbf{w}_{es} \in \mathbb{R}^{n_s}$ and $\mathbf{w}_{ea} \in \mathbb{R}^{2n_a}$ and $b_e \in \mathbb{R}$.

$$e(t_y, t_x) = \sigma\left(\mathbf{w}_{es}^{\mathrm{T}}\mathbf{s}(t_y - 1) + \mathbf{w}_{ea}^{\mathrm{T}} \cdot \mathbf{a}(t_x) + b_e\right), t_x = 1, 2, \cdots, T_x \tag{18}$$

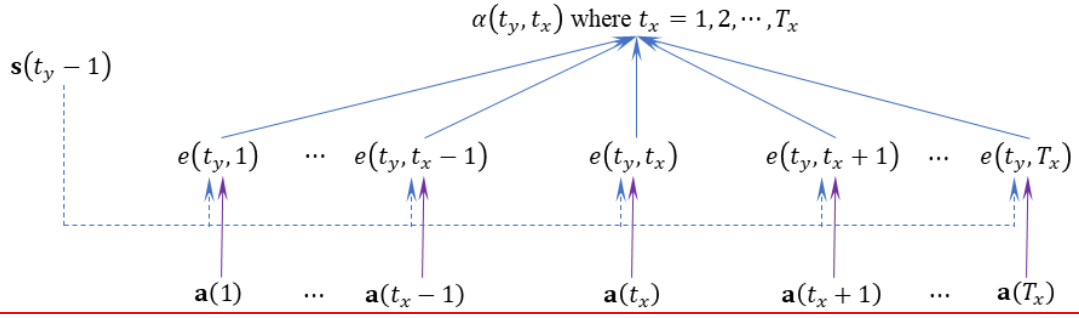


Fig. 6 Attention mechanism

From the energy scores $e(t_y, t_x)$, a softmax classification finds the attention factors $\alpha(t_y, t_x)$, which is now the normalized weighting of $\mathbf{a}(t_x)$ contributing into context variable $\mathbf{g}(t_y)$ (see Eq. (17)). Note the sum of $\alpha(t_y, t_x)$ over $t_x = 1, 2, \cdots, T_x$ equals one, i.e. $\sum_{t_x=1}^{T_x} \alpha(t_y, t_x) = 1$.

$$\alpha(t_y, t_x) = \frac{\exp\left(e(t_y, t_x)\right)}{\sum_{t_x=1}^{T_x} \exp\left(e(t_y, t_x)\right)} \tag{19}$$

Thus far, the complete architecture of our proposed BWIM neural network has been introduced. By the three stages in Fig. 5, we can count the total number of parameters for an example network with $T_x = 100$, $n_x = 60$, $n_a = 1{,}000$, $n_s = 2{,}000$ and $n_y = 1$. Here, n_a is the dimension of both $\mathbf{a}^f(t_x)$ and $\mathbf{a}^b(t_x)$ in Stage-1. In total, the number of parameters from all three stages in this example BWIM network is 40,502,002. This amount of parameters, over 40 million, could be impossible to efficiently train from only a decade ago, but fairly common in today's AI applications (Zhang et al., 2019).

3. Numerical Simulation

Section 3.1 introduces the highway bridge based on which this study is developed, as well as the corresponding finite element model. Section 3.2 describes the randomized vehicle parameters for bridge response simulation which will be used toward the training, validation, and testing data sets for the neural network.

3.1. Bridge modeling

Fig. 7 shows the testbed bridge on which this study is performed. The two-lane bridge is located in LaGrange, GA. The skewed bridge consists of four spans and supports two lanes of traffic in one direction. This work focuses on the simply supported span #1 where traffic enters; the simply supported span is expected to have the least interaction with the other spans (Fig. 8). Also, because the span is fairly short, less interaction in bridge dynamics is expected between preceding vehicles and later vehicles. The concrete

bridge deck is supported by six I-shaped steel girders, denoted as G1 ~ G6. The simply supported girders are spaced 7 feet and 10 inches away from one another, connected by lateral diaphragms. Cross section view of the bridge deck is shown in Fig. 9.

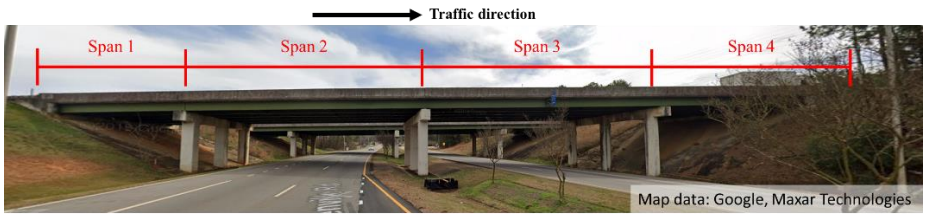


Fig. 7 Photo of the steel-girder bridge under study

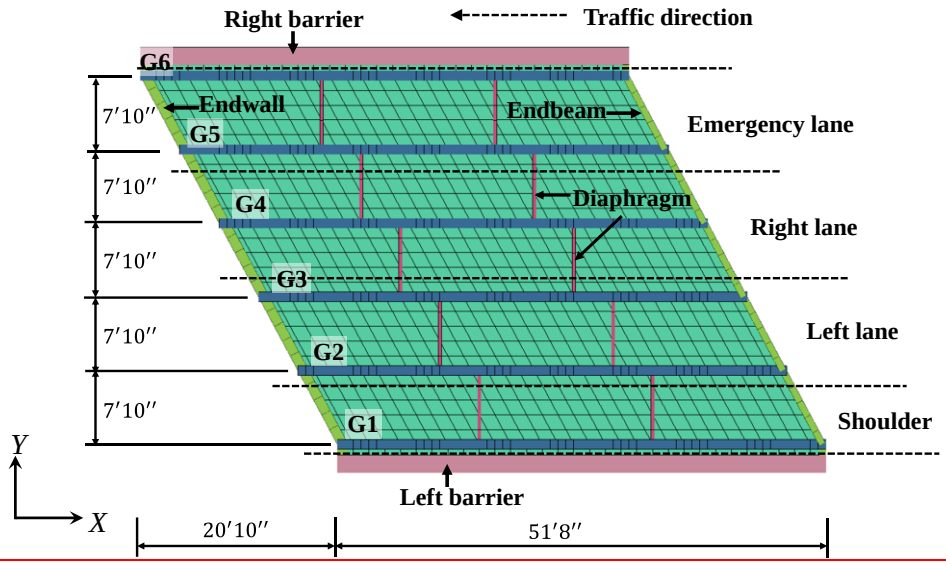


Fig. 8 Plan view of the LS-DYNA model for span #1

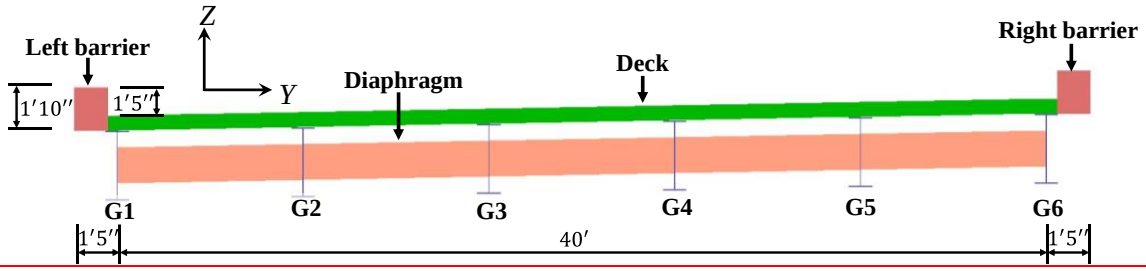


Fig. 9 Cross section view of the bridge deck (looking along the traffic direction)

Using a commercial software package LS-DYNA, a finite element model of bridge span #1 is constructed. In the modeling process, barriers, girders, endbeam, endwall and lateral diaphragms are all modeled with beam elements, while the concrete deck is modeled with shell elements. As shown in Fig. 8, a pin support is provided at the endwall, and a roller at the endbeam. Except for the six steel girders and diaphragms, all other bridge components are made of concrete. The nominal material properties are listed in Table 1.

Table 1. Nominal material property

Components	Density (ρ: lb/in3)	Young's Modulus (E: ksi)	Poisson's Ratio (ν)
Steel E_s	0.283	30,000	0.29
Concrete E_c	0.0868	3,605	0.2

The first four simulated vibration modes of the span are shown in Fig. 10. Because of the relatively short span length, the resonance frequencies are relatively high, with first mode at 8.17 Hz. As expected, the first mode shape shows single-curvature bending, while the higher modes are more complex. Rayleigh damping is set at 2% for the first two resonance frequencies.

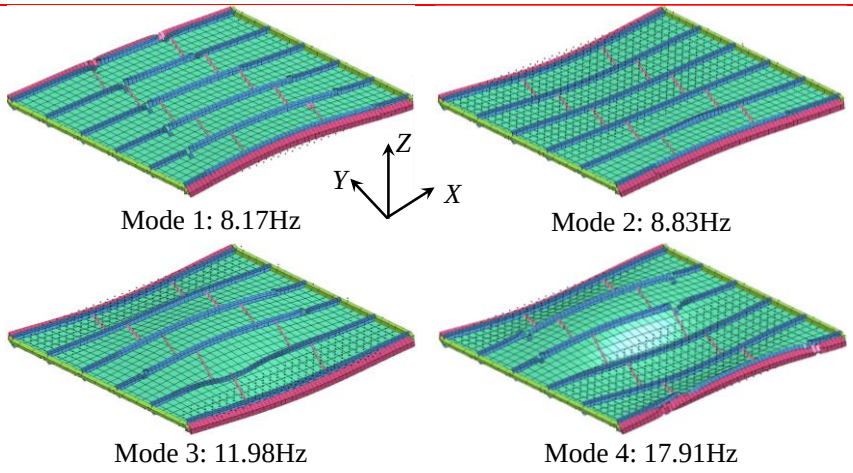


Fig. 10 Mode shape and frequency for the first four modes

3.2. Simulated traffic scenarios

When only one vehicle passes through the bridge, it is relatively easy to estimate the vehicle weight through BWIM. However, when multiple vehicles pass through the bridge at the same time, the problem becomes more challenging. Considering the size of the short bridge span, we focus on three different vehicle scenarios shown in Fig. 11. Because the objective is to identify overloaded large trucks, scenarios with only smaller vehicles (simply named as cars for brevity) are not necessary and thus not considered.

Scenario 1: Only one truck traveling in any one of the two lanes.
Scenario 2: One truck in one lane and one car in another lane.
Scenario 3: One truck in one lane and another truck in another lane.

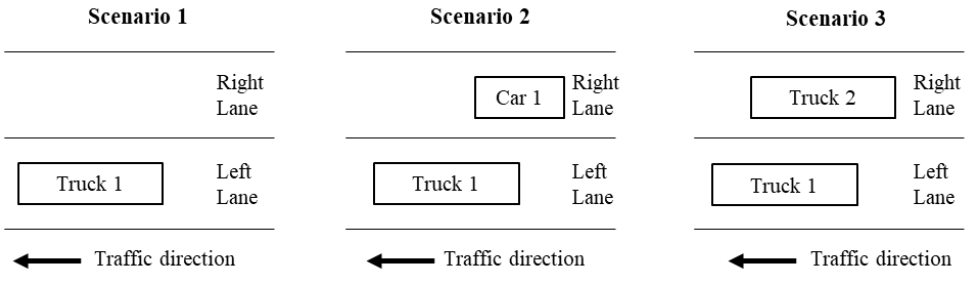


Fig. 11 Different scenarios being considered

For Scenarios 2 and 3, either one of the two vehicles may enter the bridge first, and in any of the two lanes. For all three scenarios, vibration data collection is assumed to start when a vehicle first enters the bridge. In order to mimic realistic traffic, different combinations of axle numbers, axle weights, axle distances, vehicle distances, vehicle speeds and vehicle lateral positions need to be considered. In addition, spring-mass-damper models are used to describe vehicle dynamics (Fig. 12), with randomized combinations of stiffness values and damping values associated with each axle mass. In order to improve the performance of deep learning, uniform distributions are assigned for all parameter randomizations.

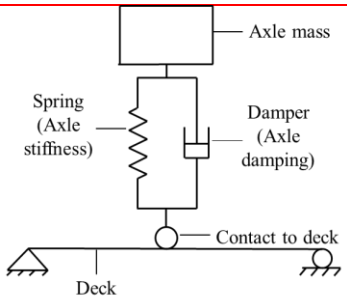


Fig. 12 Spring (axle stiffness) and damper (axle damping) associated with axle mass

(i) Axle number: The limit for a standalone single axle weight is specified as 9.05 tons or 20,340 lbf, while the limit for tandem axle weight is specified as 18.1 tons or 40,680 lbf in the State of Georgia, USA (GDOT, 2020). Here, a tandem axle refers to a group of two or more closely-spaced axles, all within 96 inches and combined together. The official limit of 40,680 lbf is specified for the entire tandem, instead of for every single axle in the tandem. Therefore, toward BWIM, a group of closely-spaced axles within 96 inches is regarded as one tandem axle; we identify their total weight together instead of the weights of individual actual/physical axles. In this study, a car is assumed to have 2 or 3 standalone single axles (the latter allowing a trailer), and no tandem axle. A truck can have 2 ~ 5 axles; each being either a standalone single axle or a tandem axle. For brevity, if not specified, hereinafter the word "axle" by default may refer to either a standalone single axle (not belonging to a tandem), or a tandem axle (a group of two or more closely-spaced axles within 96 inches).

(ii) Axle weight: Similar to the word "axle", we use the phrase "axle weight" to refer to either the weight of a standalone single axle, or the weight of a tandem axle (containing multiple closely-spaced axles). Recall that we assume a car has two or three standalone single axles and does not have any tandem axles. The axle weight in a car can vary from 0.5 to 2.0 tons (1,125 ~ 4,500 lbf). On the other hand, a truck can have 2 ~ 5 standalone single axles and/or tandem axles. In this study, the axle weight in for truck is set as a uniform distribution from 2.0 to 30.0 tons (4,500 ~ 67,550 lbf), which can be much heavier than a car axle. Note that the maximum axle weight of 67,550 lbf is set around 66% overweight of the GDOT limit of 40,680 lbf.

(iii) Axle distance: The phrase "axle distance" represents the distance between the center of two axles, where each axle can be either a standalone single axle or a tandem axle. Depending on vehicle speed, an axle distance is related to the corresponding time gap between the entrances of two axles. The axle distance of a car is randomized from 0.05 to 0.35 seconds; the axle distance of a truck can vary from 0.1 to 0.7 seconds.

(iv) Axle stiffness and damping: As shown in Fig. 12, a mass-spring-damper model describes each axle. The axle resonance frequency ω_n is modeled as a uniform distribution from 1 to 6 Hz (6.28 ~ 37.70 rad/s), while the damping ratio ζ is uniformly distributed from 0% to 20%. Consequently, the stiffness and damping parameters are calculated as $k = m\omega_n^2$ and $c = 2m\omega_n\zeta$.

(v) Vehicle distance: As shown in Fig. 11, Scenarios 2 and 3 both have two vehicles involved. The distance between the two vehicles is also randomized for neural network training. Similar as axle distance,

vehicle distance is modeled by the time gap between the entrance times of two vehicles. The corresponding random variable has a uniform distribution in the interval 0 ~ 1 second.

(vi) Vehicle speed: For efficient data generation and training, the speed of the truck is modeled as a uniform distribution in the interval 30 ~ 80 mph (13.33 ~ 35.56 m/s), while the speed of car is modeled as a uniform distribution in the interval 30 ~ 90 mph (13.33 ~ 40 m/s).

(vii) Transverse vehicle position: As summarized in the literature review, many conventional methods provide inaccurate BWIM results when the transverse vehicle position changes. To avoid this issue, transverse position of a vehicle within the 3.4 meter-wide lane is also randomized in the training data generation. As shown in Fig. 13, the distance from centerline of a car to the lane center is modeled in a uniform distribution -0.6 ~ 0.6 m. In addition, the track width (distance between left and right wheels) of a car is randomized between 1.5 ~ 2 m. Similarly, the distance from the centerline of a truck to the lane center can be -0.4 ~ 0.4 m; the track width of a truck is randomized at 2 ~ 2.6 m.

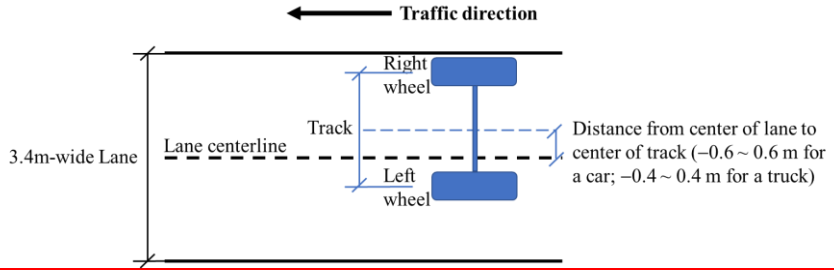


Fig. 13 Transverse wheel positions (plan view)

During the simulation, the bridge vibration data starts to be recorded when the first vehicle enters the bridge. The recording ends after the last vehicle leaves the bridge. The longest measurement recording duration is set as 5 seconds, which is sufficient for the worst-case scenario with long and slow vehicles.

4. Recurrent neural network for Bridge Weigh-in-Motion

Section 4.1 details the how the proposed 3-stage BRNN network is applied on the BWIM application, including the input and output data structures. Section 4.2 describes the performance of the proposed network and compares it with the conventional MFI method.

4.1. Input and output data structures for the proposed neural network

4.1.1. Strain history simulation as input

While performing simulation with each randomized parameter set, strain response at selected locations is recorded as the sensor measurement data for BWIM. As shown in Fig. 14, six "virtual" strain gages are allocated for the entire bridge span, one on each girder. Every strain gage is installed on the bottom flange of a girder. The sampling frequency is set as 200 Hz. A low-pass Butterworth filter is applied to the strain data, with cutoff frequency at 30 Hz. As a result, six strain histories at 200 Hz for five seconds provide a 6-by-1,000 matrix (1,000 samples in every strain history).

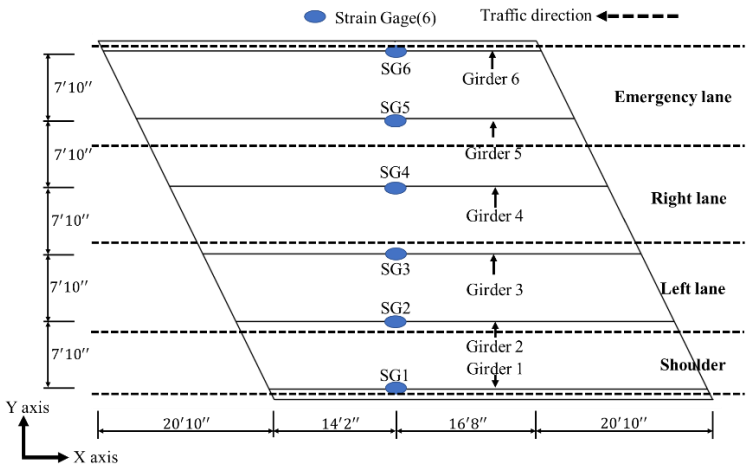


Fig. 14 Six strain measurement locations

For each of the three traffic scenarios in Fig. 11, simulations are performed by randomizing the parameters described in Section 3.2. Fig. 15 plots 2.5 seconds of example strain data for simulating a Scenario 3 with two trucks driving through. The truck on the left lane enters the bridge first, while the truck on the right lane enters the bridge later. As a result, the peaks appear first in strain gages SG2 and SG3 (under the left lane), and later in SG4 (under the right lane). Since no vehicle traveled in the emergency lane, SG6 is farthest away from traffic and shows lowest amplitude as expected.

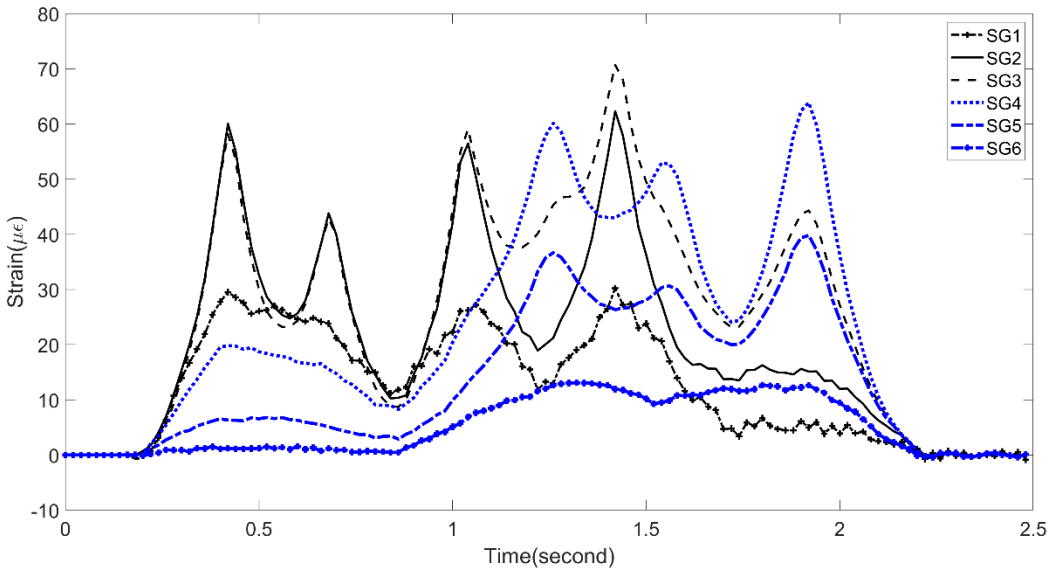


Fig. 15 Strain response for two trucks driving through

In total, LS-DYNA performed 10,000 randomized simulations of Scenario 1, 15,000 randomized simulations of Scenario 2, and 20,000 randomized simulations of Scenario 3. Towards deep learning, these original simulation data sets are then divided into training, validation and test sets. For each of the three scenarios, 80% of the original data sets are used for training, another 10% are used for validation, and the final 10% are used for testing. For each scenario, these data set numbers from original LS-DYNA simulation are summarized in the first half of Table 2.

Table 2. The number of original data sets and augmented data sets

Different cases	Scenario 1	Scenario 2	Scenario 3
Original simulation data sets $\tilde{\mathbf{x}}$ (total number)	10,000	15,000	20,000
Training set (80%)	8,000	12,000	16,000
Validation set (10%)	1,000	1,500	2,000
Test set (10%)	1,000	1,500	2,000
Augmented data sets **x** (total number)	100,000	150,000	200,000
Training set (80%)	80,000	120,000	160,000
Validation set (10%)	10,000	15,000	20,000
Test set (10%)	10,000	15,000	20,000

4.1.2. Input and output data of the BWIM neural network

To mimic practical application, Gaussian noise is added to the original data. For each sensor time history $\tilde{\mathbf{x}} \in \mathbb{R}^{1,000}$ (recall 200 Hz for 5 secs), the standard deviation is first evaluated as s. The corresponding noise vector $\mathbf{n} \in \mathbb{R}^{1,000}$ is formed with a standard normal distribution where $n_i \sim \mathcal{N}(0,1)$. To adjust the noise level, we use variable η with ten different percentage values, at $1\%, 2\%, \cdots, 10\%$. The corresponding noisy history $\mathbf{x} \in \mathbb{R}^{1,000}$ produced from original simulated history $\tilde{\mathbf{x}}$ is thus:

$$\mathbf{x} = \tilde{\mathbf{x}} + \eta \cdot s \cdot \mathbf{n} \tag{20}$$

Since the noise level η has ten different values, each original data set $\tilde{\mathbf{x}}$ provides ten augmented data sets **x**. As a result, taking Scenario 1 for example, the 8,000 training data sets from the original simulation is augmented into 80,000 data sets for the actual network training. As shown in Table 2, similar augmentation took place for the validation and test sets of Scenario 1, and for the data sets of other two scenarios. The augmentation has two benefits. First, ten times more data sets become immediately available for training the neural network, without entailing computationally expensive dynamic time history simulation of the highly detailed FEM model in LS-DYNA. Second, training the network with varied sensor noise levels makes the network more robust against sensor noises. Standard normalization is performed on the augmented strain history prior to feeding into the neural network.

As described earlier, five seconds of six strain measurements at 200Hz provide 6,000 data points. As input to the neural network, a natural choice would be setting $n_x = 6$ and $T_x = 1000$, i.e. each $\mathbf{x}(t_x) \in \mathbb{R}^6$ with the sequence step number $t_x = 1,2, \ldots, 1000$. However, despite the ability of LSTM modules handling longer input sequences, $T_x = 1000$ still proves to be impractical if not infeasible given current hardware limitations (Cho et al., 2014). Thus, reshaping is performed to the input data and illustrated in Fig. 16. The entire sequence of 1,000 sampling steps is divided into T_x =100 pieces of 10 steps (the corresponding time length for each piece of 10 sampling steps is 0.05 seconds = 10×1/200Hz). Ten samples from six strain sensors together are taken as input at one time step t_x to the network. Therefore, the length of each input vector $\mathbf{x}(t_x)$ is $n_x = 6 \times 10 = 60$. Fig. 16 illustrates how to transform the original 1,000 samples at six

strain gages into new shapes, data from the first two sampling steps at stacked as the beginning part of $\mathbf{x}(1) \in \mathbb{R}^{60}$, and similarly data from the final two sampling steps are stacked as the ending part of $\mathbf{x}(100) \in \mathbb{R}^{60}$. In summary, this reshaping uses hyperparameters $n_x = 60$ and $T_x = 100$.

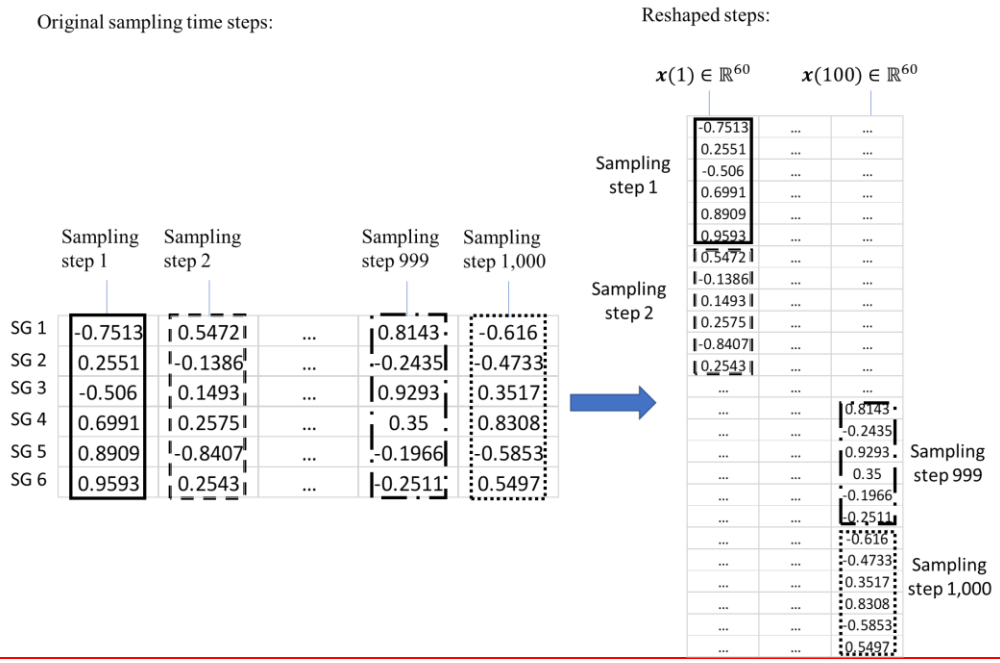


Fig. 16 Input for recurrent neural network when 10×6 samples are lumped as one input sequence step

It's obvious that different reshaping of the strain gage data could have been performed toward input into the network. To investigate the effect of different dimension hyperparameters, the following sections will also study a different case of $n_x = 30$ and $T_x = 200$. In other words, 0.025 seconds (i.e. 5 sampling steps) of six strain gage data are used as one input sequence.

The output of a BWIM neural network contains axle weights. As introduced earlier, a known output used for training a neural network is termed a label. As shown in Fig. 11, the test span has 2 vehicles traveling at most. Also recall that the weight of a tandem axle is considered altogether, and we assume that the maximum number of axles (either standalone or tandem) per vehicle is 5 axles on a truck. Thus, maximum number of axle weights on the test span is 5+5 = 10. This is set as the length of output sequence, $T_y = 10$, and the output at each step is a scalar $\hat{y}(t_y) \in \mathbb{R}$ that represents one out of the 10 axle weights. The first five numbers represent the axle weights in the left lane, while the last five numbers represent the axle weights in the right lane. If there is no vehicle in one lane, all the five corresponding values should be zero for an output label. If the number of axles is smaller than five, then the remaining output entries should be zero. Taking Fig. 17 for example, suppose that the truck is in the left lane. Then, the output label should have $y(1) = 15.5$ kip, $y(2) = 21.3$ kip for the tandem axle, and $y(3) = \cdots = y(10) = 0$.

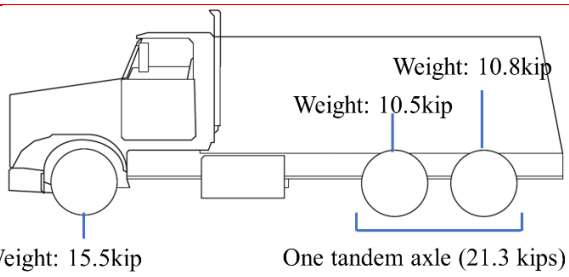


Fig. 17 A truck with one standalone front axle and one tandem axle that includes two closely-spaced rear axles

4.2. BRNN performance and comparison with MFI

4.2.1. BRNN results

As described in Section 4.1.2, we study two sets of dimension hyperparameters for the input data: (i) $T_x = 100$, i.e. $n_x = 60$; (ii) $T_x = 200$, i.e. $n_x = 30$. For both input reshaping cases, the length of output sequence is $T_y = 10$ and the output at each step is scalar $y(t_y) \in \mathbb{R}$. In addition, two other major dimension hyperparameters are listed as follows.

(i) As shown in Stage-1 of Fig. 5, the pre-attention BRNN with LSTM, both forward and backward intermediate variables $\mathbf{a}^{\mathrm{f}}(t_x)$ and $\mathbf{a}^{\mathrm{b}}(t_x)$ have dimension n_a. After some trial-and-error, n_a is chosen to be 1,000 in this study.

(ii) Both Stage-2 and Stage-3 involve another post-attention BRNN intermediate variable $\mathbf{s}(t_y) \in \mathbb{R}^{n_s}$, whose dimension is denoted as n_s and chosen as 2,000.

The proposed BRNN is trained using the augmented simulation data from all three scenarios (Table 2). With n_a and n_s fixed above, the network performance is studied for two cases of input hyperparameters: (i) $T_x = 100$; (ii) $T_x = 200$. In this BWIM application, we adopt a simple performance index that quantifies the relative error of truck weight identification, averaged between all trucks in the simulation.

$$e = \frac{1}{m}\sum_{i=1}^{m} \frac{\left|W(i) - \widehat{W}(i)\right|}{W(i)} \tag{21}$$

Here $W(i) \in \mathbb{R}$ is the actual weight of the i-th truck, $\widehat{W}(i) \in \mathbb{R}$ is the identified whole weight of the i-th truck and m represents the total number of trucks. Since cars are much lighter in BWIM, they are neglected in this index. Table 3 shows the error index e for the training set, validation set and test set. The time required to train the neural network doubles as the number of input T_x increases from 100 to 200. The increase in computation time brings about better network performance across all three data sets. The BRNN parameters trained from $T_x = 200$ are used next for comparison with MFI.

Table 3. BRNN performance evaluated by error index e

T_x	Training time (hour)	Training error e	Validation error e	Test error e
100	67.73	1.54%	2.65%	2.38%
200	116.37	1.39%	2.32%	1.88%

4.2.2. Comparison with MFI method

This section compares the performance of the proposed BRNN approach with the traditional moving force identification (MFI) method, which identifies the time history of contact forces between the bridge deck and vehicle axles using a finite element model. The time history of contact forces is then used to derive the axle weights of a vehicle. In particular, the comparison is performed with MFI with dynamic

programming (Law and Fang, 2001; Gonzalez et al., 2008). The basic formulation of the MFI method starts with the bridge dynamics equation for an *N*-DOF model:

$$\mathbf{M}\ddot{\mathbf{q}}(t) + \mathbf{C}\dot{\mathbf{q}}(t) + \mathbf{K}\mathbf{q}(t) = \mathbf{L}(t)\mathbf{p}(t) \tag{22}$$

where $\mathbf{q}(t) \in \mathbb{R}^N$ is the displacement vector; $\mathbf{M} \in \mathbb{R}^{N\times N}$, $\mathbf{C} \in \mathbb{R}^{N\times N}$, and $\mathbf{K} \in \mathbb{R}^{N\times N}$ are the time invariant mass, Rayleigh damping, and stiffness matrices of the bridge model; $\mathbf{p} \in \mathbb{R}^{n_u}$ is the external force vector, containing axle weights in BWIM application, and n_u represents the number of forces to be identified, which varies from two to ten in the three traffic scenarios under study. Finally, $\mathbf{L}(t) \in \mathbb{R}^{N\times n_u}$ is the time-varying force location matrix that represents the time-changing locations of vehicles. Thus, the MFI approach requires accurate vehicle positions traveling through the bridge.

Since the total DOFs of the bridge is large, model reduction is necessary for computational efficiency. The total number of DOFs after model reduction is $n_z \in \mathbb{R}$.

$$\mathbf{q}(t) \approx \mathbf{\Phi}\mathbf{z}(t) \tag{23}$$

where $\mathbf{\Phi} \in \mathbb{R}^{N\times n_z}$ is a matrix with the first n_z normalized eigenvector; $\mathbf{z}(t) \in \mathbb{R}^{n_z}$ is the modal coordinates. We use $\alpha \in \mathbb{R}$ and $\beta \in \mathbb{R}$ to denote the Rayleigh damping coefficients, and $\mathbf{\Omega} \in \mathbb{R}^{n_z\times n_z}$ to denote a diagonal matrix whose diagonal entries contain the first n_z number of resonance frequenices.

$$\ddot{\mathbf{z}}(t) + \left(\alpha\mathbf{I}_{n_z\times n_z} + \beta\mathbf{\Omega}\right)\dot{\mathbf{z}}(t) + \mathbf{\Omega}\mathbf{z}(t) = \mathbf{\Phi}^{\mathrm{T}}\mathbf{L}(t)\mathbf{p}(t) \tag{24}$$

In this numerical example, we keep the first fifty modes, i.e. $n_z = 50$. The corresponding frequencies range from 8.17 Hz to 140.24 Hz, which should be sufficient for the MFI method to capture the bridge dynamics. From modal coordinates $\mathbf{z}(t)$, standard state-space formulation is introduced.

$$\dot{\mathbf{X}}(t) = \mathbf{A}\mathbf{X}(t) + \mathbf{B}(t)\mathbf{p}(t) \tag{25}$$

where $\mathbf{X}(t) = \begin{bmatrix}\mathbf{z}(t)\\ \dot{\mathbf{z}}(t)\end{bmatrix} \in \mathbb{R}^{2n_z}$, $\mathbf{A} = \begin{bmatrix}\mathbf{0} & \mathbf{I}\\ -\mathbf{\Omega} & -(\alpha\mathbf{I} + \beta\mathbf{\Omega})\end{bmatrix} \in \mathbb{R}^{2n_z\times 2n_z}$ and $\mathbf{B}(t) = \begin{bmatrix}\mathbf{0}_{n_z\times n_u}\\ \mathbf{\Phi}^{\mathrm{T}}\mathbf{L}(t)\end{bmatrix} \in \mathbb{R}^{2n_z\times n_u}$. The continuous-time state-space formulation is then converted to discrete-time domain using standard zero-order hold, with certain sampling time period. Using discrete-time index *j*, the state and input vectors are denoted as $\mathbf{X}_j$ and $\mathbf{p}_j$, respectively. Suppose there are n_m number of sensor measurements. We use $\boldsymbol{\varepsilon}_j^{\mathrm{sim}} \in \mathbb{R}^{n_m}$ to denote the simulated strain vector at the *j*-th time step. The vector can be obtained by a simple linear transformation from state vector $\mathbf{X}_j$ that contains displacements information. In the sense that once the structural model is chosen, strain history is determined by the excitation history $= \{\mathbf{p}_0, \mathbf{p}_1, \dots, \mathbf{p}_{n_t}\}$, we can denote the simulated strain $\boldsymbol{\varepsilon}_j^{\mathrm{sim}}(\boldsymbol{\mathcal{P}})$ as a function of the input $\boldsymbol{\mathcal{P}}$.

While MFI attempts to minimize the difference between simulated and measured strain, early researchers found first-order Tikhonov regularization significantly improves MFI performance (Gonzalez, et al., 2008). The regularization takes $\boldsymbol{\mathcal{P}}$ as the optimization variable and minimizes not only the difference between simulated $\boldsymbol{\varepsilon}_j^{\mathrm{sim}}$ and measured strain $\boldsymbol{\varepsilon}_j^{\mathrm{meas}}$, but also the input change between two neighboring time steps.

$$\underset{\boldsymbol{\mathcal{P}}=\{\mathbf{p}_0,\dots,\mathbf{p}_{n_t}\}}{\text{minimize}} \sum_{j=0}^{n_t-1}\left[\left(\boldsymbol{\varepsilon}_j^{\mathrm{meas}} - \boldsymbol{\varepsilon}_j^{\mathrm{sim}}(\boldsymbol{\mathcal{P}})\right)^{\mathrm{T}}\mathbf{W}\left(\boldsymbol{\varepsilon}_j^{\mathrm{meas}} - \boldsymbol{\varepsilon}_j^{\mathrm{sim}}(\boldsymbol{\mathcal{P}})\right) + \left(\mathbf{p}_{j+1} - \mathbf{p}_j\right)^{\mathrm{T}}\mathbf{R}\left(\mathbf{p}_{j+1} - \mathbf{p}_j\right)\right. \tag{26}$$

Here $\mathbf{W} \in \mathbb{R}^{n_m\times n_m}$ is the weighting matrix (usually chosen as identity); $\mathbf{R} \in \mathbb{R}^{n_u\times n_u}$ is the regularization matrix and is required to be tuned.

As mentioned earlier, the application of moving force identification requires information including axle number, axle distance, vehicle distance, vehicle speed and transverse vehicle position, etc. (Meanwhile, it is worth noting that when estimating vehicle weight the trained BRNN does not require any such vehicle information; all that's required by BRNN is strain history). While these data can be measured through certain instruments on the bridge, the measurements inevitably come with errors. For example, in practice the number of vehicle axles can be difficult to capture by sensors, and a mistaken number of axles can cause significant error in the MFI results. When comparing with BRNN, we assume the axle number is captured

correctly so that MFI can provide better (and more comparable) performance. Instead, we consider the effects of more commonly occurring errors to MFI, by introducing from 0% to 10% errors to vehicle information including axle distance, vehicle distance, vehicle speed and transverse vehicle position.

In the comparison between MFI and the proposed BRNN approach, four examples are chosen randomly from the three traffic scenarios (two examples from Scenario 1, and one each from Scenarios 2 and 3). Table 4 shows the truck weight estimation error by BRNN and MFI, for the trucks in different examples. Among all the five trucks from four examples, the BRNN provides the lowest error of 0.1% and highest error of 2.00%. When the "correct" vehicle information is available to MFI, a largest error of 6.34% is observed. In addition, when the vehicle information with "error" is used by MFI, all estimation errors are highly unacceptable. The fact that the trained BRNN does not require any such vehicle information for estimating vehicle weight is shown to be a significant advantage.

Table 4. Relative error of truck weight estimation for four examples by BRNN and MFI

Examples		Truck weight(lbf)	BRNN	MFI - "correct" vehicle info	MFI - vehicle info with "error"
Ex. 1 – 2-axle truck (Scenario 1)		75,825.8	0.83%	4.09%	16.90%
Ex. 2 – 3-axle truck (Scenario 1)		60,988.0	2.00%	1.19%	18.42%
Ex. 3 – 3-axle car and 2-axle truck (Scenario 2)		49,574.0	0.10%	6.34%	38.68%
Ex. 4 – two trucks (Scenario 3)	Truck A (4 axles)	91,173.2	1.46%	3.33%	28.99%
	Truck B (3 axles)	79,647.2	1.48%	1.76%	25.22%

5. Summary and discussion

This paper proposes a novel 3-stage BRNN network for bridge weigh-in-motion. The proposed neural network incorporates both long short-term memory (LSTM) and attention mechanism to improve the network performance. The neural network is trained with strain measurements as input and axle weights as output. In order to acquire the large data volume to train the network, a finite element bridge model is built through the commercial software package LS-DYNA. To mimic everyday traffic scenarios, tens of thousands of randomized vehicle formations are simulated, with different combinations of vehicle types, spacings, speeds, axle weights, axle distances, etc. Dynamic response from each of the randomized traffic scenarios is recorded for training the BRNN.

Numerical results demonstrate that the BRNN network achieves high accuracy in estimating vehicle weights, in comparison with a conventional moving force identification (MFI) method. Furthermore, the MFI estimation performance is shown to deteriorate rapidly with errors in the vehicle information such as speed, transverse position, vehicle interval and axle distances. The fact that the trained BRNN does not require such information for estimating vehicle weight proves to be a significant advantage.

While the proposed BRNN method is shown to accurately identify vehicle weight using simulated bridge data, future research is needed to validate the performance in field experiments. Though we have modeled 10% Gaussian error in strain measurements, there can still be other sources of uncertainties in practice. Secondly, future simplification of the neural network can help speed up the training process and reduce the demand for training data sets.

References

Chan, T. H. T., Law, S. S., Yung, T. H. & Yuan, X. R. (1999), An interpretive method for moving force identification, *Journal of Sound and Vibration*, **219**(3), 503-524.

Cho, K., Merrienboer, B., Bahdanau, D. & Bengio, Y. (2014), On the properties of neural machine translation: encoder-decoder approaches, *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, Doha, Qatar, October 25, 2014.

Chorowski, J., Bahdanau, D., Cho, K. & Bengio, Y. (2014), End-to-end continuous speech recognition using attention-based recurrent NN: first results, *Deep Learning and Representation Learning Workshop: NIPS 2014*, Montreal, Quebec, 8-13 December 2014.

Chung, J., Gulcehre, C., Cho, K. & Bengio, Y. (2014), Empirical evaluation of gated recurrent neural networks on sequence modeling, *Deep Learning and Representation Learning Workshop: NIPS 2014*, Montreal, Quebec.

Das, P. & Deka, G. (2016), *History and Evolution of GPU Architecture*, IGI Global, Hershey, PA, USA.

Fu, G. & Hag-Elsafi, O. (2000), Vehicular overloads: load model, bridge safety, and permit checking, *Journal of Bridge Engineering of ASCE*, **5**(1), 49-57.

GDOT. (2020), Highways, bridges and ferries, in Goverment of Georgia (ed.), Official Code of Georgia.

Gonzalez, A., Rowley, C. & OBrien, E. (2008), A general solution to the identification of moving vehicle forces on a bridge, *Internaltional Journal for Numerical Methods in Engineering*, **75**, 335–354.

Goodfellow, I., Bengio, Y. & Courville, A. (2016), *Deep Learning*, MIT Press, Cambridge, MA, USA.

Graves, A., Mohamed, A. & Hinton, G. (2013), Speech recognition with deep recurrent neural networks, *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, Vancouver, British Columbia, 26-31 May 2013.

Hochreiter, S. & Schmidhuber, J. (1997), Long short-term memory, *Neural Computation*, **9**, 1735-1780.

Kawakatsu, T., Aihara, K., Takasu, A. & Adachi, J. (2019), Deep sensing approach to single-sensor vehicle weighing system on bridges, *IEEE Sensors Journal*, **19**(1), 243-256.

Kim, S., Lee, J., Park, M.-S. & Jo, B.-W. (2009), Vehicle signal analysis using artificial neural networks for a bridge weigh-in-motion system, *IEEE Sensors Journal*, **9**(10), 7943-7956.

Krizhevsky, A., Sutskever, I. & Hinton, G. (2012), ImageNet classification with deep convolutional neural networks, *Commun. ACM*, **60**(6), 84-90.

Law, S. S., Chan, T. H. T. & Zeng, Q. H. (1997), Moving force identification: a time domain method, *Journal of Sound and Vibration*, **201**(1), 1-22.

Law, S. S., Chan, T. H. T. & Zeng, Q. H. (1999), Moving force identification—a frequency and time domains analysis, *Journal of Dynamic Systems, Measurement, and Control of ASME*, **121**(3), 394-401.

Law, S. S. & Fang, Y. L. (2001), Moving force identification: optimal state estimation approach, *Journal of Sound and Vibration*, **239**(2), 233-254.

LeCun, Y., Bengio, Y. & Hinton, G. (2015), Deep learning, *Nature*, **521**(7553), 436-444.

LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P. (1998), Gradient-based learning applied to document recognition, *Proceedings of the IEEE*, **86**, 2278-2324.

Lydon, M., Taylor, S. E., Robinson, D., Mufti, A. & Brien, E. J. O. (2016), Recent Developments in Bridge Weigh in Motion (B-Wim), *Journal of Civil Structural Health Monitoring*, **6**(1), 69-81.

Minsky, M. & Papert, S. (1969), *Perceptrons: An Introduction to Computational Geometry*, MIT Press, Cambridge, MA, USA.

Moses, F. (1979), Weigh-in-motion system using instrumented bridges, *Transportation Engineering Journal of ASCE*, **105**(3), 233–249.

O'Connor, C. & Chan, T. H. T. (1988), Dynamic wheel loads from bridge strains, *Journal of Structural Engineering of ASCE*, **114**(8), 1703-1723.

Quilligan, M. (2003), Bridge weigh-in motion : development of a 2-D multi-vehicle algorithm, *Trita-BKN. Bulletin*, Byggvetenskap, Stockholm, pp. viii, 144.

Rosenblatt, F. (1957), *The Perceptron — a Perceiving and Recognizing Automaton*, Cornell Aeronautical Laboratory.

Rumelhart, D. E., Hinton, G. E. & Williams, R. J. (1986), Learning representations by back-propagating errors, *Nature*, **323**, 533-536.

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

Selfridge, O. G. (1959), Pandemonium: a paradigm for learning, *Neurocomputing: Foundations of Research*, pp. 115-122.

Skokandic, D., Znidaric, A., Mandic-Ivankovic, A. & Kreslin, M. (2017), Application of bridge weigh-in-motion measurements in assessment of existing road bridges, *The Value of Structural Health Monitoring for the Reliable Bridge Management*, Zagreb, Croatia, 2-3 March

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. (2014), Dropout: a simple way to prevent neural networks from overfitting, *Journal of Machine Learning Research*, **15**(56), 1929−1958.

Sutskever, I., Vinyals, O. & Le, Q. V. (2014), Sequence to sequence learning with neural networks, *2014 Neural Information Processing Systems*, Montreal, Quebec, 8-13, December, 2014.

Sutskever, I., Vinyals, O. & Le, Q. V. (2014), Sequence to sequence learning with neural networks, *Proc. Advances in Neural Information Processing Systems*, Montreal, Quebec, 8-13, December, 2014.

Tikhonov, A. N. & Arsenin, V. Y. (1977), *Solutions of Ill-Posed Problems*, Kluwer Academic Publishers.

Wu, S. & Shi, Z. (2006), Identification of vehicle axle loads based on FEM-Wavelet-Galerkin method, *Journal of Vibration Engineering*, **19**(4), 494–498.

Yang, J. C., Yu, K., Gong, Y. H. & Huang, T. (2009), Linear spatial pyramid matching using sparse coding for image classification, *2009 IEEE Conference on Computer Vision and Pattern Recognition*, Miami, Flordia, 20-25 June 2009.

Yu, Y., Cai, C. S. & Deng, L. (2016), State-of-the-art review on bridge weigh-in-motion technology, *Advances in Structural Engineering of SAGE*, **19**(9), 1514-1530.

Zhang, Q. R., Zhang, M., Chen, T. H., Sun, Z. F., Ma, Y. Z. & Yu, B. (2019), Recent advances in convolutional neural network acceleration, *Neurocomputing*, **323**, 37-51.

Zhang, R., Lv, W. & Guo, Y. (2010), A vehicle weigh-in-motion system based on Hopfield neural network adaptive filter, *2010 International Conference on Communications and Mobile Computing*, Shenzhen, Guangdong.

Zhu, X. Q. & Law, S. S. (2015), Structural health monitoring based on vehicle-bridge interaction: accomplishments and challenges, *Advances in Structural Engineering of SAGE*, **18**(12), 1999-2015.

Zhu, X. Q. & Law, S. S. (2016), Recent developments in inverse problems of vehicle–bridge interaction dynamics, *Journal of Civil Structural Health Monitoring*, **6**(1), 107-128.

Zhu, X. Q., Law, S. S. & Bu, J. Q. (2006), A state space formulation for moving loads identification, *Journal of Vibration and Acoustics of ASME*, **128**(4), 509-520.