Damage Detection for Structural Health Monitoring Under Uncertainty using Deep Interval Neural Networks

David Betancourt[1,3)], Steffen Freitag[2)] and Rafi L. Muhanna[3)]

[1)] *School of Computational Science and Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA, david.betancourt@gatech.edu*

[2)] *Institute for Structural Mechanics, Ruhr University Bochum, Germany, steffen.freitag@sd.rub.de*

[3)] *School of Civil and Environmental Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA, rafi.muhanna@gatech.edu*

Abstract. Most approaches for damage identification tasks in Structural Health Monitoring (SHM) are solid-mechanics based and are generally not designed for real-time damage identification. Such approaches require the comparison of responses from a finite element (FE) model analysis with post-processed sensor data in order to assess damage. On the other hand, more recent data-driven approaches in SHM that use deep neural networks (DNNs) have shown promising results for real-time damage identification. Such DNNs have been used for multiple SHM tasks for real-time damage identification without requiring an FE model. However, DNNs do not have an implicit mechanism to include input and parameter uncertainty. For this work, a novel supervised interval deep neural network (DINN) that can process input sensor uncertainty for damage detection and classification tasks in SHM is proposed. The proposed method can detect and classify damage in an SHM system solely from sensor data. For other SHM tasks, such as damage localization and severity, our method can be combined with FE models.

Keywords: deep learning, structural health monitoring, interval deep learning, epistemic uncertainty, interval finite element, machine learning

1. Introduction

Structural health monitoring (SHM) is typically concerned with the damage identification and system characterization of engineering structures. An SHM program consists of a network of sensors connected to the structural system, which measure an observed physical variable. Hereafter, we refer to *damage* as structural damage and define it as adverse changes to the stiffness of the structural system which compromise its integrity. An important concern in monitoring systems is the uncertainty in sensor measurements. In SHM systems, uncertainty can be due to environmental factors, sensor malfunctions, unknown sensor placements, miscalibrations, and errors in data acquisition which cumulatively contribute to imprecision in the sensor data.

There are two general approaches for detecting and identifying damage in SHM. The first is solid mechanics-based[1], while the second one is data-driven. The solid mechanics-based approach models the actual structure, usually via a finite element (FE) model, from which the structural responses are obtained. These responses (e.g., displacements, stresses, modes of vibration) are compared with signal-processed sensor data to predict whether there is damage. On the other hand, the data-driven approach used for this work uses a novel interval deep learning algorithm on the sensor data to detect or classify damage with or without the aid of an FE model, depending on the complexity of SHM task.

Data-driven techniques for SHM have been previously explored, mostly with unsupervised algorithms. In (Figueiredo et al., 2009) the authors present unsupervised methods for damage detection using different small-scale datasets including an 'autoassociative neural network,' which is conceptually equivalent to an autoencoder. Other works, e.g., (Neves et al., 2017), typically use simulation data from an FE model to detect damage. To account for sensor uncertainty, in (Figueiredo et al., 2011) the authors include noise in the observations to account for operational and environmental variability. More recently, (Figueiredo and Santos, 2018) the authors examine some of the same unsupervised algorithms on the Z24 bridge dataset. Around the same time, deep learning techniques have begun to be used in SHM. In (Wang et al., 2018) the authors use autoencoders to detect and identify damage using simulation data for training. In (Sarkar et al., 2016) the authors use autoencoders and image segmentation to detect and identify damage in aircraft structures. They use training data from images of carbon fiber-reinforced coupons subjected to different damage types.

The data-driven SHM techniques surveyed showed some success—albeit largely limited to damage detection at best and a high rate of false positives at worst, since the unsupervised techniques can detect *any* data anomaly. Indeed, any data anomaly does not guarantee correspondence with structural damage. Another setback is that in most cases the training data comes from FE simulations alone—not from real SHM systems—because data in SHM is difficult to obtain. In this work, we consider the problem of SHM real-time damage detection and classification under uncertainty using a supervised deep learning algorithm with interval analysis (IA). In particular, our contribution consists of developing the Deep Interval Neural Network (DINN) in a classification setting for SHM tasks. The Z24 bridge dataset is used to test our proposed methods (Krämer et al., 1999).

2. Methodologies for Damage Identification in SHM

In this section we introduce the relevant background for SHM tasks and the tools to solve them (Sec. 2.1). We also introduce the challenges with uncertainty in SHM sensor data in Sec. 2.2. Finally, we present important discussion on how to take the most advantage of machine learning to solve SHM tasks in Sec. 2.3, and based on that how we use the DINN to solve SHM tasks.

2.1. Damage Identification Tasks in SHM

Damage identification for SHM consists of the following hierarchical tasks (Figueiredo et al., 2011):

[1] solid mechanics-based is also known as physics-based in the SHM literature.

1. Damage Detection: detect whether damage (anomaly) exists.
2. Damage Localization
3. Damage Type
4. Damage Severity
5. Damage Prognosis: remaining useful lifeline.

Task 1 detects whether there is damage in the structure. Tasks 2-4 encompass the diagnosis of the damage, while Task 5 encompasses the prognosis of the damage. Solid mechanics-based or data-driven techniques can be used for each of these hierarchical tasks. Our goal is to demonstrate that the classification DINN developed in this work can be used for these damage identification tasks more efficiently (i.e., in real-time) and accurately than state-of-the-art techniques. For some tasks, such as damage detection, the DINN can be used without a solid mechanics-based model (e.g., an FE model), while for other tasks it can be used in conjunction with the FE model and/or other techniques to enhance the damage identification.

2.2. Uncertainty in SHM sensor data

Environmental and operational changes along with hardware and software errors are significant sources of uncertainty in the data collected by the SHM sensor network. For instance, some SHM damage identification techniques compute inter-story drifts in the structure from double integration of acceleration data. However, because of uncertainty in the measurements, it is difficult to predict accurate story drifts (Kaya and Safak, 2019) that can detect damage.

Overall, the simplifying common approach of adding random noise to the data cannot account for uncertainty in measurement errors when we cannot calibrate the instrument because the measurements performed by this instrument are the most accurate possible (Sun et al., 2017). Many times this is the case in engineering problems—and there is not enough knowledge to build a probabilistic model to account for the uncertainty. For such cases, interval uncertainty can be beneficial for data analysis via interval analysis (IA).

2.3. SHM Anomaly Detection with Machine Learning

SHM damage identification tasks can be framed as anomaly detection and activity recognition tasks for machine learning algorithms. Well-known applications which use anomaly detection with machine learning are transactional fraud, disease detection, and cybersecurity intrusions. Strictly speaking, in anomaly detection we seek to find rare events or observations that vary significantly from the main distribution of the training data. There are two main categories for anomaly detection with machine learning: supervised and unsupervised. Supervised learning constitutes having a fully-labeled set. Unsupervised learning techniques constitute clustering algorithms, which seek to cluster the normal examples and anomalies get classified in a different cluster. Examples of instances of unsupervised models include k-means, Gaussian Mixture models, DBSCAN, and OPTICS. Within

supervised learning, we can use specialized techniques, such as semi-supervised algorithms, which use signals in the training data itself to provide the labels.

In most cases in machine learning, we only have access to training data only with normal examples or with very few examples of anomalies. Thus, because of the class unbalance, unsupervised algorithms are the main option to detect anomalies. As a result, most of the work in this field has been unsupervised due to the nature of the data. However, while unsupervised learning algorithms can prove to be efficient to *detect* anomalies by clustering, they do not provide additional information on what types of anomalies it has found. Even in the simple case of binary anomaly detection, we seek a particular type of anomaly. Thus, detecting any type of data anomaly often results in a high false positive rate, which can be very costly in safety applications. Furthermore, while there is value in binary detection of anomalies, more often than not the main value lies in classifying the types of anomalies—we must know what type of anomaly exits in order to make an informed decision. In order to do this, we need to employ a supervised or semi-supervised algorithm that can train on a balanced enough dataset. The task therein, before we start training, is to gain access to enough instances of anomalies. These instances can be a combination of real measurements—from laboratory experiments or in-situ damage—and from simulations. For instance, the simulation in SHM would consist of using an FE model to simulate different types of damage and produce synthetic training data.

In machine learning, the main goal is to generalize—that is, to make good predictions on unseen data. For example, with disease detection, the algorithm can be trained with instances of a subset of patients $\mathcal{A}$ from the general population, and once trained, to make predictions on a different subset of patients $\mathcal{B}$—which was not used for training the ML model—and detect potential diseases. In the case of SHM, we can use training data from system $\mathcal{A}$ to make predictions on system $\mathcal{B}$. System $\mathcal{B}$ can be fine-tuned with its own SHM data and/or simulations from its own FE model, but the training should not start from scratch. This is indeed a different mindset from, and an advantage on, the solid-mechanics approach to SHM—since in that approach each system requires its own models.

We will use the DINN as a supervised algorithm as explained in the following sections.

3. Interval Deep Learning for SHM under Uncertainty

3.1. Classification DINN

The interval notation for this work is defined as follows. Let $\mathbb{IR}$ be the set of interval real numbers. Interval real numbers are a closed and bounded set of real numbers (Moore et al., 2009), such that if $\mathbf{X} \in \mathbb{IR}$ is an interval, its endpoints are $\underline{X} = \inf(\mathbf{X}) \in \mathbb{R}$ and $\overline{X} = \sup(\mathbf{X}) \in \mathbb{R}$. *Henceforth, boldface is used to represent interval numbers, matrices, and vectors.* The basic arithmetic operations with real intervals are different than with real numbers. For the reader unfamiliar with IA refer to, e.g., (Moore et al., 2009),(Moore et al., 2009), (Neumaier, 1990), (Kreinovich et al., 2013).

The classification Deep Interval Neural Network (DINN) developed in this work is applied to damage identification tasks in SHM. The classification DINN is the interval extension of a real-valued DNN to process interval-valued matrices of any dimensionality (i.e., interval tensors) and output predictive classes with uncertainty bounds around their respective softmax probabilities.

Without loss of generality, the classification DINN is a supervised learning algorithm which seeks to learn an interval predictive model $F : [\underline{X}, \overline{X}] \to y \in \{1, \ldots, C\}$, where C is the number of deterministic classes. To achieve this goal, the DINN is trained using the $\mathbf{X}$ interval matrix inputs (*features*) in a d-dimensional space with n examples, along with its known y categorical labels. This composes the training set of n examples $\mathcal{T} = \{(\mathbf{X}_1, y_1), \ldots, (\mathbf{X}_n, y_n)\}$ where $\mathbf{X} \in \mathbb{IR}^{n \times d}$ is the interval feature space and $y \in \{1, \ldots, C\}^n$ is the deterministic label space. As its output, the DINN computes for each sample i an estimate of the interval label $\widehat{F}(\mathbf{X}_i)$. In order to do so, the training algorithm iteratively reduces the difference between the true known training label y_i and its prediction label $\widehat{F}(\mathbf{X}_i)$ by minimizing a loss function $\mathcal{L}(\widehat{F}(\mathbf{X}_i), y_i; \mathbf{W})$, such as negative log likelihood, parameterized by interval $\mathbf{W} \in \mathbb{IR}$. Hence, we train the DINN to find the set of interval parameters $\mathbf{W}$ that minimize the loss function. After training, at inference time each future sample i is classified as

$$\widehat{F}(\mathbf{X}_i) = \arg\max_{\mathbf{W}} \Big\{ \texttt{mid}\Big(\sigma(\mathbf{X}_i^{new}\mathbf{W}) \Big) \Big\}, \tag{1}$$

where σ is the `softmax` function as defined on Eq. 2, $\mathbf{W}$ are the trained interval weights of the DINN, and $\mathbf{X}_i^{new}$ are the features of a future query of sensor data in the same tensor dimension as the $\mathbf{X}$ used for training. The softmax function $\sigma(\mathbf{X}_i^{new}\mathbf{W})$ outputs interval probabilities for each class, which in turn give an interval bound around the real-valued softmax probability. We take the midpoint `mid()` of each of these interval class probabilities in order to obtain the predicted class through the arg max function. The softmax function for sample i of class $j \in C$, for a vector $\mathbf{z}$ is

$$\sigma(\mathbf{z})_i^j = \frac{e^{\mathbf{z}_i^j}}{\sum_j e^{\texttt{mid}(\mathbf{z}_i^j)}} \ . \tag{2}$$

For the training data of this application, the features $\mathbf{X}$ are usually the acceleration time-series of the sensor network, strain measurements, and environmental data. The labels are obtained either from domain experts that classify each sample i in the training set, from colocated sensors that can help identify a measure of relative damage (e.g., structural displacements)—which amounts to weakly supervised data (because of the weak labeling signals), or a semi-supervised algorithm where one engineers a way to provide more labels for training given a largely unlabeled dataset. The labels y can be binary: {`Damage, No Damage`} or they can identify the damage: {`Damage Type 1,...,Damage Type C`}. Fig. 1 shows a schematic representation of the classification DINN.

Uncertainty in the data is introduced to the DINN by lower bounds and upper bounds of the features $\mathbf{X}$ for every training sample in the training set.

The ultimate goal of the DINN is to generalize to predictions of unseen uncertain data where labels y_i are not available. The following sections delve into the details of the DINN.

3.2. Fully-Connected DINN

For deep neural networks, the predictive interval function $\widehat{F}(\mathbf{X})$ at the output layer L is composed of the functions of the hidden layers, such that

$$\widehat{F}(\mathbf{X}) = F^{(L)}\Big(F^{(L-1)}\big(F^{(L-2)}\ldots\big(F^{(\ell)}\ldots\big(F(\mathbf{X})\big)\big)\big)\Big) \tag{3}$$

for $\ell = 1 \ldots L$, where ℓ is the layer number.

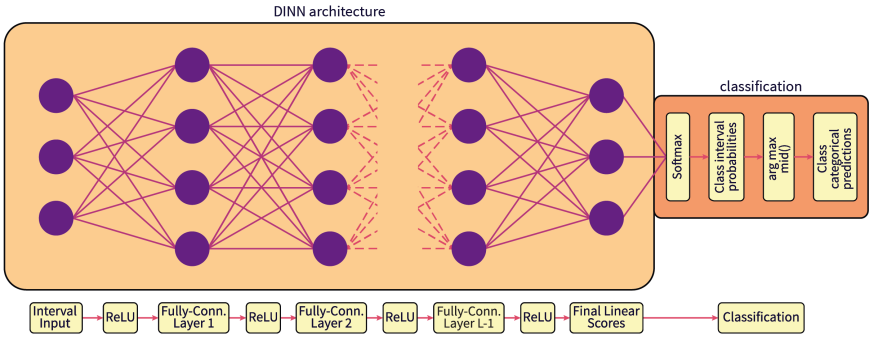

Figure 1. Schematic Representation of Classification DINN.
Nodes represent the activations from previous layers after passing through a `ReLU`
The edges represent the interval weights of the layers.

Hence, the DINN is a composition of interval-valued functions where each of these functions is the output of each of the ℓ layers of the network after passing through a nonlinear activation function. Deep neural networks have more power to express the functions that they attempt to approximate because of the combinatorial advantage of the composition of functions (Goodfellow et al., 2016). A crucial aspect of the development of the DINN is requiring the nonlinear activation function at each layer $F^{(\ell)}$ be monotonic and Lipschitz so that the final output can also be Lipschitz and as sharp as possible. Moreover, the computations of the algorithm to obtain the gradients of the interval functions are ordered in a way to reduce interval dependency by eliminating repeated values of the same variable. With these elements in place, the interval predictions $\widehat{F}(\mathbf{X})$ of the DINN can be reasonably narrow by reducing interval dependency.

3.3. Training

For softmax classification, the corresponding loss at each iteration is computed using the negative log-likelihood of the softmax function, for each mini-batch of size B, as

$$\mathcal{L}(\widehat{F}(\mathbf{X}), y) = \frac{1}{B}\sum_{i}\mathcal{L}_i(\widehat{F}(\mathbf{X}_i), y_i) = -\frac{1}{B}\sum_{i=1}^{B}\log(\mathbf{p}_i^{y_i}) = -\frac{1}{B}\sum_{i=1}^{B}\log\Big(\frac{e^{\mathbf{z}_i^{y_i}}}{\sum_j e^{\mathtt{mid}(\mathbf{z}_i^j)}}\Big), \tag{4}$$

where $\mathbf{p}_i^{y_i}$ is the class softmax interval probability for the target class y_i of sample i, $\mathbf{z}_i$ are the interval linear scores for each sample, $\mathbf{z}_i = \mathbf{X}_i\mathbf{W}$, and the denominator $\sum_j e^{\mathtt{mid}(\mathbf{z}_i^j)}$ is the sum across all classes $j \in C$. Thus, for each sample i we seek to maximize the interval softmax probability of the target class. The loss $\mathcal{L}(\widehat{F}(\mathbf{X}), y)$ it's also called the data loss and it's an interval scalar.

During training, we find a set of optimal parameters $\mathbf{W} \in \mathbb{IR}$ for each layer of the network by minimizing the loss function. In order to minimize the interval loss function $\mathcal{L}(\widehat{F}(\mathbf{X}_i), y_i)$, we use first-order gradient based optimization. In particular, we use mini-batch stochastic gradient descent (SGD) and some of its variants, SGD with momentum (Polyak, 1964) and Adam (Kingma and Ba, 2014) but with an interval modification. The vanilla gradient descent update rule at step $k+1$ for sample i at layer ℓ is defined as

$$\mathbf{W}_{k+1}^{(\ell)} = \mathbf{W}_{k}^{(\ell)} - \alpha_k \nabla_{\mathbf{W}_k^{(\ell)}} \mathcal{L}(\widehat{F}(\mathbf{X}_i), y_i), \tag{5}$$

$$\mathbf{b}_{k+1}^{(\ell)} = \mathbf{b}_{k}^{(\ell)} - \alpha_k \nabla_{\mathbf{b}_k^{(\ell)}} \mathcal{L}(\widehat{F}(\mathbf{X}_i), y_i), \tag{6}$$

where at each step k, $\mathbf{W}_k$ is the interval weight matrix , $\mathbf{b}_k$ is the interval bias vector, α_k is the step size, $\nabla_{\mathbf{W}_k}\mathcal{L}(\cdot)$ and $\nabla_{\mathbf{b}_k}\mathcal{L}(\cdot)$ are the gradients of the loss function with respect to parameters $\mathbf{W}_k$ and $\mathbf{b}_k$ (algorithmic details to obtain the interval gradients are contain in Appendix A). With mini-batch gradient descent, the algorithm chooses uniformly at random a batch of samples of size B from the full training set $\mathcal{T}$ and updates the gradient with Eq. 5 and Eq. 6 or its variants, which trains the network. There are three major steps to training a deep neural network:

1. Compute the loss at each training iteration via forward propagation of the input activations through the network's layers.

2. Compute the gradient of the loss function with respect to the model's parameters via the Backpropagation algorithm.

3. Minimize the loss function via stochastic gradient descent.

Algorithm 1 shows the I-Adam algorithm developed to process interval input and based on the original Adam algorithm (Kingma and Ba, 2014). Both vanilla SGD and SGD with momentum do not require changes to the update rule for interval gradients. However, the Adam optimization algorithm has an adaptive learning rate and implementing it with interval arithmetic causes it to diverge quickly. Thus, we modify the Adam algorithm for interval gradients and call I-Adam. Notice that the algorithm keeps running estimates of the gradients—first moments (mean) and second moments (variance)—in order to perform the weight update. In I-Adam, the second moment biased estimates v_k are modified so that they are real-valued—by squaring only the mid points of the interval gradients in $\texttt{mid}(\mathbf{G}_k)^2$, this modification maintains an adaptive learning rate while reducing divergence.

3.3.1. *Stochastic Gradient Descent with Warm Restarts (SGDR)*
Vanilla SGD and SGD with momentum do not have an adaptive learning rate like Adam. Thus, we use Stochastic Gradient Descent with Warm Restarts (SGDR) developed in (Loshchilov and Hutter, 2016), which follows a cosine decay annealing schedule in order to set the learning rate for every iteration. It is a warm restart because the gradient update restarts from the last iteration, but with a higher learning rate. The learning rate α_k at iteration k, for the i^{th} run of the scheduler is

$$\alpha_k = \alpha^i_{min} + \frac{1}{2}\left(\alpha^i_{max} - \alpha^i_{min}\right)\left(1 + \cos\left(\frac{T_{cur}}{T_i}\right)\right), \tag{7}$$

where $(\alpha^i_{max} - \alpha^i_{min})$ is the range of the learning rate decrease, T_i is the number of iterations performed in a cycle, T_{cur} tracks how many epochs have been performed since the last restart. By using SGDR, we seek to prevent finding suboptimal local minima and to avoid large interval gradients that lead to a blow-up.

Algorithm 1: Adam Algorithm for Interval Input

Input: Step size α
Input: Decay rates $\beta_1, \beta_2 \in [0, 1)$
Input: Initial parameters $\mathbf{W} \in \mathbb{IR}$
Initialize $\mathbf{m}_0 \in \mathbb{IR} = 0$
Initialize $v_0 \in \mathbb{R} = 0$
Initialize $k = 0$
while *stopping criteria not met* **do**
  $k \quad k + 1$
  $\mathbf{G}_k \quad \nabla_{\mathbf{W}} \mathcal{L}_k$ compute interval gradients at step k
  $\mathbf{m}_k \quad \beta_1 \mathbf{m}_{k-1} + (1 - \beta_1)\mathbf{G}_k$ biased interval first moment estimate (interval)
  $v_k \quad \beta_2 v_{k-1} + (1 - \beta_2) \cdot \mathtt{mid}(\mathbf{G}_k)^2$ biased real second moment estimate (real)
  $\widehat{\mathbf{m}}_k \leftarrow \mathbf{m}_k / (1 - \beta_1^k)$
  $\widehat{v}_k \quad v_k / (1 - \beta_2^k)$
  $\mathbf{W}_k \leftarrow \mathbf{W}_{k-1} - \alpha \cdot \widehat{\mathbf{m}}_k / (\sqrt{\widehat{v}_k} + \epsilon)$
return $\mathbf{W}_k$

3.3.2. *Regularization*

Regularization is used for two purposes in the DINN. First, regularization is used to prevent overfitting to the training set, with the goal of achieving better performance in unseen data. It achieves so by shrinking the weight contribution of less relevant features. Second, it stabilizes the solution which for finding interval gradients is of crucial importance. Two forms of regularization are used for the DINN. $L1$ (Lasso) regularization uses the l_1 norm, which is

$$\Omega(\mathbf{W}) = \|\mathbf{W}\|_1 = \max_{1 \le j \le n} \sum_{i=1}^{m} |\mathbf{A}_{ij}|, \tag{8}$$

and $L2$ (Ridge) regularization which is defined as

$$\Omega(\mathbf{W}) = \sum_m \sum_n \mathbf{W}^2_{m,n}. \tag{9}$$

The $L2$ regularization is computed in a way to directly square the elements of the $\mathbf{W}$ in INTLAB, so as to obtain a monotonic expression. The regularization is added to the data loss, as $J(\widehat{F}(\mathbf{X}), y; \mathbf{W}) = \mathcal{L}(\widehat{F}(\mathbf{X}), y; \mathbf{W}) + \lambda\Omega(\mathbf{W})$, where $J(\widehat{F}(\mathbf{X}), y; \mathbf{W})$ is the regularized (total) loss and λ is the regularization strength.

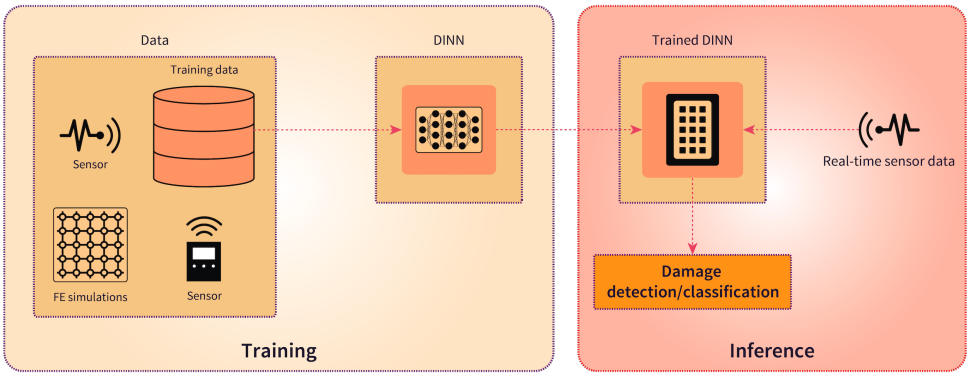

Figure 2. Flowchart for classification DINN for damage detection and classification. The labeled training data consists of sensor measurements and possibly FE simulations. The query data for inference is unlabeled real-time sensor data. The output of the classication DINN for SHM is the detection or classification of the type of structural damage.

3.4. Predictive SHM Tasks Performed with DINN

The DINN is used 'model-free,' i.e., without a structural FE model for the Damage Detection task (see hierarchical tasks in Section 2.1). Nonetheless, the DINN can be used with an interval finite element model (IFEM) for more in-depth damage identification of a particular task or to generate simulation training data. Once the DINN is trained, it can be queried with unlabeled real-time sensor data to detect or classify damage. The overall process is shown on Fig. 2.

4. Experiments

4.1. Iris Dataset

The Iris dataset is a small and simple dataset commonly used to test classifiers, retrieved from the UCI machine learning repository (Dua and Graff, 2017). We thus use this well-benchmarked dataset as a first step to evaluate the skill of the classification DINN to be an efficient discriminator with interval uncertainty in the features. The Iris dataset consists of 150 samples, four features, and three classes. Each class has 50 samples, thus making it a balanced dataset. The features correspond to the sepal and width dimensions, while each class corresponds to a type of iris plant $y = \{\texttt{Setosa, Versicolour, Virginica}\}$.

For the Iris dataset, we conduct separate experiments with increasing uncertainty levels at $\beta = \{2\%, 5\%, 10\%\}$ for all features in $\mathbf{X}$. We use the SGD and the Interval Adam (I-Adam) optimizers. We use a two-layer DINN, with 100 units in the hidden layer, and a maximum of 200 epochs with early stopping. The batch size is 12 samples, while the learning rate is 0.01 and 0.001

for SGD and I-Adam, respectively. A training/validation/test split of 60/10/30 is used for training. The results are summarized on Table I.

Table I. Iris dataset accuracy

Accuracy of two-layer DINN with Iris dataset					
$\beta = 2\%$		$\beta = 5\%$		$\beta = 10\%$	
SGD	I-Adam	SGD	I-Adam	SGD	I-Adam
88.2%	88.9%	85.1%	85.7%	84.5%	84.8%

4.2. Z24 Bridge Dataset

The Z24 bridge dataset has been used as a benchmark to test different SHM damage identification algorithms, both solid mechanics-based and data-driven. It consists of the SHM sensor data from nearly one year of monitoring of a post-tensioned three-span concrete box girder bridge in Switzerland. The bridge was subjected to normal operations for nine months and to artificially induced damage during the last month of operations. In addition to SHM accelerometers, environmental factors were also recorded in parallel. Detailed information about the Z24 bridge SHM program and the different tests is found on (Krämer et al., 1999), (De Roeck et al., 2000).

For the duration of the SHM program on the bridge, different SHM subprograms were conducted on the Z24 bridge with their own goals. The two main SHM subprograms were: (1) a long-term continuous monitoring test for the one-year period before induced damage. This test had the goal of examining the environmental effects on the bridge structural dynamics. (2) short-term progressive damage tests (PDTs) under different damage scenarios for a one-month period prior to the demolition of the bridge. This subprogram had the goal of examining the bridge dynamics under fifteen different damage conditions and two reference undamaged conditions.

For subprogram (2), the bridge contained nine sensor network set-ups, each of which recorded the vibrations associated with the progressive damage tests (PDT). Each set-up has 33 accelerometers. Fig. 3 shows a side view and elevation view from the bridge, based on (Krämer et al., 1999). Refer to sensor locations on (Krämer et al., 1999) and (De Roeck et al., 2000).

4.2.1. *Damage Detection Experiments*

The first task performed with the classification DINN is damage detection (refer to Sec. 2.1). In order to detect damage, we take the sensors readings from `set-up 1` on the bridge deck for the ambient vibrations. This gives a total of 21 accelerometer readings in the vertical, longitudinal, and transverse directions.

The experiments consist of the followings steps:

1. Perform interval uncertainty quantification on the dataset and convert real-valued data to interval-valued data based on the determined uncertainty levels. This step produces the interval training data.

2. Train the classification DINN with the interval training data.

3. Perform inference with the trained classification DINN by querying it with new interval data to obtain the corresponding class predictions.

4. Obtain corresponding interval softmax probability for the predicted class.

4.2.1.1. *Interval Uncertainty* Sensor uncertainty is likely to be present in an SHM system due to multiple sources, as explained in Sec 2.2. By performing these experiments, we seek to assess the classification performance of the DINN under interval uncertainty. We can represent this uncertainty in the sensors via intervals and then propagate it through the DINN in order to predict the class of each sample. The DINN is very flexible as to how the interval input is defined. Intervals can be defined for subsets of samples and features, and a different interval uncertainty can be defined for each sample separately. A scenario could be one in which during some environmental event (e.g., snow, high winds, abnormal traffic), a subset of sensors readings might become corrupted. In order to quantify the uncertainty, the training or query data can be given an interval according to some expert belief. Such interval can be given manually by the SHM operators, by a rules-based system, or even by a separate machine learning model. Furthermore, one can see that the DINN can be beneficial in cases of missing data at some time intervals for some sensors but we decide to exclude the missing data scenario from the experiments and leave it to a future study.

For the Z24 bridge, we use an uncertainty level of $\beta = 10\%$ for sensor `100V` and $\beta = 1\%$ for all other sensors in $\mathbf{X}$ (refer to (Krämer et al., 1999) for sensor locations). Then, we perform another experiment with the same uncertainty level for the same sensor and an uncertainty level of $\beta = 5\%$ for all other sensors in $\mathbf{X}$.

4.2.1.2. *Classification DINN Model Details* We trained the classification DINN with SGD with momentum *SGD_mom* and I-Adam. We use two hidden layers, 500 units per layer, and 300 epochs. The batch size is 2048 samples. For time series data, small batches are sometimes not too informative and this proves to be the case here. We normalize all the features by the l_2 vector norm of each feature. We use a random weight initialization based on a normal distribution with zero mean and unit variance and a weight scale of 0.05. To evaluate the skill of the model, we use a training/validation/test split of 60/10/30.

For SGD with momentum we use a cosine annealing learning rate schedule with warm restarts (SGDR) with an initial learning rate of 0.001 and 5 initial steps, while doubling the number of annealing steps after every cycle. Using the cosine annealing scheduler with warm restarts proved to be highly beneficial when compared with a linearly decaying learning rate without restarts. For I-Adam, we use a learning rate of 0.00005 and linearly decay it by 0.99 after every iteration.

4.2.2. *Results*

Table II shows the comparison of test set results of the trained classification DINN trained with SGDR and Interval-Adam (I-Adam). It is shown that higher accuracy is achieved with SGDR than with I-Adam, and the difference is more significant at higher levels of uncertainty (90.6% vs. 72.2%). Fig. 4 shows the training loss of SGD with momentum with linear learning rate decay (*SGD_mom*),

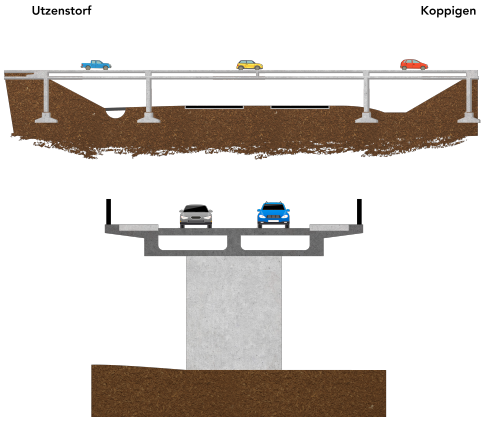

Figure 3. Former Z24 Bridge side and elevation views adapted from (Krämer et al., 1999)
Top: Side view
Bottom: Pier elevation view

SGDR, and I-Adam. On Fig. 4, we can see that SGDR has a flatter loss curve than I-Adam and that *SGD_mom*. Despite the flatter loss curve, SGDR has a narrower loss and outperforms I-Adam in the test set by a wide margin as shown on Table II. One reason for the inferior performance of I-Adam in this dataset could be that the adaptive learning rate of Adam is overly ambitious and goes into interval gradient regions where it gets stuck in local minima. This observation gave us insight into first-order optimization for interval gradients and led us to experiment with SGDR. As its real-valued counterparts, intervals need an efficient learning rate but also benefit greatly from restarts to escape both local minima and avoid exploding interval gradients.

Table II. Z24 test set results

Accuracy			
{ 1%, 10%}		{5%, 10%}	
SGDR	I-Adam	SGDR	I-Adam
95.8%	85.3%	90.6%	72.2%

5. Conclusion and Future Work

We presented a novel interval deep learning algorithm which can be used for real-time damage detection tasks under sensor uncertainty. The algorithm, the Deep Interval Neural Network (DINN) classifier, can be used to detect damage in a structure in real-time, using sensor data alone and subject to uncertainty. The algorithm is in a supervised setting, which implies that both normal and abnormal (structural damage) samples are needed to train the algorithm. We tested the algorithm with the Z24 bridge benchmark and reached an accuracy of 95.8% using an interval uncertainty level of $\beta = 10\%$ for one sensor and $\beta = 1\%$ for all others in the feature set.

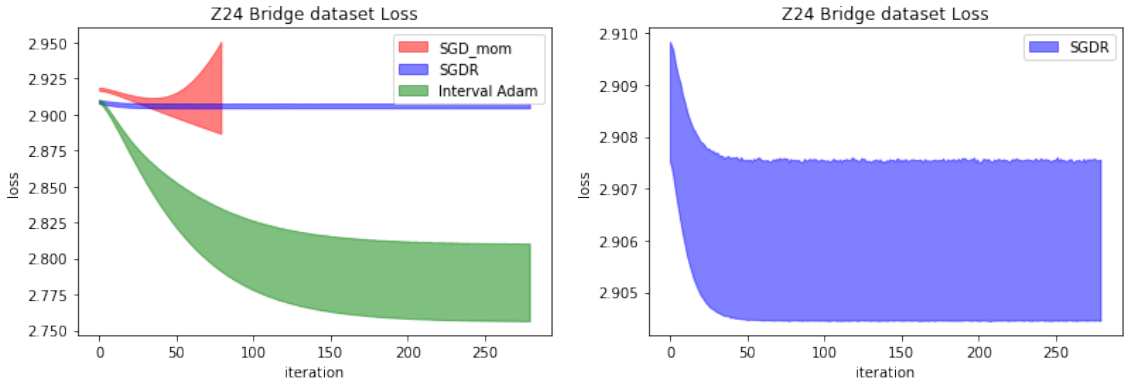

Figure 4. Training loss history comparison between SGD with momentum with only linear learning rate decay, SGD with momentum with cosine decay and restarts (SGDR), and I-Adam (left); Training loss history of SGD with momentum with cosine decay and restarts (SGDR). SGDR is the best performer for the held-out dataset. SGD with momentum with only linear learning rate decay blows up after about 80 iterations. I-Adam gives a smooth loss but the interval is too wide to make correct predictions.

While samples of structural damage were included in this dataset, it is not the case for most SHM system. Nonetheless, one of the advantages of deep learning techniques is that damage samples can be obtained from other SHM systems (either real or from laboratories) to pretrain the DINN and fine-tuned with simulations from FE models. Future promising research directions for this work include weakly or semi-supervised models and more complex SHM tasks, such as damage localization.

Appendix

A. Classification DINN Algorithmic Details

A.1. Interval Forward Propagation

Algorithm 2 of this appendix computes the predictions $\widehat{F}(\mathbf{X})$ and the loss $\mathcal{L}(\widehat{F}(\mathbf{X}_i), y)$ of the DINN through forward-propagation of each mini-batch of data. The layer output $\mathbf{z}^{(\ell)}$ (scores) is computed through an affine transformation as $\mathbf{z}^{(\ell)} = \mathbf{h}^{(\ell-1)}\mathbf{W}^{(\ell)} + \mathbf{b}^{(\ell)}$, at each layer ℓ; $\mathbf{W}^{(\ell)} \in \mathbb{IR}^{H^{(\ell-1)} \times H^{(\ell)}}$, $\mathbf{h}^{(\ell-1)} \in \mathbb{IR}^{B \times H^{(\ell-1)}}$, $\mathbf{b}^{(\ell)} \in \mathbb{IR}^{H^{(\ell)}}$, and $\mathbf{z}^{(\ell)} \in \mathbb{IR}^{B \times H^{(\ell)}}$, $H^{(\ell-1)}$ is the dimension of layer $\ell - 1$, $H^{(\ell)}$ is the dimension of layer ℓ, and B is the number of samples in each mini-batch of data. Then, the layer output $\mathbf{z}^{(\ell)}$ passes through an element-wise activation function $F(\mathbf{z}^{(\ell)})$. The default activation function for the DINN is a `ReLU`.

In the classification setting, for the final layer L, the prediction is the output of the softmax function $F(\mathbf{z}^{(L)}) = \texttt{softmax}(\mathbf{z}^{(L)})$, where $\mathbf{z}^{(L)} \in \mathbb{IR}^{B \times C}$ for B mini-batch samples in C classes. Finally, the loss at the end of each epoch is computed using negative log likelihood for each mini-batch of size B, as

$$\mathcal{L}(\widehat{F}(\mathbf{X}), y) = \frac{1}{B}\sum_{i} \mathcal{L}_i(\widehat{F}(\mathbf{X}_i), y_i) = -\frac{1}{B}\sum_{i=1}^{B} \log(\mathbf{p}_i^{y_i}) = -\frac{1}{B}\sum_{i=1}^{B} \log\Big(\frac{e^{\mathbf{z}_i^{y_i}}}{\sum_j e^{\texttt{mid}(\mathbf{z}_i^j)}}\Big). \tag{10}$$

The loss is an interval scalar. Notice that `ReLU`, `softmax`, and their derivatives are monotonic functions, which allows for calculation of sharp interval enclosures. The `ReLU` is the standard layer activation function for the DINN but it can be replaced by a different activation function with ease.

Algorithm 2: Forward Propagation Algorithm for Each Mini-batch of Data

Obtain the input raw real-valued features X and labels y
Embed X into interval input features $\mathbf{X}$ considering lower $\underline{X}$ and upper bounds $\overline{X}$ for each sample
Choose number of layers L
Require model parameters $\mathbf{W}^{(\ell)}$, for each layer ℓ
$\mathbf{h}^{(0)} = \mathbf{X}$
for $\ell = 1 : L$ **do**
  $\mathbf{z}^{(\ell)} = \mathbf{h}^{(\ell-1)}\mathbf{W}^{(\ell)} + \mathbf{b}^{(\ell)}$
  $\mathbf{h}^{(\ell)} = F(\mathbf{z}^{(\ell)})$
Prediction $\widehat{F}(\mathbf{X}) = \mathbf{h}^{(L)}$
Compute loss $\mathcal{L}(\widehat{F}(\mathbf{X}_i), y_i)$

A.2. Interval Backpropagation

The first step for SGD is to find the gradients of the loss with respect to the weights, i.e., $\nabla_{\mathbf{W}}\mathcal{L}$. These gradients are computed via automatic differentiation in the computational graph of the neural network. For the DINN, we use the Backpropagation algorithm to take derivatives of interval matrices. In particular, the chain rule is used to compute the gradient as follows

$$\nabla_{\mathbf{W}^{(\ell)}}\mathcal{L} = \frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(\ell)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(\ell)}}\frac{\partial \mathbf{z}^{(\ell)}}{\partial \mathbf{W}^{(\ell)}} \tag{11}$$

where ℓ is a layer, $\frac{\partial \mathbf{z}^{(\ell)}}{\partial \mathbf{W}^{(\ell)}}$ is the partial derivative of the score with respect to the weights, and $\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(\ell)}}$ is the partial derivative of the loss with respect to the score at layer ℓ.

This procedure is summarized in Algorithm 3 of this appendix for general loss and activation functions. In the algorithm, the $\boldsymbol{\delta}$'s denote error terms that recursively update the error in the network at each layer. The interval gradients are accumulated in the $\mathbf{G}$ variable, a data structure containing the interval tensors of gradients. Regularization $\lambda \cdot \Omega(\mathbf{W})$ is also added to the gradient computation as shown on Algorithm 3 of this appendix. In the implementation of this algorithms, care must be taken not to repeat interval variables in order to avoid interval dependency.

Algorithm 3: Backpropagation Algorithm for Interval Loss for Each Mini-batch of Data

Run forward computation algorithm to obtain layer activations $\mathbf{h}^{(\ell)}$
Compute gradient of loss at output layer $\boldsymbol{\delta}_{out}^{(L)}$
$\mathbf{G} = \boldsymbol{\delta}_{out}^{(L)}$
$\nabla_{\mathbf{W}^{(L)}}\mathcal{L} = \mathbf{h}^{(L-1)^T}\mathbf{G} + \lambda\nabla_{\mathbf{W}^{(L)}}\Omega(\mathbf{W})$
$\nabla_{\mathbf{b}^{(L)}}\mathcal{L} = \mathbf{G} + \lambda\nabla_{\mathbf{b}^{(L)}}\Omega(\mathbf{W})$
$\mathbf{G} \leftarrow \mathbf{G}\mathbf{W}^{(L)^T}$
for $\ell = L-1, \ldots, 1$ *layers* **do**
  $\mathbf{G} \leftarrow \boldsymbol{\delta}_{\mathbf{z}}^{(\ell)} = \mathbf{G} \odot F'(\mathbf{z}^{(\ell)})$
  $\nabla_{\mathbf{W}^{(\ell)}}\mathcal{L} = \mathbf{h}^{(\ell-1)^T}\mathbf{G} + \lambda\nabla_{\mathbf{W}^{(\ell)}}\Omega(\mathbf{W})$
  $\nabla_{\mathbf{b}^{(\ell)}}\mathcal{L} = \mathbf{G} + \lambda\nabla_{\mathbf{b}^{(\ell)}}\Omega(\mathbf{W})$
  $\mathbf{G} \leftarrow \mathbf{G}\mathbf{W}^{(\ell)^T}$

For the classification DINN as presented on this paper, with cross-entropy loss (negative log likelihood) and `ReLU` activations, the partial derivatives in Algorithm 3 specialize as follows. At the final layer L, for each class $j \in C$, we have

$$\boldsymbol{\delta}_{out}^{(L)} = \boldsymbol{\delta}_{\mathbf{z}}^{(L)} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(L)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{h}^{(L)}}\frac{\partial \mathbf{h}^{(L)}}{\partial \mathbf{z}^{(L)}} = (\mathbf{p}_i^j - \mathbb{1}\{y_i = j\}), \tag{12}$$

where $\mathbb{1}\{\cdot\}$ is the indicator function. For $\frac{\partial \mathbf{z}^{(\ell)}}{\partial \mathbf{W}^{(\ell)}}$, we have

$$\frac{\partial \mathbf{z}^{(\ell)}}{\partial \mathbf{W}^{(\ell)}} = \mathbf{h}^{(\ell-1)^T}. \tag{13}$$

At the hidden layers ($\ell = L-1, L-2, \ldots, 1$), going backwards in the network graph, the partial derivative with respect to the score is

$$\begin{aligned}\boldsymbol{\delta}_{\mathbf{z}}^{(\ell)} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(\ell)}} &= \boldsymbol{\delta}_{\mathbf{z}}^{(\ell+1)}\mathbf{W}^{(\ell+1)^T} \odot \underbrace{F'(\mathbf{z}^{(\ell)})}_{\substack{\text{derivative of activation}}} \\ &= \boldsymbol{\delta}_{\mathbf{z}}^{(\ell+1)}\mathbf{W}^{(\ell+1)^T} \odot \underbrace{\mathbb{1}\{\mathbf{h}^{(\ell)} \geq 0\}}_{\substack{\text{derivative of } \texttt{ReLU} \\ = 1 \text{ if } \mathbf{h}^{(\ell)} \geq 0 \\ = 0 \text{ otherwise.}}}.\end{aligned} \tag{14}$$

where $\mathbf{A} \odot \mathbf{B}$ is the Hadamard (element-wise) product of $\mathbf{A}$ and $\mathbf{B}$.

Finally, the gradient with respect to the weights at each layer is

$$\nabla_{\mathbf{W}^{(\ell)}}\mathcal{L} = \mathbf{h}^{(\ell-1)^T}\boldsymbol{\delta}_{\mathbf{z}}^{(\ell)}, \tag{15}$$

and the gradient with respect to the biases at each layer is

$$\nabla_{\mathbf{b}^{(\ell)}}\mathcal{L} = \boldsymbol{\delta}_{\mathbf{z}}^{(\ell)}. \tag{16}$$

References

De Roeck, G., Peeters, B., and Maeck, J. (2000). Dynamic monitoring of civil engineering structures. *Computational Methods for Shell and Spatial Structures*.

Dua, D. and Graff, C. (2017). UCI machine learning repository.

Figueiredo, E., Park, G., Farrar, C. R., Worden, K., and Figueiras, J. (2011). Machine learning algorithms for damage detection under operational and environmental variability. *Structural Health Monitoring*, 10(6):559–572.

Figueiredo, E., Park, G., Figueiras, J., Farrar, C., and Worden, K. (2009). Structural health monitoring algorithm comparisons using standard data sets. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States).

Figueiredo, E. and Santos, A. (2018). Machine learning algorithms for damage detection. *Vibration-Based Techniques for Damage Detection and Localization in Engineering Structures*, pages 1–39.

Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y. (2016). *Deep learning*, volume 1. MIT press Cambridge.

Kaya, Y. and Safak, E. (2019). Structural health monitoring: Real-time data analysis and damage detection. In *Seismic Structural Health Monitoring*, pages 171–197. Springer.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Krämer, C., De Smet, C., and De Roeck, G. (1999). Z24 bridge damage detection tests. In *IMAC 17, the International Modal Analysis Conference*, volume 3727, pages 1023–1029. Society of Photo-optical Instrumentation Engineers.

Kreinovich, V., Lakeyev, A. V., Rohn, J., and Kahl, P. (2013). *Computational complexity and feasibility of data processing and interval computations*, volume 10. Springer Science & Business Media.

Loshchilov, I. and Hutter, F. (2016). Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*.

Moore, R. E., Kearfott, R. B., and Cloud, M. J. (2009). *Introduction to interval analysis*, volume 110. Siam.

Neumaier, A. (1990). *Interval methods for systems of equations*, volume 37. Cambridge university press.

Neves, A., Gonzalez, I., Leander, J., and Karoumi, R. (2017). Structural health monitoring of bridges: a model-free ann-based approach to damage detection. *Journal of Civil Structural Health Monitoring*, 7(5):689–702.

Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17.

Sarkar, S., Reddy, K. K., Giering, M., and Gurvich, M. R. (2016). Deep learning for structural health monitoring: A damage characterization application. In *Annual conference of the prognostics and health management society*, pages 176–182.

Sun, L., Dbouk, H., Neumann, I., Schön, S., and Kreinovich, V. (2017). Taking into account interval (and fuzzy) uncertainty can lead to more adequate statistical estimates. In *North American Fuzzy Information Processing Society Annual Conference*, pages 371–381. Springer.

Wang, R., Li, L., and Li, J. (2018). A novel parallel auto-encoder framework for multi-scale data in civil structural health monitoring. *Algorithms*, 11(8):112.