Learning Damage Event Discriminator Functions with Distributed Multi-instance RNN/LSTM Machine Learning - Mastering the Challenge

Stefan Bosse

University of Bremen, Dept. Computer Science, Bremen, Germany

Abstract. Common Structural Health Monitoring systems are used to detect past damages occurred in structures with sensor networks and external sensor data processing. The time of the damage creation event is commonly unknown. Numerical methods and Machine Learning are used to extract relevant damage information from sensor signals that is characterised by a high data volume and dimension. In this work, distributed multi-instance learning applied to raw time-series of sensor data is deployed to predict the event of the occurrence of a hidden damage in a mechanical structure using typical vibrations of the structure. The sensor processing and learning is performed locally on sensor node level with a global fusion of prediction results to estimate the damage location and the time of the damage creation. Recurrent neural networks with a long-short-term memory architecture are considered implementing a damage discriminator function. The sensor data required for the evaluation of the proposed approach is generated by a multi-body physics simulation approximating material properties.

Keywords. Distributed Machine Learning; Recurrent Neural Networks, Long-short-term Memory, Distributed Sensor Networks; Time-series Forecasting

1. Introduction

Detecting damages and changes in technical structures and materials affecting the mechanical and operational capabilities is eminent for the safe deployment of a wide range of machines, devices, and daily life products under varying environmental conditions. Damage detection can be performed at run-time (on-line non-destructive testing), before the deployment, or at periodic intervals off-line. There are different damage classes that can be monitored:

1. Visible damages (with or without in-depth propagation) and damages on the surface (e.g., scratches);
2. Hidden damages inside the material without any visibility outside the material caused by external forces;
3. Hidden damages inside the material without any visibility outside the material caused by internal forces;
4. Altering of materials and material properties (continuous, inside or on the surface);

The detection of hidden damages is the most difficult damage class to be handled by automatic damage detection systems, especially concerning an unknown response of the measuring system and anisotropic material properties. Moreover, impact events from the environment acting on hybrid materials and composite structures (e.g., fiber-metal laminates) typically create hidden damages without any visibility at the surface, although the damage was caused by external forces.

There are at least four different levels of information that can be delivered by a Structural Health Monitoring (SHM) systems [1]:

1. Detection of damages and material changes;
2. Localization of damage;
3. Assessment of damages and impact on operational safety;
4. Prediction of mechanical and operation behaviour.

Damage diagnostic measuring systems commonly use guided waves (e.g., [2] and [1]) to detect

changes (e.g., damages like material inhomogeneity, delaminations of layers, deformation) of hybrid and composite materials. Vibration-based diagnostics are frequently used in SHM systems [3]. There are active and passive measuring techniques, generating vibrations actively or using already existing operational vibrations to detect changes of the structures.

Machine learning approaches are used for the data analysis in SHM if there is no known or an incomplete damage-sensor response function $f_{\text{damage}}(s)$: $s \rightarrow d$, typically resulting from unknown or only partially known material behaviour models. In [2] and [1], Artificial Neural Networks (ANN) are deployed in SHM[4]. They are applied, e.g., to ultrasonic data measured by an externally applied sensor-actuator system or by analysing vibration signatures.

In this work, time-resolved data from strain-gauge sensors applied to the surface of a device under test (DUT) is used to detect hidden damages at run-time and on-line using time-series prediction methods and recurrent ANNs (RNN, sub-class of generic ANNs with state memory cells). In earlier work [5], snapshots of data from strain-gauge sensors of a distributed sensor network were used successfully to predict hidden damage locations (discretized) using different ML approaches and a typical vibration situation of the DUT, even under distortions. In [5] the distributed multi-instance learning approach was introduced, but the damages were static, i.e., they exist already before the typical vibration and DUT oscillation was measured and evaluated. In this work, dynamic changes (i.e., damage events) of the DUT are investigated in real-time an on-line, i.e., the damage event occurs during the measuring. The goal of this work is the temporal and spatial estimation of a hidden damage not visible outside by using state-based ANNs applied to time-series of raw strain-gauge sensor data and performing a vibration signature analysis. The ANN should implement a time-dependent discriminator function that amplifies a damage event and damps the input signal otherwise. Ideally, the response output is a squared impulse if a damage occurred in the neighbouring of the sensor node:

$$f_{dam}(s,t) = \begin{cases} 1, & feat_{dam}(s, t > t_0) \wedge \\ & t \in [t_0, t_0 + \Delta t] \\ 0, & otherwise \end{cases} \quad (1)$$

A distributed sensor network is assumed consisting of sensor nodes with two perpendicular orientated strain-gauge sensors applied to the DUT surface. It is assumed that a single sensor node can detect a hidden damage nearby by analysing the time-resolved sensor data of a typical vibrations of the structure. But is also assumed that there are multiple nodes able to detect a damage event (location and time) with a certain probability and that some nodes may fail to detect the damage event (for any reason). Distributed sensor fusion is used finally to get a global damage state information based on majority voting and mass of centre computation for the estimation of the location.

The following sections introduce time-series learning and prediction using recurrent neural networks, the special features required for learning discriminator functions, and distributed multi-instance learning.

Finally, an extensive simulation study is used to evaluate the proposed approach and poses lessons learned using state-based ANNs for damage diagnostics.

2. Learning of Time Series and Prediction of Events

In this work time-series forecasting should be adapted and extended to labelled time-series forecasting. Originally, time-series forecasting is used to predict a variable $x(t)$ at a given sample point t for future values $x(t+1)$,$x(t+2)$,..,$x(t+m)$. In this work, time-series sensor data is used to predict an associated labelled value $y(x)$ correlated with the sensor input data $x(t)$, $x(t-1)$, .., $x(t-n)$. The associated value is here a binary indicator for the occurrence of an damage event (0/1).

Originally statistical Autoregressive and Moving Average (ARMA) and integrating (ARIMA) models were used to predict future evolving of uni- and multi-variate variables. In the last decades, the de-

ployment of feed-forward and recurrent artificial neural networks gain attraction for time-series forecasting.

2.1 Discriminator Function and Learning

The main goal of the training of the neural networks is synthesis of a discriminator function that amplifies damage events and damps non-damage events. I.e., the time-resolved sensor data, i.e., the time-series S of sensor samples $s(t)$, excitated by an oscillation of the structure under test (e.g., measured with strain-gauge sensors) is the input of the discriminator function, and a damage event estimation value in the range [0,1] is the output considering the past n samples:

$$f(S,t,n) = \{s_t, s_{t-1}, ..s_{t-n}\} \rightarrow [0,1] \quad (2)$$

An example of a damage discriminator function output is shown in Fig. 1.

2.2 Time-series and Window Slicing

Typically on-line time-series training (and partially prediction) is performed using window slices of the signal records fed into the ANN. There are two classes of windows: 1. Input data windows; 2. Output data windows (used for training only). A moving window mask function is applied to the input data, in principle shown in Fig. 1 with two typical signal records used in this work (strain-gauge sensor signals). Among sharp shaped masks (rectangular mask window) smooth masks can be applied (e.g., gaussian or hamming windows with smooth edge transitions). Although, time-series prediction using recurrent neural networks is performed commonly using input data windows of small width (< 100 samples), the machine models in this work are trained with large width windows (or applying the entire time record). In contrast to other state-free machine learning techniques like decision tree learner, time-series prediction uses the concept of state to consider past input data. Therefore, even with large time records, each sample of a time record is applied sequentially to the network (comparable to a 1-sample window).

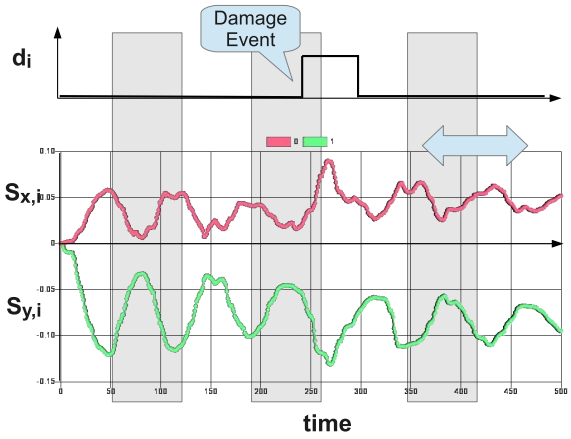


Figure 1. Time-resolved sensor input signals (bottom) and output feature (damage event) fragmented by dynamically moving or static partitioned windows

2.3 Recurrent Neural Networks (Long-short-term Memory)

In contrast to classical feed-forward ANNs, a recurrent ANN (RNN) contains a feedback of the output to the input nodes to learn and predict sequences of input values. A typical RNN architecture is shown in Fig. 2, basically a NN state-machine. Although, in principle it can be used to learn sequences and predict any future development of a sequence, RNN suffer from two problems during training: vanishing gradient and exploding gradient (i.e., error minimisation of network with respect to training data), which make it unusable.

An improvement over simple RNNs are Long-short-term Memory RNNs (LSTM) [6] that introduce a more complex memory cell architecture, shown in Fig. 2. In contrast to LSTM with internal memory cells, feed-forward networks (FFN) can be also used for short-memory applications by saving past data samples externally and passing them to input nodes of the FFN [7]. A deeper discussion of the operation of LSTM-RNNs can be found in [6].

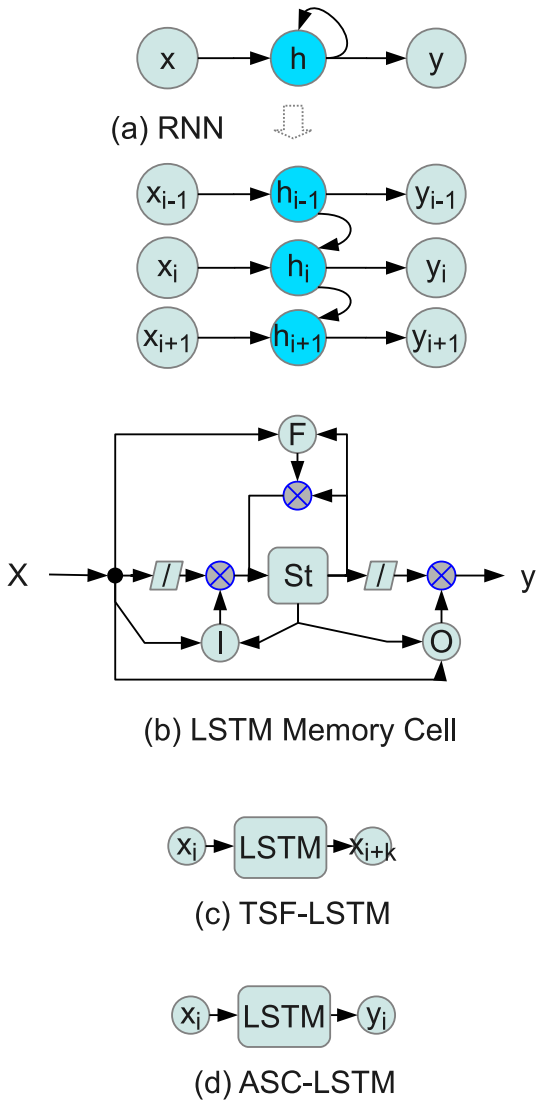


Figure 2. (Top) Recurrent Neural Networks and LSTM memory cell architecture (Bottom) Traidtional TS forecasting LSTM versa Associated LSTM

In contrast to typical use of LSTM-RNNs predicting future outcome of the input variable $x(t+k)$, in this work a modified version is used to predict an associated target variable $y(t)$.

The LSTM-RNN uses a central internal cell state *St*, a forget gate *F*, and an input- and output gate *I* and *O*, respectively (circle nodes). The slanted nodes are transition functions (e.g., using a sigmoid function), the cross nodes are multiplier units. The LSTM can be considered as a sequence of short-term memories. Layers of the LSTM-RNN that get a small gradient update stops learning, commonly the earlier layers, and the LSTM forgets what it has seen in longer sequences, having a short-term memory.

Gers & Schmidhuber [8] introduced a variant of the LSTM (used in this work) providing a "constant error carousels" (CEC) facing training issues mentioned previously and direct access of the gates the internal CEC-states.

In the evaluation section it will be shown that LSTMs are still instable to train and to find an optimal hypothesis function for the discrimination task is a challenge (and an iterative procedure).

The recurrent neural network is typically trained with ordered batches of training samples at once (although, the learner internally processes the training data iteratively and strictly ordered).

2.4 Hypothesis Constraints

To test and evaluate a learned hypothesis function, the training samples (set of full time records with all damage cases and all repetition experiments) are used to calculate a the discriminator coverage by a mask function.

The hypothesis function represented by the trained LSTM-ANN must satisfy a set of constraints:

1. The output of the discriminator function (a trained hypothesis function) must be zero if there is no damage event
2. The output must switch from 0 to 1 within a time interval $[t_0,t_1]$ if a damage event occurred. The discriminator function may detect further damage events after the first one.
3. The output may only change if there was a damage within a bounded spatial area around the sensor position.

The constraints are scored by a scoring function described in Sec. 2.5

2.5 Scoring

Typically training of a machine model incorporates a training algorithm associated to the specific model (e.g., decision trees or neural networks). The trainer fits the model to the training data $\boldsymbol{y}$ to minimize the overall error *min error*=$|\boldsymbol{y}$-$h(\boldsymbol{x})|$ of the trained hypothesis function h represented by the machine model. But the local error minimization can result in hypothesis functions with low overall prediction quality regarding the desired deployment.

To evaluate trained machine models during training iterations, an external scoring function has to be applied using the original training data. the scoring function $Sc(y,h,x)$ returns three parameters s_0, s_1, and s_2. The s score parameters count true-positive and false-positive sample prediction of the trained discriminator function within given time intervals.

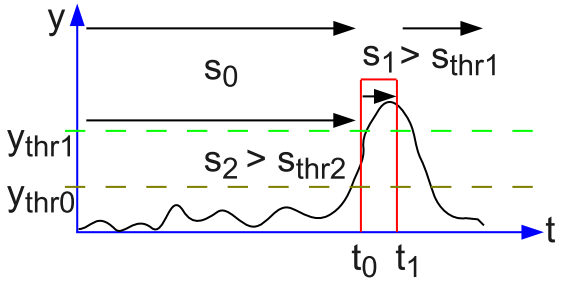


Figure 3. Scoring of a time-series prediction by a pre-trained LSTM by comparing the training output $y(t)$ with the prediction of the ANN using the time series $x(t)$ (red curve: training output variable)

The s_0 parameter scores the output of the trained discriminator function in the time windows $[0,t_0]$ and $[t_1,t_{end}]$ (no damage). The parameter returns the value 1 if the predicted output is always below the threshold y_{thr0} within the interval. True-negative values reduce the value towards zero. The parameter s_2 is only applied to the time window $[0,t_0]$, and is used in this work since the damage event discriminator function can fire after the first event again. The s_1 value should be one for successful training. Finally, the parameter s_2 scores the damage event, i.e., counts the time samples that produce an output of the discriminator function above the threshold y_{thr1}. Since the training output pulse width and starting time are chosen arbitrary, this score value must at least greater zero.

2.6 Training Algorithm

The used LSTM-RNN architecture and trainer uses a gradient error minimization algorithm to modify weights of the neural network. As discussed in Sec. 5, this gradient approach applied to the simulated sensor data and the given training output mask function was not suitable. It poses no monotonic convergence and required an higher-level training outcome scoring with the capability to reinitialize the LSTM with new randomly chosen network weights and to adapt the learning rate dynamically. The iterative training trying to find a hypothesis function of the discriminator satisfying the given constraints is shown in Alg. 1.

The training was performed with one prominent sample (damage case with nearest damage location to the node at position x,y) and randomly chosen data from one of the ten experiments performed under similar conditions (but using Monte Carlo methods to introduce variance in the data sets).

After *trainings* iterations the entire LSTM is reset and a new learning sequence is started. If there is a scoring set found not satisfying all constrains but seems to be a starting point for convergence the learning rate will be decreased to refine the model.

```
 1: function train(
 2:   x,y,
 3:   sam,
 4:   par):
 5: for 1 to trials do
 6:   ML ← new LSTM(par)
 7:   for 1 to trainings do
 8:     exp ← random ∈ {0,1,..,9}
 9:     ds ← {
10:       data100[sam][exp]
11:       data100[0][exp]
12:       data250[sam][exp]
13:       data250[0][exp]
```

```
14:      }
15:      ∀ data ∈ ds do
16:        ML.train(data,par);
17:      ∀ data ∈ ds do
18:        sc[data] ←
19:          score(ML.test(data),data);
20:      if sc[data100[sam]].s1>s1Thr ∧
21:         sc[data100[sam]].s2>s2Thr ∧
22:         sc[data250[sam]].s1>s1Thr ∧
23:         sc[data250[sam]].s2>s2Thr ∧
24:         sc[data100[0]].s2>s2Thr ∧
25:         sc[data100[0]].s2>s2Thr
26:      then
27:         found solution return ML;
28:      if sc[data100[sam]].s2>0.9*s2Thr ∨
29:          sc[data250[sam]].s2>0.9*s2Thr
30:      then
31:          decrease learning rate;
32:   done
33: done
```

Alg. 1. Training algorithm for one localized discriminator function (at node (x,y))

The desired output (damage event prediction) of the discriminator function is trained with a rectangular pulse function. Using smooth pulse shapes showed no improvement with respect to the learning progress and outcome. The starting time t_0 is set to the simulation step where the damage was introduced (but showing a delayed effect). The width of the pulse was chosen arbitrarily but with respect to the overall learning outcome observed in multiple preliminary experiments. The pulse function was used by the scoring, too.

2.7 Distributed Multi-Instance Learning

Each local trained discriminator function is capable to detect a damage event within the (ideally circular) neighbourhood of radius r with a probability p. Neighbouring nodes can detect the same damage event simultaneously enabling damage localization by computing the mass of centre point. The principle distributed sensor fusion is shown in Fig. 4.

Additionally, the detection probability increases by fusion of multiple prediction instances. False-positive detections result in a distortion of the damage position localization. Furthermore, false-positive results can be rejected by cluster analysis.

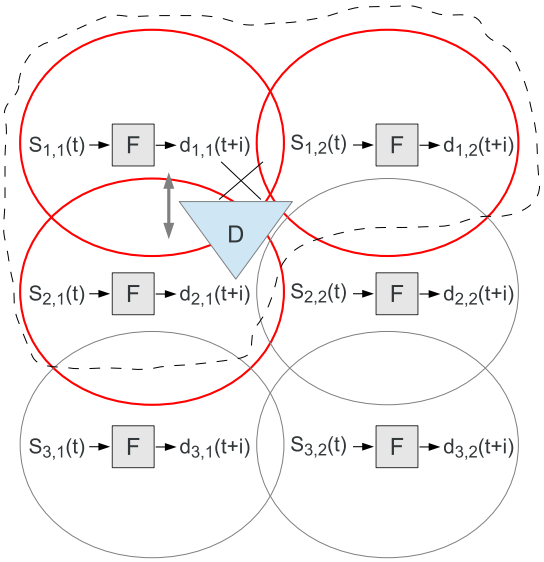


Figure 4. Distributed Multi-Instance Prediction (D: Damage, s: Sensors, d: Damage Discriminator Function, cross: estimated position of damage)

2.8 Model Fitting and Permutation

Distributed multi-instance learning is effected by bias of local environment constraints. I.e., a local learning instance is only trained with its local data. To derive more general models, a pre-trained network can be exchanged by sensor nodes in the network to improve bias-free or bias-reduced models or used as a starting point for learning (instead of random initial values for the weights). Although model permutation is an eminent concept for learning of generalised discriminator functions, it is not known in advance if this improves or degrades the local models and has to be tested on case basis. The experimental evaluation will show no improvement (in contrast, a degradation is observed).

With respect to ANN the permutation can also be applied to weights of neural node edges and nodes itself including the change of the model structure (removing or adding nodes and edges).

3. Modelling of Mechanical Structure and Damages (Fatigue)

Getting time-resolved experimental sensor data from structures with a wide variety of hidden damage cases is a challenge. To evaluate the proposed multi-instance learning approach, synthetic sensor data was computed by physical simulation using a simplified multi-body physics model (MBP) consisting of meshed mass-spring systems. A MBP simulation is faster than commonly used FEM simulation by several orders of magnitude.

3.1 Structural Model

A mechanical structure is modelled with a Graph *St*=<**M**,**Sp**> ([5]), where **M** is a set of mass nodes with a specific mass m_i, and **Sp** is a set of spring-damper edges connecting mass nodes.

$$
\begin{aligned}
St(P) &= \langle \mathbf{M}(\mathbf{P}), \mathbf{Sp}(\mathbf{P}) \rangle \\
\mathbf{M} &= \begin{bmatrix} m_1 & \cdots & m_j \\ \vdots & \ddots & \vdots \\ m_i & \cdots & m_n \end{bmatrix}, \\
\mathbf{Sp} &= \begin{bmatrix} sp_1 & \cdots & sp_l \\ \vdots & \ddots & \vdots \\ sp_k & \cdots & sp_m \end{bmatrix}
\end{aligned}
\tag{3}
$$

Each node m_i has spring connections that are associated to this node pointing to up to seven neighbouring nodes (in three dimensions). details can be found in [5].

The general MSN model graph *St(P)* is paremeterised by a set of parameters defining the node mass mm_i, the spring stiffness constant sk_j, and optionally a damper constant sd_j. The choice of the parameters, which can be applied to all masses and springs of the network, or individually to domains of nodes and edges, is crucial for the mapping of real physical material behaviour on the physical simulation model described in the next section.

In this work, the node masses and spring constants were chosen to characterise a linear elastic material, typically a purely elastic elastomer. Alternatively, a non-linear spring force-distance model function can be used to model inealstic material behaviour.

The Multi-body physics (MBP) simulation is computed by the CANNON physics engine [11]. Strain sensor data can be directly calculated from the mass-spring network requiring only the distances between mass nodes (available directly from the *CANNON* engine). A low mass-node density is still sufficient.

3.2 Damages and Fatigue

Modelling of real damages is a challenge. Commonly damages are modelled statically, i.e., without an accurate time-dependent behaviour. Damages and fatigue of materials develop and change over time. Complex and hybrid materials like fiber-metal laminates show a broad range of possible damages (e.g., delaminations and cracks of fibers). The accurate modelling of damage is out of scope of this work. For the sake of simplicity, a damage within the material as a result, e.g., of an impact, is treated as a material inhomogeneity. Such an inhomogeneity is created in the mass-spring model by increasing the spring constants and reducing the damper constants of mass nodes belonging to a virtual damage by an order of magnitude.

In the experimental section the test structure is a simple plate that is composed of $14 \times 8 \times 3$ mass nodes, and damages are represented by volumes of 3×3 mass nodes. This is a coarse grain modelling of structure and damage. With respect to real damage and material behaviour, it is expected that there will be a large gap between simulation and real world experiments. With an significant increase of mass node density more accurate results can be obtained and structures with complex geometries or hybrid materials can be modelled. The computational time increase basically linearly in the order of $\Theta(n)$ since each mass node requires the solving of seven constraints [13].

4. Analysis and Learning Framework

A closed-loop unified simulation and analysis framework used in this work is shown in Fig. 5.

All input and output data is stored in SQL data basses accessed by an unified SQLjson interface with remote procedure call operational semantics and encoding SQL requests in JSON+ format (extended with objects, function code, and typed arrays). The SQL servers provide generic SQL requests and advanced support for a SQL-based file system organizing tables in directory trees.

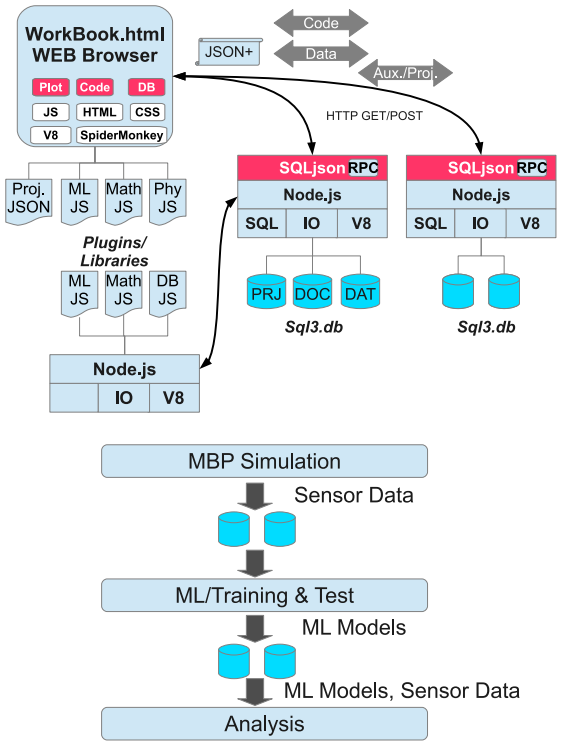


Figure 5. WorkBook connected to SQLjson server via HTTP

All computations are performed in *JavaScript* either in WEB-based Workbooks, similar to *Matlab* scripts and *Jupyter* workbooks, but without any server code execution (except the SQL servers). The data is exchanged via the SQLjson RPC interface with external SQL servers.

The workbooks (a command line version for *nodejs* execution is available, too) provide a basic support for code execution, plotting, communication, interactive tables and some more GUI elements. Additional plug-ins can be loaded providing extended math modules, physical simulation using the CANNON solver, and the ML plug-in providing a large set of ML models and algorithms including a generic ANN module [12] with a network architect. The network architect is used to construct the LSTM-RNN from basic cells.

There were a set of workbooks used for simulation (generating sensor data), machine learning deriving the discriminator functions, and finally analysis. The learned ML models as well as projects and documentations are stored in the SQL data bases, too.

5. Simulation and Evaluation

5.1 Test Structure

The test structure used to evaluate the proposed multi-instance learning is a rectangular plate that is layered at two sides on walls. The plate was modelled with $14 \times 8 \times 3$ mass nodes connected by springs and dampers. The simulation starts with all springs in equilibrium state. The gravity force acting on all mass elements result in a bending of the plate. After the highest bending was reached, the plate swings back (elastic model). This results in a damped vibration that is used for the detection of damages. Details can be found in [5].

A damage is introduced after a given simulation step (two situations with t_0=100/250) by relaxing of all springs between mass nodes of the virtual damage and between nodes of the damage to the surrounding nodes. The damage was a quadratic volume of $3 \times 3 \times 1$ mass nodes.

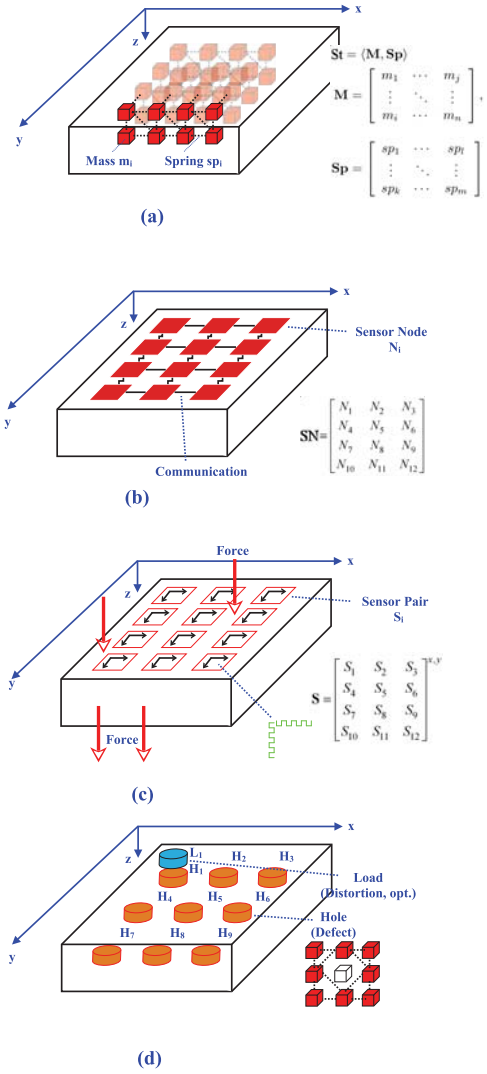


Figure 6. (a) MBP Modell of a plate (DUT) (b) Sensor Network applied to surface of DUT (c) Strain-gauge Sensors connected with sensor nodes (d) Hidden holes in the DUT (red) and optional load applied to the surface (blue) [5]

5.2 Sensor Data

The sensor network consists of 7 × 4 sensor nodes, shown in Fig. 6 (b). Each node samples two connected synthetic strain-gauge sensors (orthogonally orientated), shown in Fig. 6 (c), which were derived by computing the main strain of a volume of 2 × 2 mass nodes.. The originally undamaged plate DUT can be modified by adding virtual holes inside (by relaxing springs connecting mass nodes of the virtual hole) and prediction can be disturbed by applying additional load to the upper surface (not considered in this work), shown in Fig. 6 (d). A typical time record of both strain sensors was shown in Fig. 1.

Each load/damage situation was repeated ten times using Monte Carlo techniques to get signal and training variance. There are nine different damage cases (H1, H2, .., H9) differing in their location and two different time points at which the damage was created (t_0=100,250 sim. steps).

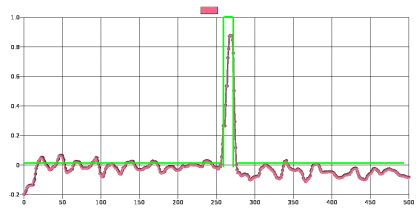
5.3 Experiments

The temporal sensor data sequences (strain in x- and y-direction) derived from the physical simulation was used to investigate different learner models, algorithms, and learning parameter sets. Each node of the sensor network uses a trained discriminator function implemented with a LSTM-RNN to detect damages in the spatial neighbourhood (radius < 3 arb. units). Each node performs training of the ANN and application of sensor data to the ANN. The LSTM consists of one hidden layer with five neurons. The output neuron was feed back to the memory and input gate cells.

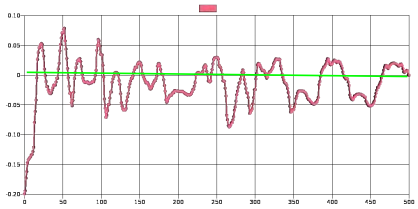
5.3.1 Training and Prediction

Some examples of the application of a trained discriminator function is shown inf Fig. 7. The training was performed iteratively to find optimal solutions:

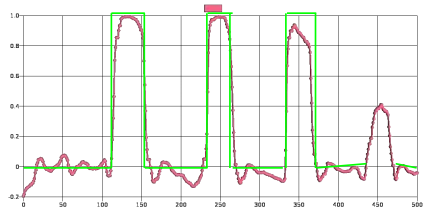
- The typical training time for one node varies between 8s and 80s using two positive case records (450 time points each) and two negative case records.



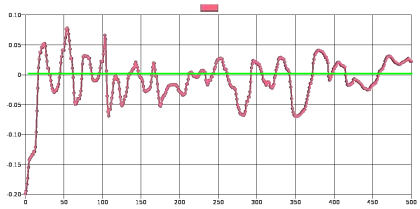
(a) T250 N1 H1



(b) T250 N1 H2



(c) T100 N1 H1



(d) T100 N1 H0

Figure 7. Some examples of the output of a trained node instance for two different damage cases (damage after T=100 and T=250 steps) and different sub-cases (Hole nearby node N1/H1 or in the neighbourhood H2 or no damage H0) using raw sensor data (raw SIGNAL approach). The green lines show the output of the post-processing using a threshold function.

- Application of the discriminator functions requires about 8ms for an entire signal record (500 time points).
- For all nodes a fixed LSTM-ANN architecture was chosen with two input neurons getting the raw time-resolved strain signals (s_x,s_y), five hidden layer neurons, and one output neuron providing the damage prediction.

5.3.2 Lessons learned

1. The ANN metrics (connection network, layers, nodes) are difficult to be determined and the optimal choice of the ANN metric can depend on the sensor node position and its sensor signal characteristics, the specific damage situation, and the signal features.
2. Learning of a hypothesis function satisfying the constraints is basically a random process. The learning process is often trapped in local minima with a hypothesis function not satisfying the constraints.
3. Different ANN metrics (hidden layer architecture), training metrics (learning rate, error threshold, training scores), and input vectors transformations (SIGNAL: raw sensor signal, DERIV: high-pass or derivation, SFFT: moving frequency window spectrum) were investigated to derive a suitable damage event discriminator function. Different settings can be improve training results of different sensor nodes! Derivation, high-pass, and spectral transformations did not improve the results. Therefore only raw unfiltered signals were processes by the ANN.
4. There is no monotonic convergence in the learning observed! In some cases a suitable hypothesis function was found after one training iteration, in other cases hundreds of iterations were needed to find a solution satisfying the constraints. Furthermore, model re-initialisation with random weights is required to find a suitable solution.
5. Small moving signal windows (batches) fed in the ANN during training do not show any benefit over a full time-scale frame of the entire signal

record that was finally used for training. In contrast, window slicing during training leads often to an empty hypothesis set (no suitable model function found).

6. The gradient learning approach (fitting the network weights) is not suitable. Internal low-level cost and error functions applied to one training record are not satisfying. Instead, higher level scoring functions evaluating a large set of training samples is required to find a solution.
7. Post-training with a low training rate of a pre-trained network (with a higher training rate) can improve the prediction quality (but sometimes instead the prediction quality of the discriminator function decreases).
8. Dynamic learning rate adaptation starting with a high rate at the beginning to explore the solution space with large steps and finally decreasing the learning rate to a low value to settle to a solution satisfying the constraints is promising, but without a guarantee of convergence.
9. Node permutation or re-use of pre-trained models from one node applied to another node failed always. Generalized discriminator function models could not be derived.

5.3.3 Distributed Multi-Instance Prediction

A trained discriminator function of a single network node can be used to detect the occurrence of a damage event within a spatially bounded neighbourhood of the node. There are multiple neighbouring nodes that can predict the same damage event (correlated clusters, see Fig. 8). This fact is utilized to perform distributed sensor fusion by computing the centre of mass position for a set of nodes predicting a damage event within a given time interval.

Results are shown in Tab. 1, concluding to:

- For early events (T100) all spatially different damage cases can be detected accurately.
- For later events (T250) some damage cases are not detected (H2, H8).
- The prediction quality of trained discriminator functions vary for different nodes (and spatial positions)
- There are some nodes of the network where the training failed an no discriminator function could be derived (nodes at logical positions (1,1), (1,2), and (4,1)). The reason is unknown.

Damage	**T100** $\mathbf{n_{found}}$	**T250** $\mathbf{n_{found}}$	**T100 error**	**T250 error**
None	0	0	-	-
H1	10	9	1.6 ± 0.3 (12%)	2.0 ± 0.02 (15%)
H2	7	0	1.7 ± 0.04 (13%)	-
H3	10	7	1.5 ± 0.1 (11%)	1.7 ± 0.1 (13%)
H4	10	9	1.2 ± 0.04 (9%)	2.0 ± 0.7 (15%)
H5	10	3	1.2 ± 0.2 (9%)	1.4 ± 1.7 (11%)
H6	10	10	1.1 ± 0.07 (8%)	1.4 ± 0.2 (11%)
H7	10	3	1.9 ± 0.2 (15%)	1.2 ± 0 (9%)
H8	10	0	1.1 ± 0.2 (9%)	-
H9	10	7	1.6 ± 0.3 (12%)	1.3 ± 0.3 (10%)

Table 1. Fusioned (7 × 4 sensor nodes) true-positive damage event predictions injected after 100 and 250 simulation steps (10 experiments and 9 different damage cases, position prediction error $\varepsilon \pm \sigma^2$ values are in arbitrary plate dimension units, percentage value is position error relative to plate size)

- The prediction quality (true-positive probability) is higher for early damage events (T100, about 95%) than for later damage events (T250, about 60%).

- The mean position error of the fusioned damage location is about 10% (relative to the plate size) and can be considered as a precise location estimation.

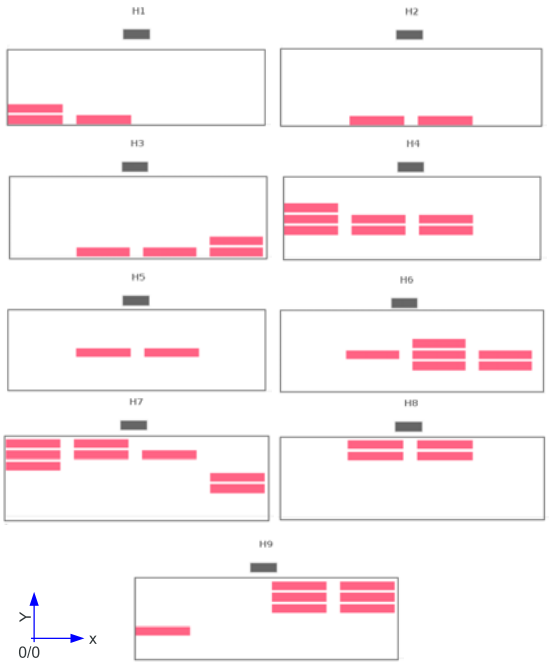


Figure 8. Examples of the response of the individual discriminator functions of the distributed sensor network on time records (around t=[100,150]) with different damage cases (H1..H9) occurred at t=100.

6. Conclusion

Although the training of state-based damage discriminator functions mapping time-series of raw sensor signals on damage detectors is a challenge, some remarkable results could be achieved. The quality (or even the successful training) of discriminator functions across the distributed sensor networks varies strongly. But due to overlapping damage detection areas (redundancy) of single nodes, the fusion of the outputs of all sensor nodes lead to a significant improvement of the overall damage event detection probability. The estimation of the location of the damage by simple calculation of the mass of centre of the sensor positions could be achieved with an accuracy of about 10% relative to the DUT size.

A hypothesis to be proved is that evolutionary algorithms with permutation and back-tracking can deliver better results and faster and more accurate learning convergence of the discriminator functions than gradient-based iterative ANN trainer used in this work.

Finally, the physical model used to for the simulation of the DUT has to be improved with respect of mass node density, real material behaviour, support for more complex geometrical structures and hybrid materials. Damage and fatigue modelling closer to real world experiments have to be established and evaluated.

7. References

[1] L. Roseiro, U. Ramos, and R. Leal, *Neural networks in damage detection of composite laminated plates*, in Proceedings of the 6th WSEAS Int. Conf. on NEURAL NETWORKS, Lisbon, Portugal, June 16-18, 2005, pp. 115-119.

[2] V. Ewald, R. M. Groves, and R. Benedictus, *DeepSHM: A Deep Learning Approach for Structural Health Monitoring Based on Guided Lamb Wave Techniques* (2019)

[3] E. P. Carden, *Vibration Based Condition Monitoring: A Review*, Structural Health Monitoring, vol. 3, no. 4, pp. 355-377, 2004.

[4] R. Jha and S. V. Barai, *Neural Networks and Genetic Algorithms in Structural Health Monitoring*, in Structural Health Monitoring Technologies and Next-Generation Smart Composite Structures, J. A. Epaarachchi and G. C. Kahandawa, Eds. 2016.

[5] S. Bosse, D. Lehmhus, *Robust detection of hidden material damages using low-cost external sensors and Machine Learning*, 6th International Electronic Conference on

Sensors and Applications (ECSA), 15-30 Nov. 2019, MDPI

[6] Y. Hua, Z. Zhao, R. Li, X. Chen, Z. Liu, and H. Zhang, *Deep Learning with Long Short-Term Memory for Time Series Prediction*, (2018) ArXiv 1810.10161v1 [cs.NE]

[7] Z. Tang, P. A. Fishwick, *Feed-forward Neural Nets as Models for Time Series Forecasting*, TR91-008 Computer and Information Sciences, University of Florida.

[8] F. A. Gers, J. Schmidhuber, (2001). *LSTM Recurrent Networks Learn Simple Context Free and Context Sensitive Languages*, IEEE Transactions on Neural Networks. 12 (6): 1333–1340. doi:10.1109/72.963769. PMID 18249962

[9] S. Yan, *Understanding LSTM and Its Diagrams*, WEB resource https://medium.com/mlreview/understanding-lstm-and-its-diagrams-37e2f46f1714 (accessed on 26.3.2020).

[10] N. M. Vural, S. Ergut, and S. S. Kozat, *An Efficient and Effective Second-Order Training Algorithm For LSTM-based Adaptive Learning*. ArXiv 1910.09857v3

[11] Hedman, S., CANNON, http://schteppe.github.io/cannon.js, accessed on 1.11.2019

[12] Neataptic, Thomas Wagenaar, https://github.com/wagenaartje/neataptic (accessed 1.12.2019)

[13] A.P Pentland. (1991) ThingWorld: A multibody simulation system with low computational complexity. In: Sriram D., Logcher R., Fukuda S. (eds) Computer-Aided Cooperative Product Development. WCACPD 1989. Lecture Notes in Computer Science, vol 492. Springer, Berlin, Heidelberg