
Proceedings Paper:
Tsialiamanis, G., Wagg, D.J., Dervilis, N. orcid.org/0000-0002-5712-7323 et al. (1 more author) (2020) A neat approach to structural health monitoring. In: Papadrakakis, M., Fragiadakis, M. and Papadimitriou, C., (eds.) EURODYN 2020: Proceedings of the XI International Conference on Structural Dynamics. EURODYN 2020: XI International Conference on Structural Dynamics, 23-26 Nov 2020, Athens, Greece. European Association for Structural Dynamics (EASD) , pp. 3832-3845. ISBN 9786188507227

10.47964/1120.9313.19022

EURODYN 2020
XI International Conference on Structural Dynamics
M. Papadrakakis, M. Fragiadakis, C. Papadimitriou (eds.)
Athens, Greece, 22—24 June 2020

A NEAT APPROACH TO STRUCTURAL HEALTH MONITORING

G. Tsialiamanis[1], D. J. Wagg[1], N. Dervilis[1], K. Worden[1]

[1]Dynamics Research Group, Department of Mechanical Engineering, University of Sheffield,
Mappin Street, Sheffield S1 3JD

Keywords: Structural health monitoring, machine learning, neural networks, neuroevolution of augmenting topologies (NEAT).

Abstract. *In the current paper, an application of the neuroevolution of augmenting topologies (NEAT) algorithm is considered in a structural health monitoring (SHM) application. The algorithm is a variation of genetic algorithms, applied in neural networks, and has the goal of optimising both the topology and the weights and biases of a neural network model. The algorithm is applied here to an SHM problem instead of using feedforward neural networks. The algorithm is called to search for the best-fitting topology in the task, which would otherwise be sought through experimenting with the size and number of the layers of the neural network. Having used the algorithm, the accuracy is found to be close to the one achieved using classically trained neural networks. Another aspect of the application is that subnetworks were defined for every damage case of the problem, whose topologies are much simpler than a fully-connected feedforward neural network. These subnetworks define classification submodels that may be used in different combinations, building models for a subset of damage cases and input features.*

1 Introduction

Structures are essential elements of human everyday life. From living in houses to moving from one place to another, structures play a key role in activities that are vital for everyone and monitoring them is required in order to make sure that they operate in the way in which they were designed. A malfunctioning structure could cause various problems. A building that has been damaged endangers the lives of those that live within it. Malfunctioning wind turbines may produce reduced electrical energy or even collapse, having as a result huge economic costs. A damaged bridge that requires stopping traffic on it, results in huge delays and even bigger financial damage to people who need to cross the bridge as part of their business. Taking into account the potential consequences of failure of structures, the need to monitor them emerges.

The discipline of acquiring data from structures through sensors, processing them and inferring results about the health state of the structure is called *structural health monitoring* (SHM). There are several approaches to monitoring structures but the most dominant use *data-driven* or *machine learning* (ML) methods [1], in order to infer results about the data. The models created are, in general, black-box models that rely on existing data acquired from a structure. There are several tasks that such a model may be called on to perform; a hierarchical way of classifying these tasks is given by Rytter in [2]:

1. Is there damage (*existence*)?
2. Where is the damage in the system (*location*)?
3. What kind of damage is present (*type/classification*)?
4. How severe is the damage (*extent/severity*)?
5. How much useful (safe) life remains (*prognosis*)?

The first two tasks are the most simple ones and in the context of machine learning, they have been dealt with using quite simple approaches. In [3], Mahalanobis distances were calculated between the baseline condition of a structure and the condition that is tested for the existence of damage. More complicated and more robust approaches for detecting the existence of damage are given in [4]. Another novelty measure that indicates existence of damage is given in [5]; it is defined using autoencoders in order to describe the manifold to which the data of the normal condition of the structure belong. If some testing data belong to the same manifold, the samples are considered as healthy; otherwise, it is considered that damage exists in the structure and it should be examined. Regarding the second level of Rytter's hierarchy; in [6], neural networks have been used to localise damage in an aircraft wing. The neural network was trained to classify damage into nine location different classes, according to features that were extracted from sensors placed on the wing.

Neural networks are machine learning algorithms that can approximate any arbitrary function [7], that explains the relationship between some input and some output values. For this reason, they have been widely used in the context of SHM and in many other disciplines including natural language processing, computer vision, etc. A major drawback of neural networks is that they require large training datasets in order to generalise well on unseen data. Moreover, selecting their architecture involves hyperparameters that have to be tuned by the user with a view to creating a model that fits the training data and generalises well on testing data.

In the current work, an approach to avoid selecting the exact architecture of a neural network for

an SHM problem is studied. This approach is called *neuroevolution of augmenting topologies* (NEAT) [8]. According to this approach, a genetic algorithm scheme is followed to train the neural network weights and at the same time create its topology from scratch [9]. The mainstream approach of training a feed-forward neural network requires trying different numbers and sizes for the hidden layers. In the end, the neural network that performs best on the task that they were trained for is used. This may be considered a brute force search for the best architecture and sometimes may take a lot of time. In addition, the architectures that are tested through this approach follow a specific pattern and more complicated approaches are not considered. The algorithm is tested here on experimental data with the task of localising/classifying damage on a structure.

2 Training neural networks using genetic algorithms

2.1 Genetic algorithms

Genetic algorithms are widely used in many scientific disciplines; they are a means of optimisation of given objective functions, as explained in [10]. In engineering, such algorithms have been used in order to optimise the design of whole structures [11], according to a quantity of interest, or even to select the model that best describes some acquired data [12].

Genetic algorithms draw inspiration from the natural selection scheme that exists in nature; individuals - structures, models or whichever entity is set to be optimised - are initially born with random characteristics. Afterwards, their strength is calculated through a fitness function that reflects the quantity whose optimisation is sought. In every generation, individuals that do better according to the fitness function, survive, while the rest die. One of the most important aspects of a genetic algorithm is the creation of offspring. Couples of individuals that have survived, create offspring that inherit characteristics from both their parents and have a probability of a mutation in their characteristics. Following this framework, the population moves towards the areas in the parameter space that generate more appropriate values regarding the fitness function and through creation of offspring through mutation, the space is further explored. The individuals may be points in an n-dimensional space representing parameters of a model, or even whole structures. There are many variations on the described genetic algorithm but they all, more or less, follow this scheme.

One of the main reasons that genetic algorithms are chosen as an optimisation tool is that the quantities that are being optimised or minimised may be a function without a specific formula. A quite common way of optimising function parameter values is using gradient descent [13], a requirement of which is having a formula that needs to be minimised or maximised and also, the formula should be differentiable. Quite often, the quantity that needs to be optimised may be a discrete one, such as the accuracy of a model in terms of number of correct decisions.

Another aspect that makes genetic algorithms powerful, is that they are thought to be exploring the parameter space to a greater extent than other methods, including gradient descent. An ideal solution to the problem of finding the optimal value for some quantities of interest, would be an exhaustive search in the whole parameter space. Through this kind of search, finding the optimal solution/global minimum or maximum is ensured, but in most cases it is computationally intractable. In contrast, a gradient-descent search, will follow a single path in the parameter space that depends on the initial value of the parameters and on the gradient of the objective function on each point during search and may hit a local minimum. Using a genetic algorithm, discrete points in the parameter space are chosen and the ones that do better in optimising the

objective function survive. The population slowly moves towards the areas in space that optimise the objective function, but at the same time explores a bigger part of the space than the gradient descent, since mutations create many different points in space rather than following a single path. This property is also the reason that a genetic algorithm is less likely to get stuck in local minima/maxima.

In the context of neural networks, where gradient descent through backpropagation is the most common way of training, tuning the trainable parameters may be alternatively performed using a genetic algorithm [14]. Using this approach, local minima may be avoided and also the loss function does not need to be differentiable. According to this framework of training, the neural network still has a fixed architecture. This idea may not allow finding the optimal solution, since the optimal network architectures vary according to the task for which they are used. Since the topology/architecture of the network also affects its functionality, exploring the space of topologies along with the space of parameters would be convenient and so a different scheme is proposed in [8] -the neuroevolution of augmenting topologies.

2.2 Neuroevolution of augmenting topologies (NEAT)

Following the aforementioned approach, the architecture of the network is not predefined. To avoid a predefined topology and force the algorithm to search for the one that performs best in some task, an augmenting topology scheme is followed. Again, the general genetic algorithm is used, as random individual neural networks are generated. The initial state of networks is considered to be a totally unconnected network with some input and output nodes. The first population is formed by creating random connections between these nodes. A fitness function is defined and performance of each network is evaluated in every generation. Consequently, the procedures of genetic encoding, mutation and crossover between networks have to be defined.

2.2.1 Genetic encoding

In order to perform the operations needed for the evolution of the population, an encoding of each individual neural network is needed. This encoding is referred to as the *genome* or *genotype*, and the network it corresponds to is the *phenotype*, in parallel with human DNA being an encoding and the phenotype being the way this encoding is translated into human characteristics. For the purposes of the algorithm, the genome will be a list with the connections, called *genes*, that have been defined in the network with a chronological order. The list will increase in length as generations pass, since new connections will be defined due to mutations or crossover.

By defining a chronological order or *historical marking* as mentioned in [8], a global naming for every mutation that has happened is achieved. Since mutations will be happening at random within the population, it is plausible that the same mutation may occur more than once. In such a case, the genes that correspond to the same connection in the networks, should be identified as so. During crossover, if a historical marking exists in both parents, it should be inherited to the offspring. An example of a genome and its phenotype is shown in Figure 1, where the encoding of a network is shown. Using the historical markings, encoding of the same connection in many different genes, is avoided.

In: 1 Out: 5 Weight: 0.7 Bias: 0.3 Enabled Hist. marking: 1	In: 2 Out: 5 Weight: -0.5 Bias: 0.12 Enabled Hist. marking: 3	In: 3 Out: 4 Weight: -0.17 Bias: 0.92 Enabled Hist. marking: 5	In: 5 Out: 4 Weight: 1.22 Bias: 0.83 Enabled Hist. marking: 6

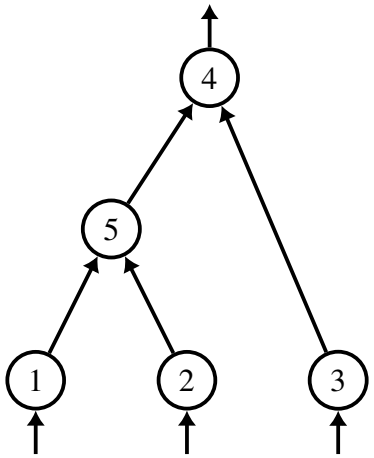

Figure 1: Example of a genome and the corresponding *phenotype* [8].

2.2.2 Mutation

One of the basic operations of a genetic algorithm is the mutation of individuals; this allows further exploration of the parameter space. In the case of NEAT, mutations should be able to explore both the network topology space and the parameter space of the networks. In order to explore the parameter space of the networks, mutations are allowed in the weights and biases of the connections and the activation and aggregative functions of nodes.

Mutations of the weights and biases are quite trivial. At the end of every generation, offspring have a probability of mutating a weight or a bias term from their connections. The mutations are simply changes in the values of the variables. The most common way of defining the new value is through sampling from a normal distribution with mean value μ equal to the current value of the parameter and a variance σ^2 that is defined by the user. This scheme introduces four hyper-parameters in the algorithm that the user has to define; the probabilities of a weight (p_w) or a bias mutation (p_b) and the power of the mutations or the variances (σ_w^2 and σ_b^2) of the distributions used to define the mutated value. Small values assigned to these hyper-parameters will result in a more thorough search in the parameter space, but also probably in a slower convergence of the algorithm.

The first mutation that applies to the topology of the networks is the *add connection* mutation; a connection is added between two nodes that so far have not been connected. Nodes and their IDs are shared amongst every individual in the population. Every network though has different connections defined in between the nodes and through this mutation type, new connections are defined within the mutated individual. The weight of the new connection is sampled from a random distribution and a historical marking is assigned to it. The opposite mutation should also be considered, the *disable connection*. This type of mutation, straightforwardly disables an existing connection in an individual. An example of how an add connection mutation is encoded

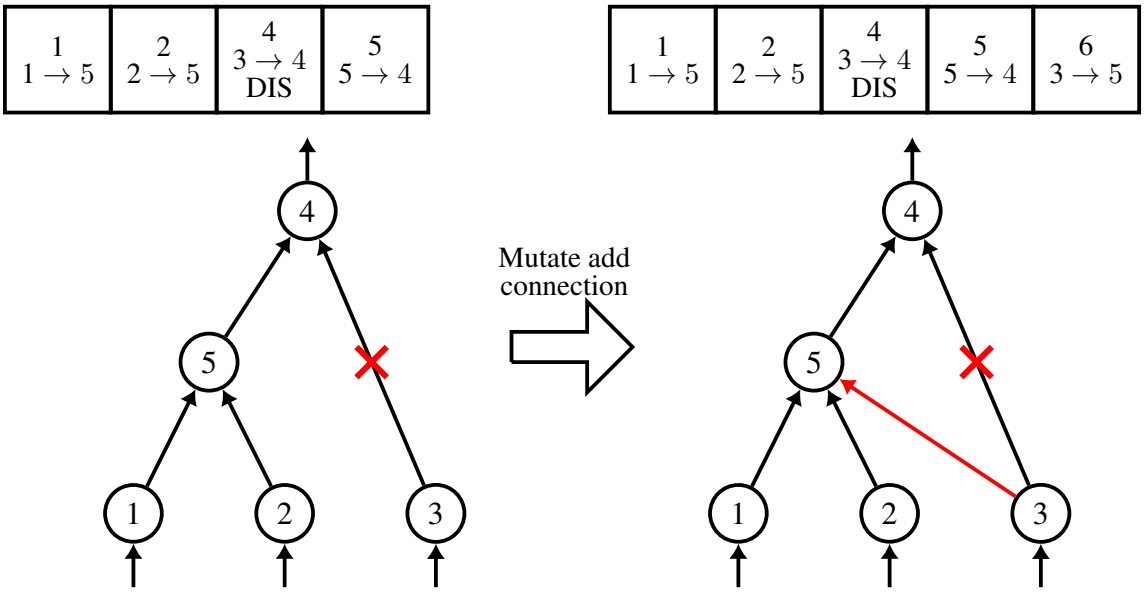

Figure 2: Example of an *add connection* mutation and the corresponding genome change [8].

is shown in Figure 2, where also a disabled connection is shown.

The second mutation regarding the topology is the *add node*, where a new node that did not exist so far is introduced. The way that such a node is added is by splitting an existing connection into two and placing a node between the two new ones. The old connection is disabled and the two new ones are added to the genome of the individual. The connection leading to the new node has a weight of 1 and the second one has weight equal to the weight of the old connection. The new node has a new ID and it is available in every individual to be connected through *add connection* mutations with other nodes. An example of the *add node* mutation is shown in Figure 3.

2.2.3 Crossover

Finally, the crossover operation is needed to completely define the genetic algorithm procedure. The crossover is performed using the genetic encoding of the networks. The genomes of the parents are placed in parallel according to the historical marking of each gene in the genomes. If a gene with a specific historical marking exists only within the genome of one parent, it is directly inherited to the offspring. If a gene with the same historical marking exists in both parents, it is inherited to the child by the most fit parent. An example is shown in Figure 4.

3 Application on experimental data

3.1 Experiment description

In order to test the algorithm, an experimental setup is considered [6]. The experiment is that of simulating damage on an aircraft wing. The wing was set in the laboratory and was excited with white noise using a shaker attached to its bottom surface. Sensors were placed on it as shown in Figure 5. The experiments were performed using the wing in both damaged and undamaged states. The damaged states were simulated by removal of one out of nine different wing panels. The panels that correspond to a different damage case are each shown in Figure 6.

The quantities that were recorded during the experiments with the damaged wing, were the

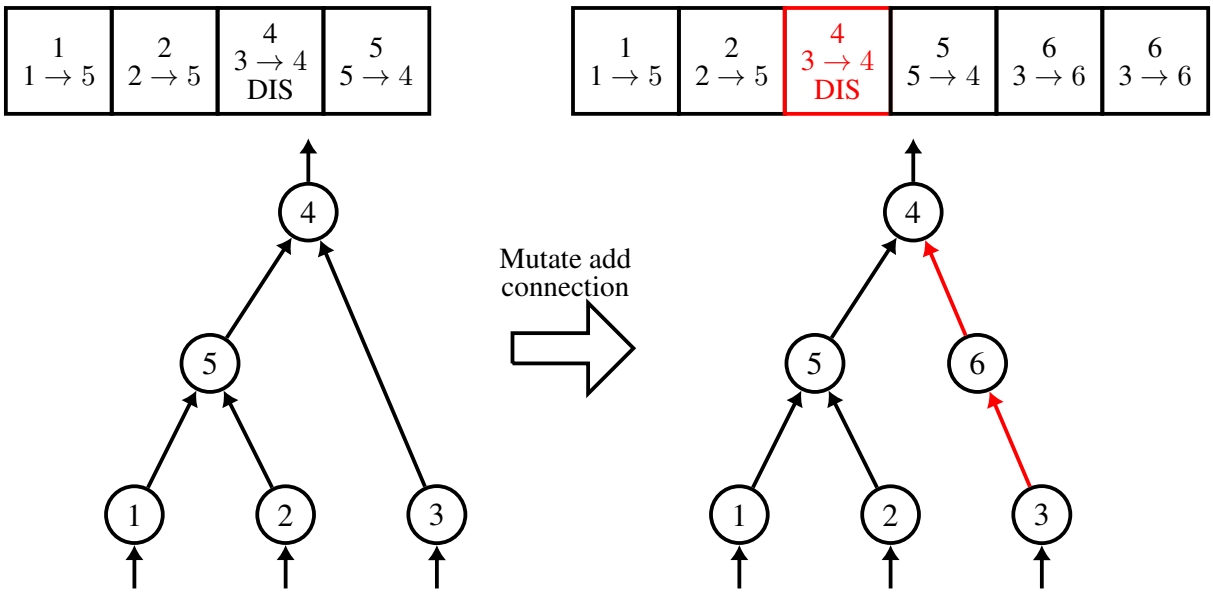

Figure 3: Example of an *add node* mutation and the corresponding genome change [8].

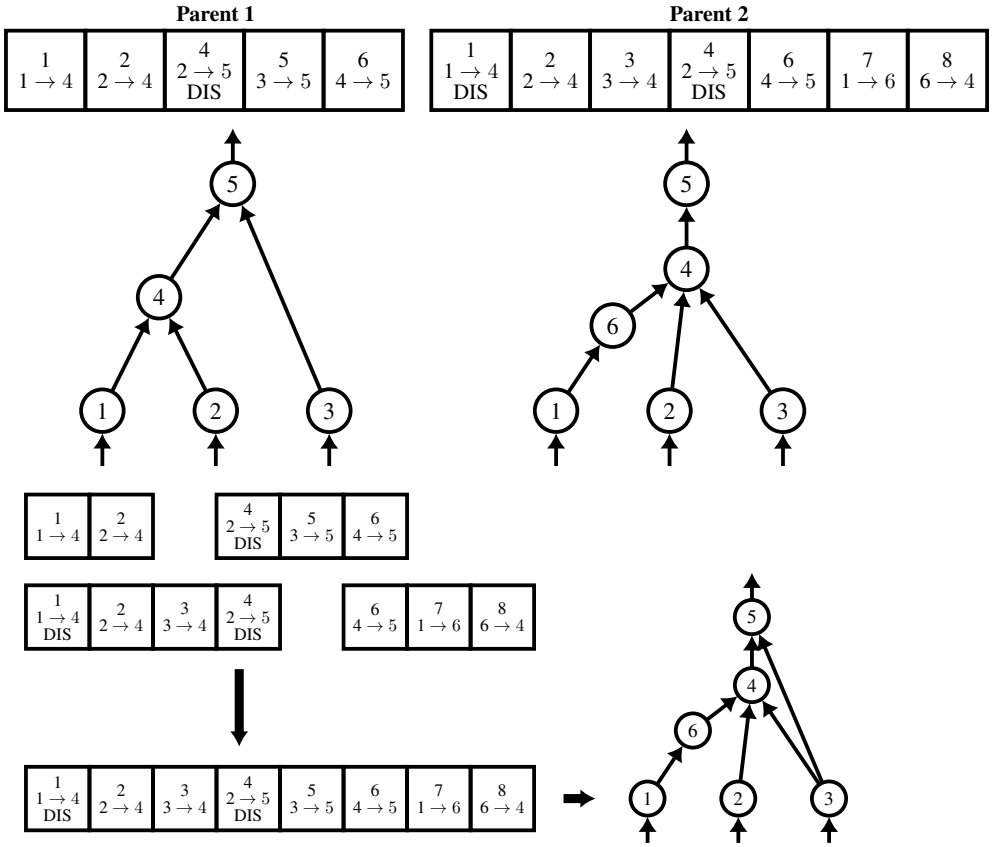

Figure 4: Crossover process given two parents and their genomes [8].

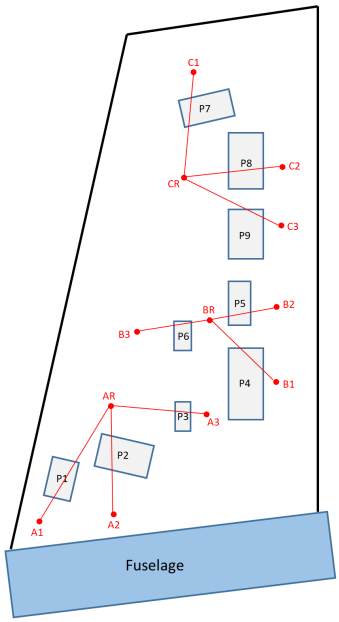

Figure 5: Configuration of sensors placed on the GNAT wing [6].

transmissibilities between two sensors. Transmissibilities are a quite useful feature, since knowledge about the excitation force is not required, in contrast to the case of a *frequency response function* (FRF). The transmissibilities were recorded between sensors AR and A1, A2 and A3 and similarly between the rest of the sets shown in Figure 5. This procedure resulted in nine different transmissibilities that were available for both the undamaged and the damaged states. Subsequently, intervals within the transmissibilities that were more sensitive to damage were identified "by eye"; this resulted in 72 different intervals and for each one of them, novelty indices were calculated between the normal condition transmissibilities and the testing cases. The novelty indices were calculated using the Mahalanobis distance,

$$D_{\zeta}^2 = (\mathbf{x_\imath} - \overline{\mathbf{x}})^T S^{-1} (\mathbf{x_\imath} - \overline{\mathbf{x}}) \tag{1}$$

where $\mathbf{x_\imath}$ is the sample whose Mahalanobis distance is calculated, $\overline{\mathbf{x}}$ is the mean vector of the observations and S^{-1} is the covariance matrix of the observations.

The type of model that was selected to classify the data was a neural network taking as inputs the values of the novelty indices and as output one out of nine classes of damage. In order to further reduce the feature space, a genetic algorithm was used to select the best subset of features out of the 72 novelty indices [15]; this resulted in nine novelty indices. The neural network was selected to be a feedforward neural network with one input layer, a hidden layer and a decision/output layer. The best size for the hidden layer was selected according to the performance of the networks in the validation set after having tried different sizes. The optimal network had nine nodes in the hidden layer and the its accuracy is shown in the confusion matrix in Table 1. The total accuracy was 98.14% on the testing set.

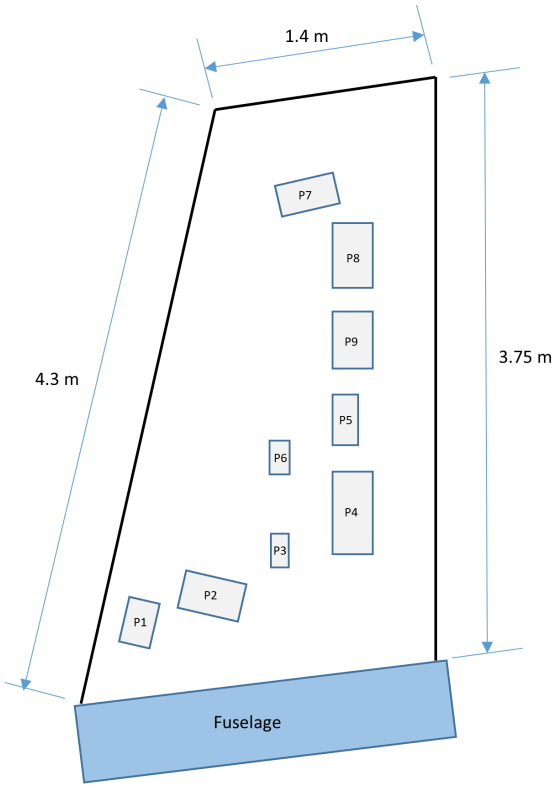

Figure 6: Schematic drawing of wing panels removed to simulate the nine damage cases [6].

Predicted panel	1	2	3	4	5	6	7	8	9
Missing panel 1	65	0	0	0	0	0	0	0	1
Missing panel 2	0	65	0	1	0	0	0	0	0
Missing panel 3	1	0	62	0	0	1	0	1	1
Missing panel 4	0	0	0	66	0	0	0	0	0
Missing panel 5	0	0	0	0	66	0	0	0	0
Missing panel 6	0	3	0	0	0	62	0	1	0
Missing panel 7	0	0	0	0	0	0	66	0	0
Missing panel 8	1	0	0	0	0	0	0	65	0
Missing panel 9	0	0	0	0	0	0	0	0	66

Table 1: Confusion Matrix of neural network classifier, test set, total accuracy: 98.14% [15].

Predicted panel	1	2	3	4	5	6	7	8	9
Missing panel 1	63	1	0	0	0	0	0	0	2
Missing panel 2	0	62	0	1	0	0	0	0	3
Missing panel 3	0	1	61	2	0	0	0	0	2
Missing panel 4	1	0	0	65	0	0	0	0	0
Missing panel 5	0	0	0	0	66	0	0	0	0
Missing panel 6	1	1	1	0	1	61	0	0	1
Missing panel 7	0	0	0	0	0	2	63	0	1
Missing panel 8	1	0	0	0	0	0	0	65	0
Missing panel 9	1	0	0	0	0	2	0	0	63

Table 2: Confusion Matrix of fittest NEAT individual, test set, total accuracy: 95.79%.

3.2 NEAT training

Taking into account the time needed to select the optimal size for the hidden layer, as well as the fact that the neural network selected had a restricted architecture since it was a feedforward one, application of NEAT was attempted on the same problem. By applying NEAT in this problem, a different topology for the network is expected and maybe a more interpretable model. The exact datasets were used in order to facilitate comparison between the two algorithms.

The population for NEAT was chosen to be a 500-neural network population with a *rectifier* or *relu* activation function on their nodes. The probabilities of mutating a weight or a bias term were both 0.7 and the mutate power was 0.1 in both cases. Moreover, the probabilities of adding a new connection or a new node were both 0.5 and the probability of deleting a connection was 0.05. The fitness function was set to be the accuracy of predictions given by a softmax function σ,

$$\sigma(\mathbf{x})_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \tag{2}$$

applied on the values of the output nodes of each network, using this function on the output of the networks, the algorithm allows their interpretation as probabilities.

3.3 Results and splitting into subnetworks

Having trained the population, after 3216 generations and about a day of training, the fitness of the "strongest" individual had been stable for many generations and so it was considered to have converged. The confusion matrix for the fittest individual on the testing set is shown in Table 2. The accuracy is slightly lower than the accuracy of the original work where feedforward networks were used.

It is worth isolating the sub-networks that describe how each output node is activated. This is done by looking for paths from every input node to each output node separately. Doing so, the subnetworks that are found for every damage class are shown in Figure 7. Connections with green colour represent positive weights while the ones in red correspond to negative weights. It is particularly interesting that subnetworks whose structure is quite simple have been created. For example the subnetwork corresponding to panel 9, is affected only by input features 3, 7 and

Predicted panel	1	2
Missing panel 1	64	2
Missing panel 2	0	66

Table 3: Confusion Matrix regarding classifying cases of missing panels 1 and 2 using the respective subnetworks, accuracy = 98.48%.

8. This separation could result in further insight into the physics of the problem, as it is more interpretable than a fully connected neural network.

Another aspect of this approach is that these subnetworks could be combined and used as a modular classification model for a subset of damage cases. For example, one may be interested in creating a classifier that would be able to distinguish between the classes of missing panel 1 or missing panel 2. The procedure to be followed is to use the subnetworks of the two cases and assign a label to the sample according to which subnetwork generated a greater value on its output node. By applying this algorithm for the case of damage cases 1 and 2, the confusion matrix for the testing data is shown in Table 3. It is also worth noting that using this approach, the values of the input features that are needed are those of features 1, 2, 3, 4, 5, 6, 8, since they are the subset of input features that are used by the two subnetworks.

4 Conclusions

An alternative approach to training neural networks for SHM was presented, that of augmenting topologies. The algorithm used does not have a specific topology for the neural network that is going to be trained, but is searching for an optimal one using a genetic algorithm. Following this approach, the size and architecture of the network to be used does not have to be predefined. Even though selection of a topology is avoided, new training hyperparameters have to be defined, for example the probabilities of the creation of a new node or a new connection in the population of neural networks.

The results of the classification task in the SHM example described, reveal that the accuracy of the fittest individual was close to the accuracy of a feedforward neural network trained using backpropagation. The training time, taking into account recent advances in hardware used in applying backpropagation in neural networks, is much larger for the NEAT algorithm. The training time for NEAT could be lower if a parallelisation scheme was followed, which was not used in the current work.

Although there are some disadvantages in applying such an algorithm, some advantages emerged; the first is that the fitness function may be a discrete quantity that could not be used as a cost function in a feedforward neural network. Furthermore, the isolation of subnetworks presented, generated quite simple sub-models for each damage case, whose extraction is not applicable in a feedforward neural network. The use of such subnetworks for classification of a subset of damage cases was also shown to be possible. These modular models used a subset of features which may prove useful in cases that some features may not available. In such cases, inference may still be made for subsets of damage cases.

Finally, more simple or more complex architectures that are not achievable by using a mainstream feedforward network are considered. The approach of augmenting topologies allows a greater variety of networks to be tested for the task at hand. In many problems, this may result in models that achieve greater accuracy compared to other neural networks. The algorithm has also the

potential of allowing recurrent connections that may also fit the underlying physics of a problem and approximate better the real relationship between inputs and outputs.

Acknowledgements

The authors would like to acknowledge the support of the European Union (EU). G.T is supported by funding from the EU's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement DyVirt (764547). The authors also gratefully acknowledge the support of the UK Engineering and Physical Sciences Research Council (EPSRC) through grant references EP/R003645/1, EP/R004900/1 and EP/S001565/1.

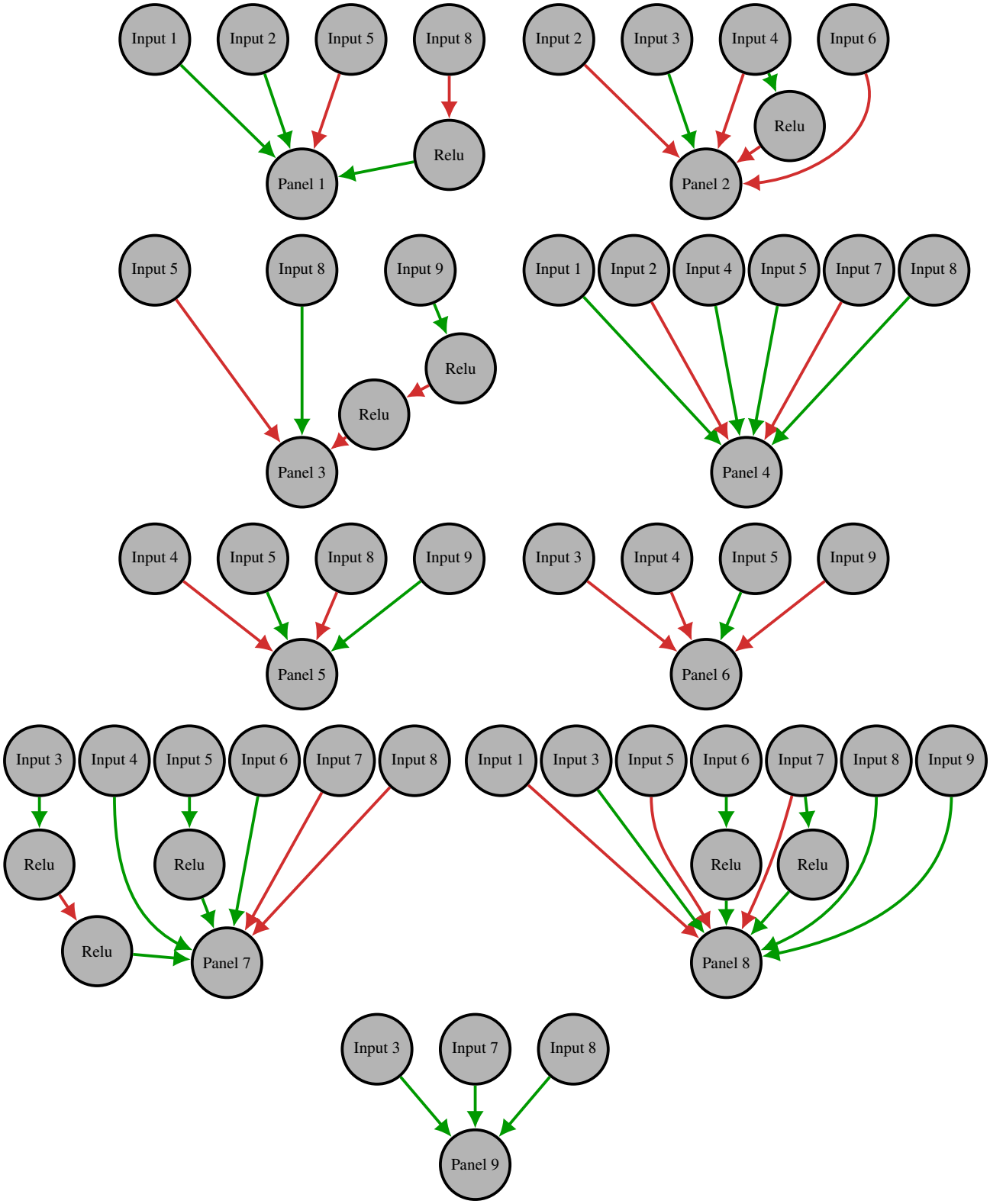

Figure 7: Subnetworks for each damage case.

REFERENCES

[1] C. R. Farrar, K. Worden, Structural Health Monitoring: A Machine Learning Perspective, John Wiley & Sons, 2012.

[2] A. Rytter, Vibrational based inspection of civil engineering structures, Ph.D. thesis (1993).

[3] E. Papatheou, G. Manson, R. J. Barthorpe, K. Worden, The use of pseudo-faults for novelty detection in SHM, Journal of Sound and Vibration 329 (12) (2010) 2349–2366.

[4] N. Dervilis, I. Antoniadou, R. J. Barthorpe, E. J. Cross, K. Worden, Robust methods for outlier detection and regression for SHM applications, International Journal of Sustainable Materials and Structural Systems 2 (1/2) (2016).

[5] K. Worden, Structural fault detection using a novelty measure, Journal of Sound and Vibration 201 (1) (1997) 85–101.

[6] G. Manson, K. Worden, D. Allman, Experimental validation of a structural health monitoring methodology: Part III. Damage location on an aircraft wing, Journal of Sound and Vibration 259 (2) (2003) 365–385.

[7] G. Cybenko, Approximation by superpositions of a sigmoidal function, Mathematics of Control, Signals, and Systems 2 (4) (1989) 303–314.

[8] K. O. Stanley, R. Miikkulainen, Evolving neural networks through augmenting topologies, Evolutionary Computation 10 (2) (2002) 99–127.

[9] M. Mitchell, An introduction to genetic algorithms, MIT press, 1998.

[10] D. Whitley, A genetic algorithm tutorial, Statistics and Computing 4 (2) (1994) 65–85.

[11] V. K. Koumousis, P. G. Georgiou, Genetic algorithms in discrete optimization of steel truss roofs, Journal of Computing in Civil Engineering 8 (3) (1994) 309–325.

[12] A. B. Abdessalem, N. Dervilis, D. Wagg, K. Worden, Model selection and parameter estimation in structural dynamics using approximate Bayesian computation, Mechanical Systems and Signal Processing 99 (2018) 306–325.

[13] S. Ruder, An overview of gradient descent optimization algorithms, arXiv preprint arXiv:1609.04747 (2016).

[14] D. J. Montana, L. Davis, Training feedforward neural networks using genetic algorithms., in: IJCAI, Vol. 89, 1989, pp. 762–767.

[15] K. Worden, G. Manson, G. Hilson, S. Pierce, Genetic optimisation of a neural damage locator, Journal of Sound and Vibration 309 (3-5) (2008) 529–544.