Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science

Explainable neural networks

Diploma thesis

Vojtěch Drahý

Study program: Open Informatics
Specialisation: Cyber Security
Supervisor: Ing. Radek Mařík, CSc.

Prague, May 2023

Thesis Supervisor:
Ing. Radek Mařík, CSc.
Department of Telecommunication Engineering
Faculty of Electrical Engineering
Czech Technical University in Prague
Czech Republic

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Drahý Vojtěch**	Personal ID number:	**481891**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Computer Science**		
Study program:	**Open Informatics**		
Specialisation:	**Cyber Security**		

II. Master’s thesis details

Master’s thesis title in English:

Explainable neural networks

Master’s thesis title in Czech:

Vysvětlitelnost neuronových sítí

Guidelines:

The aim is to propose a computational method for evaluating explainability of neural networks with regard to the input data and network components.
Recommended steps:
1/ Create a survey of published methods or their approximations for evaluating explainability of neural networks with regard to the input data and network components.
2/ Select suitable methods and implement them or adapt existing software libraries appropriately.
3/ Design an experimental validation and evaluate the obtained results.
4/ Try to apply the proposed procedure to bridge structure data.

Bibliography / sources:

[1] N. Deo, Graph theory : with applications to engineering and computer science. Prentice-Hall of India ; Prentice-Hall International, 1994.
[2] M. Sundararajan, A. Taly, and Q. Yan, “Axiomatic attribution for deep networks,” 2017.
[3] E. Štrumbelj and I. Kononenko, “Explaining prediction models and individual predictions with feature contributions,” Knowledge and Information Systems, vol. 41, no. 3, pp. 647–665, 2014.
[4] R. Mitchell, J. Cooper, E. Frank, and G. Holmes, “Sampling permutations for shapley value estimation,” 2021.
[5] Freeborough, W.; van Zyl, T. Investigating Explainability Methods in Recurrent Neural Network Architectures for Financial Time Series Data. Appl. Sci. 2022, 12, 1427.
[6] Shapley, Lloyd S. (August 21, 1951). "Notes on the n-Person Game II: The Value of an n-Person Game". Santa Monica, Calif.: RAND
Corporation.

Name and workplace of master’s thesis supervisor:

Ing. Radek Mařík, CSc. Department of Telecommunications Engineering FEE

Name and workplace of second master’s thesis supervisor or consultant:

Date of master’s thesis assignment: **10.02.2023** Deadline for master's thesis submission: **26.05.2023**

Assignment valid until: **22.09.2024**

Ing. Radek Mařík, CSc.	Head of department’s signature	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature		Dean’s signature

III. Assignment receipt

The student acknowledges that the master’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master’s thesis, the author must state the names of consultants and include a list of references.

.

Date of assignment receipt	Student’s signature

Declaration

I hereby declare I have written this diploma thesis independently and quoted all the sources of information used in accordance with methodological instructions on ethical principles for writing an academic thesis. Moreover, I state that this thesis has neither been submitted nor accepted for any other degree.

In Prague, May 2023

....
Vojtěch Drahý

Acknowledgements

I would like to thank my supervisor Ing. Radek Mařík, CSc., for his patience, guidance and help. But most of all, I thank him for the knowledge and experience he gave me. Furthermore, I thank Matouš Pikous for his patient text proofreading.

Abstract

The explainability of neural networks is an often discussed issue in machine learning. Neural networks have experienced massive development in recent years - they are used in a wide range of tasks, from image recognition to language processing to marketing. However, people perceive neural networks as black boxes that have some inputs, some outputs, and a lot of unknowns inside. This problem of unknown, poorly grasped and humanly incomprehensible behaviour makes applying them in fields such as medicine or critical infrastructure difficult. Apart from obtaining the result, in these fields, is even more important to be able to justify it and describe the way in which it came about. Therefore, it is necessary to investigate neural networks and make them explainable and interpretable. Explainability tells which parts of the input data are important to the model and which parts of the model process them. Interpretability based on the conclusions of explainability aims to create a humanly understandable summary of the aspects of the solved problem from the point of view of the neural network.

This work presents an overview of different approaches to solving the explainability of neural networks. Furthermore, a task from the critical infrastructure field in materials engineering is presented here. The work aims to use neural networks and their explainability to conduct an initial investigation of the given task, to find out the aspects of machine learning application and propose the direction of the further procedure from the results achieved.

Keywords: Neural networks, explainability, communities detection, Shapley values, material engineering.

Abstrakt

Často diskutovaným problémem ve strojovém učení je vysvětlitelnost neuronových sítí. Neuronové sítě zaznamenaly v posledních letech masivní rozvoj – používají se v široké škále úloh, od rozpoznávání obrazu přes zpracování jazyka až po marketing. Lidé však vnímají neuronové sítě jako černé skříňky, které mají uvnitř nějaké vstupy, nějaké výstupy a spoustu neznámých. Tento problém neznámého, špatně uchopeného a lidsky nepochopitelného chování ztěžuje jejich uplatnění v oborech, jako je medicína nebo kritická infrastruktura. V těchto oborech není důležité jen říct výsledek, ale ještě důležitější je umět jej zdůvodnit a popsat způsob, jakým k němu došlo. Proto je nutné neuronové sítě prozkoumat a učinit je vysvětlitelnými a interpretovatelnými. Vysvětlitelnost říká, které části vstupních dat jsou pro model důležité a které části modelu je zpracovávají. Interpretovatelnost na základě závěrů vysvětlitelnosti má za cíl vytvořit lidsky srozumitelný souhrn aspektů řešeného problému z pohledu neuronové sítě.

Tato práce představuje přehled různých přístupů k řešení vysvětlitelnosti neuronových sítí. Dále je zde uvedena úloha z oblasti kritické infrastruktury v materiálovém inženýrství. Práce si klade za cíl pomocí neuronových sítí a jejich vysvětlitelnosti provést prvotní prozkoumání zadané úlohy, zjistit aspekty aplikace strojového učení a z dosažených výsledků navrhnout směr dalšího postupu.

Keywords: Neuronové sítě, vysvětlitelnost, detekce komunit, Shapleyho čísla, materiálové inženýrství.

List of Tables

List of Figures

Contents

Chapter 1

Introduction

1.1 Motivation

People usually think of neural networks as a black box. They believe that when they have input data and corresponding labels, they just need to take some large enough neural network architecture and push such a universal box on any problem thanks to the enormous computing power of graphics cards. The first, and often the last choice in business practice, is the creation of several fully-connected / convolutional layers of a neural network, with the expectation that it will somehow turn out. People often do not care about the distribution of the input data, whether some parts of the input data carry similar information, whether the proposed architecture correctly reflects the problem being solved and whether it is robust enough to solve problems with critical deficiencies in the training data. Although neural networks provide a decision on a sample, people are unable to explain the path to the decision. This very property prevents neural networks and other machine learning algorithms from being applied in critical systems such as medicine or critical transport infrastructure, including civil engineering. Generating the results is not enough; the “reason why” is a business fundamental.

Understanding the input data, sections on which the model focuses, and understanding the significance of the components and layers that make up the model is important for its further development and for justification in a specific industrial solution. It is important to go down to the level of individual neurons and trainable parameters. Revealing the insignificant ones will make it possible to prune the model and reduce its memory requirement. Nowadays, neural networks are often used in computer vision applications on smartphones, where there is strong pressure to optimise hardware requirements and reduce memory requirements.

Thanks to the explainability of data and statistical models, it is, therefore, possible to better understand the problem being solved, optimise the model architecture (topology and trainable parameters), better set the policy of the processes that the network is supposed to manage and present humanly comprehensible conclusions about the functioning of the model, which are an essential part of every security management principle critical infrastructure.

1.2 Problem statement

The goal of the solved problem is to determine the material parameters of the concrete beam of a bridge; the schema of the object is shown in Figure 1.1a. Load-dependence curves on the material deflection are the input data. The required output is a vector of four parameters characterising the material: modulus of elasticity (E_c), compressive strength (F_c), tensile strength (F_t) and fracture energy (G_f).

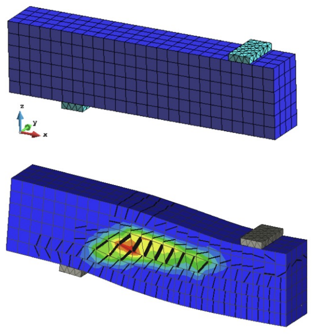
Conventionally, such a task is solved using the finite element method, a theory described by Brenner and Scott in [BS08]. Its main disadvantage is that it takes a lot of time to compute. In practice, extensive problems, from the point of view of a complex object in shape and deformation, are incalculable. The goal is to replace the finite element method calculations with a statistical model that will allow recalculation in near real-time. In practical use, structural engineers on bridge constructions will be able to perform immediate calculations on mobile devices. Therefore, the utilization of neural networks for this task is examined here.

The data is subject to three problems. The first is the considerable lack of samples, the second is their variability (including labels), and the third is non-periodically sampled curves. Three datasets were available. The first A-dataset with 100 samples, the second B-dataset with 400 samples, and the C-dataset with 1000 samples. Each sample represents a 2D curve of 61 points.

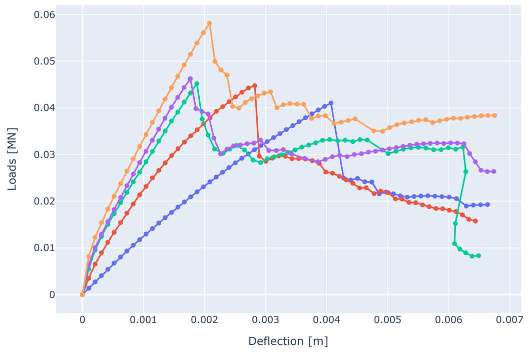
As a criterion for the success of the statistical model, a relative error of the middle value of up to 20 % was established since that error occurs when using the finite element method.

In Figure 1.1b are five example samples from the A-dataset. It is noticeable that the curves are similar in shape but differ in the point where they break and are scaled differently. There is, therefore, a problem of data following a global course but locally varying.

The dataset consists of curves (the sample is a matrix of two-dimensional points) representing the input data. Each curve is assigned the four-dimensional vector of material parameters that represent labels. A fundamental problem in training the models was the critical lack of samples. Figure 1.1b shows that the individual samples are similar in the first third of their curves (elastic deformation of the material). On the contrary, significant



(a) The material object on which the dataset was generated with displayed deformation.



(b) Example of samples from the A-dataset.

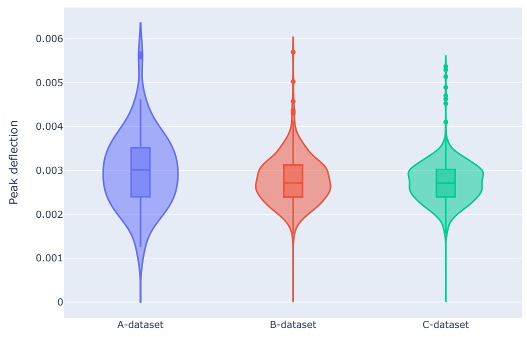
differences are visible between samples in the other parts (inelastic deformation, fracture, and deformation).

In general, the load behaviour of the material can be described as follows. First, elastic deformation occurs when the material bends without breaking and is able to return to its original shape after the end of the load. As the load continues, elastic deformation is followed by inelastic deformation. Slight cracks in the material may already begin to occur during it. However, these cracks are not considered to be fundamental damage to the material. On the curve of dependence of load on deflection, they appear as slight local decreases. At the highest point of the curve, the material breaks and is deformed. Subsequently, the curve decreases or undulates depending on how the cracks spread in the object. It can be concluded that the moment of destruction of the material, i.e. the position of the breaking point (peak) on the curve, will play a significant role in the parameters characterising the quality of the material.

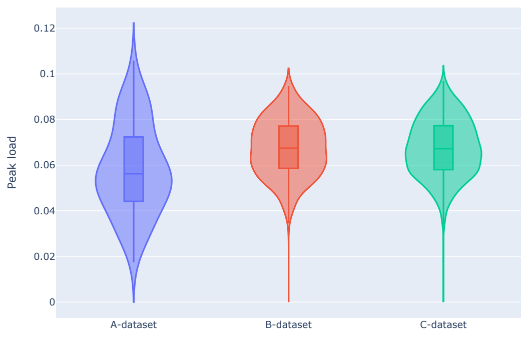
Figure 1.3 shows the dependence of the material parameters (labels) values on the peak point of the sample curve (the point where the curve breaks and the material failure occurs from a physical point of view). Evidently, the values are not correlated, and we can speak of an even distribution. Heteroskedasticity can be observed for the dependence of the fracture energy G_f. With the increasing number of samples in the dataset, according to this initial exploration, it could be said that the location of the breaking point is slightly correlated with the fracture energy and thus has an effect on this parameter.

Figure 1.4 shows the correlation of the example points (it is the Pearson correlation between certain points identified by their indexes on the curve; calculated over the entire dataset). The red colour points are outliers determined by the **ccf method** introduced by Barratt et al [BAB20] and implemented by Satman et al. [SAAA21]. From the example, there are visible correlations of nearby points, thus, it makes sense to deal with the detection of communities. It is noticeable that points close in order carry similar information, while points far from each other on the curve (sample) are not correlated. From this, it can be considered that processing the curve in parts that carry similar information could not only reduce the number of trainable parameters of the model and thus its computation complexity but also improve the functioning of the model itself.

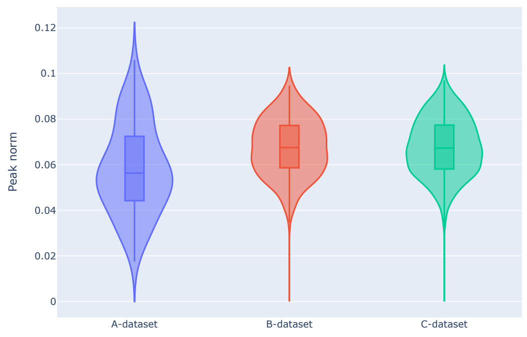
Figure 1.2 shows the distribution of characteristics (load, deflection, norm - distance from the origin) of the peak point for individual datasets. It is visible that the A-dataset, marked in blue, has very significantly scattered peak points - so it can be said that, unlike the two other datasets, it contains samples with strongly different material qualities. Combined with the fact that this dataset has the least number of samples, the worst results can be expected on it. In contrast, the C-dataset has peak points located relatively close to each other with less variance, and therefore good results can be predicted on it.



(a) Deflection

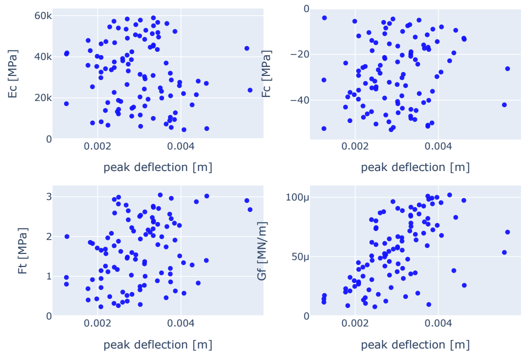


(b) Load

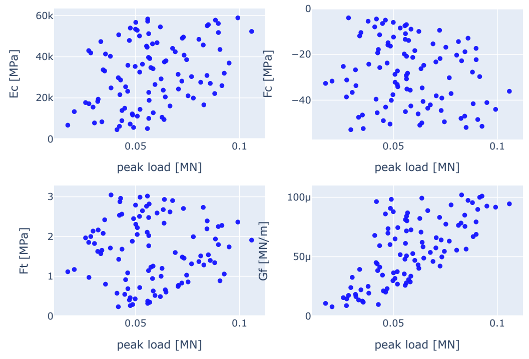


(c) Norms

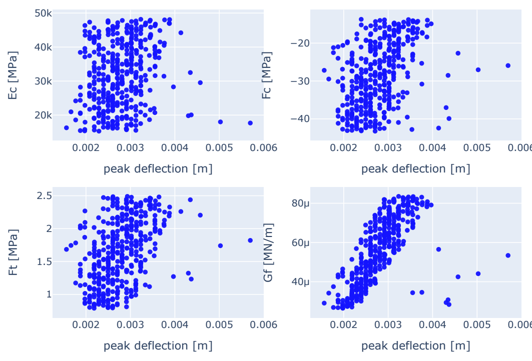
Figure 1.2: Violin distribution of peaks points.



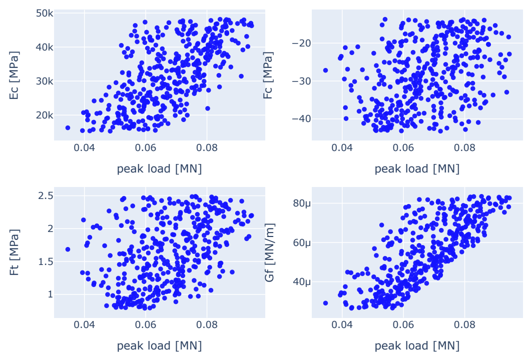
(a) A-dataset (deflection)



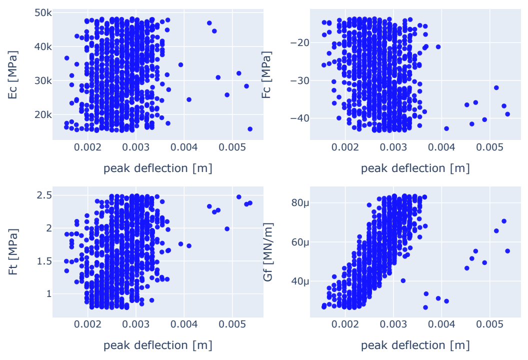
(b) A-dataset (load)



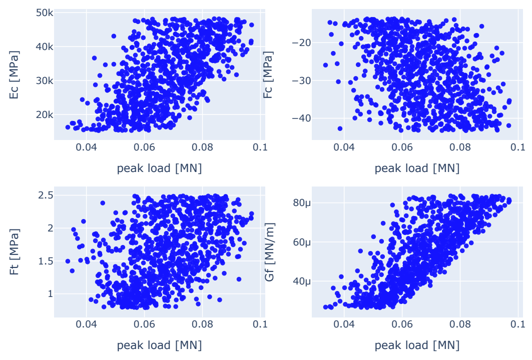
(c) B-dataset (deflection)



(d) B-dataset (load)

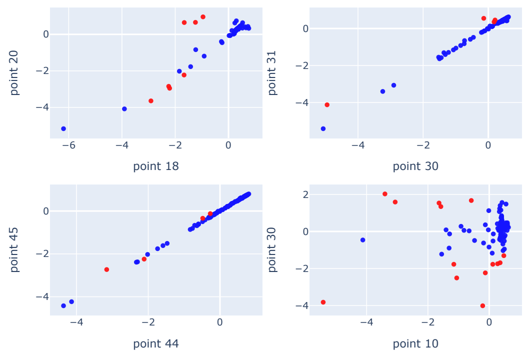


(e) C-dataset (deflection)

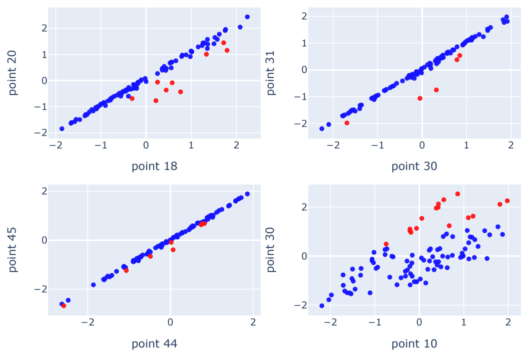


(f) C-dataset (load)

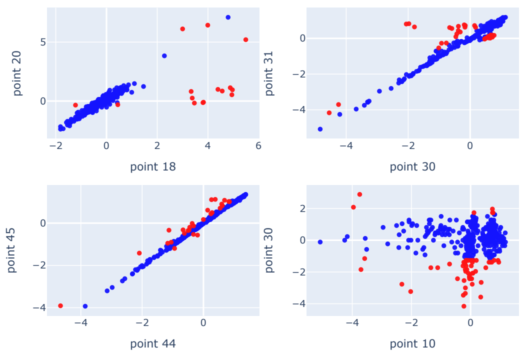
Figure 1.3: Distribution of peaks point according to the material parameters.



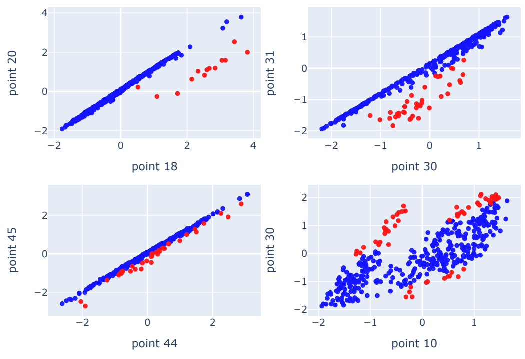
(a) A-dataset (deflection)



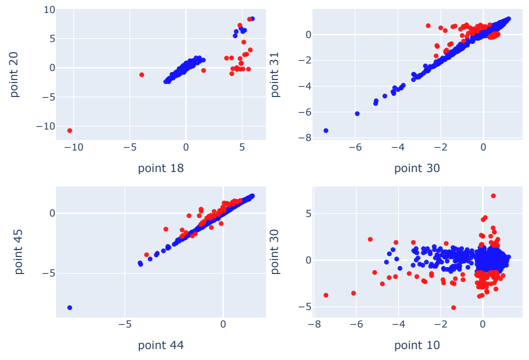
(b) A-dataset (loads)



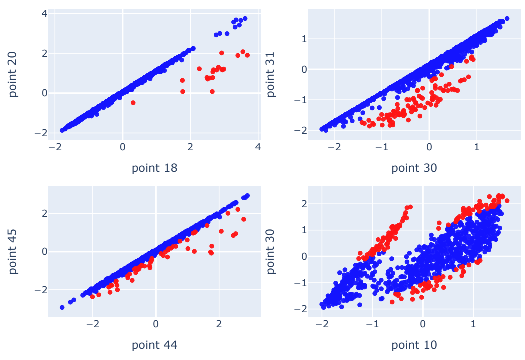
(c) B-dataset (deflection)



(d) B-dataset (loads)



(e) C-dataset (deflection)



(f) C-dataset (loads)

Figure 1.4: Correlation of example points.

Chapter 2

State of the art

This part provides an overview of state-of-the-art approaches in conventional neural networks, the explainability of machine learning models, and community detection on graphs to simplify explainability tasks.

2.1 Analysing neural network topologies

This subsection provides a state-of-the-art overview of analysing and explaining neural network topologies.

2.1.1 Analysing and arranging topologies

The different topologies and architectures may be more suitable for different tasks and machine learning problems. Depending on the processed data, the architectures may have different strengths and weaknesses in fitting and generalising. When developing a neural network, it is advisable to consider several possible architectures. A common procedure in the search for a suitable model is to convert the solved problem to a known and already solved problem, typically from the field of computer vision. Different approaches, for example, for sequence processing, overlap in many ways. When solving specific and atypical problems, existing architectures are combined with the aim of obtaining, orchestrating and putting together the solution for the given task. To solve problems with a small amount of training data or poorly distributed data (for example, subject to heteroscedasticity), it is necessary to look for topologies that describe the essence of the problem.

Stier et al. [SGGZ18] state that machine learning is always built on large, mostly fully connected architectures; however, large parts are redundant and irrelevant. The worst case of the topology behaviour lies in the complete impossibility of optimisation and immediate overfitting. When developing a neural network, it is important to be able to identify those parts of its topology that directly cause inaccurate or bad model output.

The method of dividing the neural network into submodels to increase accuracy is discussed by Yang et al. in [YGG$^+$13]. Their approach divides the solved problem into several smaller sub-models while always training together those whose outputs are correlated. It is not a method that aims to reduce the number of trainable parameters but to better organise the training process. Yang et al. state that the method helps to resolve the internal interference inherence inside large networks.

Two fundamental concepts in system processing and developments are applicable to identify the optimal architecture of the neural network model - *bottom-up* and *top-down*. The *bottom-up*, promoted by psychologist E. J. Gibson [Gib66], methods start from a small architecture with only a few trainable parameters or layers and interactively extend the model range to improve accuracy until overfitting occurs. The *top-down* (promoted by Harlan Mills and Niklaus Wirth in 1970 [Mil83]) approach methods start with a large architecture that overfits on initialisation and prunes the topology of the architecture by removing the low-contributing components. The first of these approaches requires prior knowledge of the solved problem. When proposing the model architecture, there must be awareness of the direction of the design workflow. The second of the mentioned approaches has the advantage that we do not need any apriori knowledge about the problem at the beginning of the model development. Applying a suitable method of explaining the model's components (both from the point of view of computational complexity and the provided outputs) leads to the knowledge of the information flow and the possibility of rebuilding the architecture with the aim of highlighting and trimming irrelevant parts. Thanks to the trimming, the number of trainable parameters is reduced. Together with parameters and components reduction, the computational complexity of training is also reduced. By removing confusing components for decision-making, the model outputs can become more accurate.

Furthermore, there is one more purely practical business reason why it is important to reduce the model's complexity and number of parameters - large neural networks take up a large part of the memory, apart from training, also during evaluation. Although this is usually not a problem on servers and computers, it is a significant problem on mobile devices such as smartphones or IoT devices. An unnecessarily complex neural network model in a phone application can make a demand on hardware and slower service functioning.

2.1.2 Explainability of neural networks

Before the introduction approaches interpretation and explanation of neural network models, there is a terminology list related to the work presented by Saleem et al. [SYK+22].

- **Understandability** represents the ability to characterize a model without knowing its internal structure. It is possible to answer questions such as “Where should the model be used?”, “What are the limits of the model?” and “For what inputs does the model fail?”

- **Explainability** represents the ability to characterize the model’s internal structure. The explainability answers the questions “Which parts of the model are irrelevant?”, “Which parts of the input are confusing for the model?” and “How model makes decisions and what influences them?”.

- **Interpretability** is the justification and human-capable summarising of the explainability results. The discussions about the principles of how the model works are conducted.

- **White box** is a model with completely provided information about architecture and parameters. It is possible to describe the computational graph of the model structure and mathematically formalise it. However, a specific description is not available to justify and explain the decision-making process, including the logic of assigning a semantic description.

- **Transparent box** is the addition to the white box. It provides not only a complete description of the structure but furthermore it also allows the justification of that behaviour and its explanation.

- **Black box** is a model with a completely unknown architecture and the internal structure cannot be altered. It is unclear whether some very extensive architectures of deep learning neural networks should be called white or black boxes. Although very extensive network topologies are formally white boxes, but their scale makes them so difficult to read and informatively explain from a human point of view that they can be considered black boxes.

Understandability, explainability, and interpretability can have several human-capable goals. These goals differ depending on the purpose for which the model is investigated.

- **Input data** - investigating the usefulness of individual parts of the input data. In the case of images and point clouds, these can be bounded areas of pixels and points. In processing time-dependent events, the usefulness of input data can be examined with regard to either the flow of information over time or the characteristics of the features in the given time records. Furthermore, the input data can be examined from the point of view of multiple instance learning, i.e. which objects or communities within the input sample are carriers of information.

- **Components of the model** - investigating the contribution of the sub-parts that make up the complex model. It is an examination of the topology of the computational graph with the aim of reflecting the complexity and structure of the solved problem. Revealing the low-benefit parts of the topology allows the model's architecture to be rebuilt, pruned, and improved.
- **Trainable parameters** - examining the usefulness of individual trainable parameters of the model or their groups (such as neurons or groups of neurons) will allow revealing those that reduce accuracy or those that are redundant for functioning. Due to the reduction of their number, the computational effort will be reduced and the training of the model will be accelerated while freeing up the hardware memory for training.
- **Information flow** - monitoring the flow of information between individual components of the model - vertices of the computational graph - will allow less significant connections to be cancelled. This simplifies the topology and speeds up differentiation during training.

There are various specific methods for the explainability of statistical models and neural networks, depending on the goals mentioned above and, simultaneously, on the computational complexity of the algorithms of the given methods for the explanation.

Activation mapping approaches

Tjoa and Guan [TG19] state that in computer vision, a common and popular approach to explaining models processing image data is the use of Class Activation Mapping approaches. These approaches aim to point to the parts/areas of the input sample that the model considers essential for its decision. A typical visualisation of the output of these methods is a heatmap covered over the input sample (image).

According to Zhou et al.,u [ZKL$^+$15], the CAM method (Class Activation Mapping) rearranges the model (neural network based on convolutional layers) by replacing the last fully-connected layers with a layer of global average pooling and one fully-connected layer with softmax function. The modified network generates a feature map. The final class activation map is computed as a linear combination of (previously generated) feature maps. Figure 2.1 shows the computation process.

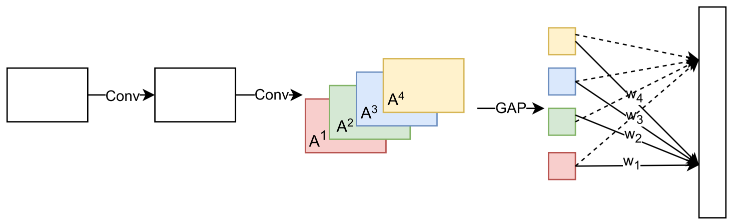


Figure 2.1: Diagram of CAM-network. [Koz21]

The following equation is expressed as computing the CAM heatmap for class k.

$$\mathrm{CAM}^i = \sum_k w_i^k \mathbf{F}^k, \tag{2.1}$$

where w_i^k is a weight connecting the k^{th} feature map with the i^{th} class, and $\mathbf{F}^k$ is the global-average-pooled output obtained in the following way:

$$\mathbf{F}^k = \frac{1}{\mathrm{Z}} \sum_m \sum_n \mathbf{A}_{ij}^k, \tag{2.2}$$

where $\mathbf{A}^k$ is the k^{th} feature map at location (m, n) and Z is the count of entities inside the feature map.

According to Rahimzadeh et al., [RPSM21], the disadvantage of the CAM method is that there could be a loss of spatial information. The problem lies in global average pooling, especially in processing large feature maps.

The Grad-CAM method, introduced by Selvaraju et al. [SCD+19], performs the computation using the gradient information obtained with respect to the last convolutional layer. Compared with the pure CAM method, where the weights of the linear combination of the feature maps are obtained from the single fully-connected layer, the weights used in Grad-CAM are taken from the gradient information. The algorithm is described by Isaac Castro [Cas19] in three following steps in Algorithm 1.

The Grad-CAM method has an improvement called Score-CAM. The generated activation maps are used as convolutional masks in the original input sample. The forward-passing of such newly obtained masked samples creates score-based weights related to the classes. According to Wang et al., [WDYZ19], the score-CAM results are a linear combination of activation maps and score-based weights.

Algorithm 1 Grad-CAM algorithm

```
Input:  the neural network model, input sample and corresponding label.
Output:  heatmap.

  1. Let be $y^i$ output for the $i^{\text{th}}$ class.  The gradient for the feature map $\mathtt{A}^k$ will
     be
```

$$f'_{i,k}(y, \mathrm{A}) = \frac{\partial y^i}{\partial \mathtt{A}^k}. \tag{2.3}$$

```
  2. The outputs of the global average polling will be
```

$$\alpha^i_k = \frac{1}{\mathtt{Z}} \sum_m \sum_n f'_{i,k}(y, \mathrm{A}). \tag{2.4}$$

```
  3. The heatmap $\mathtt{H}^i$ for the $i^{\text{th}}$ class is defined as
```

$$\mathtt{H}^i = \mathrm{ReLU}(\sum_k \alpha^i_k \mathtt{A}^k). \tag{2.5}$$

Integrated gradients

The Integrated gradients is an attribution method introduced by Sundararajan et al. [STY17]. The main advantage of the method is that it does not require any modification to the original network, and at the same time, it is extremely simple to implement. The principle of the method is based on repeated calculations of the gradient of the computational graph.

The method is formally described in the following expressions. The model is represented by the following function:

$$f : \mathbb{R}^n \longrightarrow [0, 1], \tag{2.6}$$

the input to the model is $\vec{x} \in \mathbb{R}^n$, and the $\vec{x}' \in \mathbb{R}^n$ is the baseline input (often represented by zero vector).

Integrated gradients are calculated as accumulated gradients obtained along the straight line path from the baseline input $\vec{x}'$ to the current input $\vec{x}$. The path gradient for the i^{th} dimension of the input sample is defined in the following way:

$$\mathrm{InterGrads}_i(\vec{x}) = (\vec{x}_i - \vec{x}'_i) \cdot \int_{t=0}^{1} \frac{\partial f(\vec{x}' + t \cdot (\vec{x} - \vec{x}'))}{\partial \vec{x}_i} dt \tag{2.7}$$

The approximation of the integral for implementing the integrated gradients is performed via a summation in a way that follows.

$$\mathrm{InterGrads}_i^{\mathrm{approx}}(\vec{x}) = \frac{\vec{x}_i - \vec{x}'_i}{s} \cdot \sum_{k=1}^{s} \frac{\partial f(\vec{x}' + \frac{k}{s} \cdot (\vec{x} - \vec{x}'))}{\partial \vec{x}_i}, \tag{2.8}$$

where s is the count of steps in the Riemann approximation.

Opposite to the advantages, such as simple implementation, there are also disadvantages of the method. Since the approach needs to iteratively evaluate the gradient of the model with gradually slightly different inputs, Ancona et al. [ACÖG19] state that a limitation of integrated gradients is its relatively high computational cost. Ancona et al. also warn that the method for deep neural networks exacerbates the problem of shattered gradients [BFL+17] because in the model function lies a high count of piece-wise linear regions, so the course of the gradient becomes discontinuous. The result of the method turns into noise sensitively depending on small variations in input to the model. The method is thus useful for small-scale networks or piecemeal applications to subcomponents of the larger model.

The experimental evaluation of the Integrated gradients approach was evaluated (during the research for the purpose of this thesis) on the material object (bridge) problem, but the interpretability of explainability was very questionable. For that reason, the method of Integrated gradients is not presented in the chapter with experimental evaluations and results.

Ablation

The principle of the ablation approach iteratively involves removing or ablating parts or regions of the input sample to receive the region's contribution. Freeborough and Zyl present the approach [FvZ22] in investigating financial time series data.

For the explainability of time-dependent data problems, Freeborough and Zyl recommend using forward filling with the average of previous inputs as ablation. They propose this for the purpose of passing on contextual information along the time dimension. This is because the context component is ablatively removed in recurrent neural networks using the average of previous inputs. The mechanism of the approach is described in Algorithm 2.

The ablation approach is a simplification of the Integrated gradients method and a modification for time-dependent data. A fundamental advantage over Integrated gradients is simply counting the outputs of the model and not its gradients.

The disadvantage of the ablation approach is the dependence on its pattern. When ablation is performed, it is necessary to know its appropriate shape (such as zeroing, averaging, and adding Gaussian noise). Different implementations can lead to different

Algorithm 2 Ablation algorithm

```
Input: model, sample $\boldsymbol{X} \in \mathbb{R}^{l \times f}$ (where $l$ is the time dimension of the sample and $f$
is the features dimension).
Output: error matrix $\boldsymbol{E}_{avg} \in \mathbb{R}^{w \times f}$ (where $w$ is the size of the input window
step).

  $E_{avg} \leftarrow [\,]$
  $\hat{y} \leftarrow$ model($\boldsymbol{X}$)
  for q in f do
      $E_f \leftarrow [\,]$
      for i in 1:k do  $\triangleright$ $k$ is the range of region to be ablated through iterations.
          $\boldsymbol{X}_{ablated} \leftarrow$ ablate($\boldsymbol{X}, i + w, q$)
          $\hat{y}_{ablated} \leftarrow$ model($\boldsymbol{X}_{ablated}$)
          $E_f \leftarrow$ concat($E_f$, $\frac{|\hat{y}_{ablated} - \hat{y}|}{\hat{y}}$)
      end for
      $E_{avg} \leftarrow \frac{1}{k} \sum_{j=1}^{k} E_f[j]$
  end for
```

models' explanation results. Another major disadvantage in the specific implementation of ablation along the time dimension of the input sample is that the method does not consider the mutual interaction of regions (communities) of instances of the given sample. Thus, it is not possible to detect contiguity and causality between temporal events in a given sample.

Game theory approach

The problem of the explainability of a neural network can be thought of as a task from the field of game theory. Sub-parts of the model - submodels, components, layers, and parameters - can be seen as agents (players) of a team cooperative game. This point of view has the advantage of dealing with interactions such as information that passes between individual agents. The problem of explainability is thus converted to the problem of the contribution examination. The contribution is examined not only with regard to the abilities of the given agent as an individual but also with regard to its cooperative abilities and the expediency of cooperation with other agents in a broader team concept.

Pelegrina and Siraj [PS22] state that the mechanism is to distribute the proceeds of a cooperative game among the coalition members by measuring the marginal contribution to the final outcome.

In a coalition cooperative game, the game results of different quality can be obtained by applying different subsets of agents. The prerequisite for investigating such behaviour is the determinism of the game; the computational model must always provide the same output for repeated evaluations of the same input sample.

A fundamental solution to the problem of the contribution of cooperative game agents is the Shapley value calculation concept introduced by Lloyd S. Shapley [Sha51]. To obtain the contribution of the i^{th} agent to the success of the team in the game, the method calculates the differences in the contributions of all subsets of agents playing with and without the i^{th} agent. The formal expression is the following:

$$\phi_i(\text{model}, \vec{x}) = \sum_{S \subseteq U \setminus \{i\}} \frac{|S|!(|U| - |S| - 1)!}{|U|!} [\text{model}(S \cup \{i\}, \vec{x}) - \text{model}(S, \vec{x})], \tag{2.9}$$

where $\vec{x}$ is the input sample, U is the set of rankings[1] of all agents from which a model is constructed, S are (iteratively) all subsets of the agents rankings, and ϕ_i is the Shapley value of the i^{th} agent. The higher is the Shapley value, the more important is the role of the agent (component) in the team (model); on the contrary, a negative value indicates the confusing behaviour of the agent leading to a deterioration of the model's performance.

The calculation of the Shapley number for a set of agents proceeds analogously to the calculation for a single agent. The difference consists only in replacing the ranking number i with the set of ranking numbers of the corresponding agents for which the calculation is performed.

The Shapley values can be expressed in terms of permutations [MCFH22]

$$\phi_i(\text{model}, \vec{x}) = \frac{1}{|U|!} \sum_{\sigma \in P} [\text{model}([\sigma]_{i-1} \cup \{i\}, \vec{x}) - \text{model}([\sigma]_{i-1}, \vec{x})], \tag{2.10}$$

where P is the set of (all) permutations for agent rankings, σ is the permutation that assigns a rank j to the element i in a way

$$\sigma(i) = j. \tag{2.11}$$

A simple example of the σ permutation is the following. Let the permutation be

$$\sigma = (2\ 4\ 1\ 3), \tag{2.12}$$

the vector of items is

$$\vec{v} = (x_1\ x_2\ x_3\ x_4)^T \tag{2.13}$$

and is turned into the

$$\vec{v'} = (x_3\ x_1\ x_4\ x_2)^T. \tag{2.14}$$

[1]Ranking of an agent is there used in the meaning of identification.

The $[\sigma]_{i-1}$ represents the set of agent rankings that are ranked lower than i according to the order given by the permutation σ.

According to Mitchell et al., [MCFH22], the Shapley values are unique and satisfy the four properties:

1. **Symmetry** - Two agents with the same contribution to all coalitions (subsets of the rest of agents) have the same Shapley values. Formally, it is expressed by formula $\forall S \subseteq U \backslash \{i, j\} : \text{model}(S \cup \{i\}) = \text{model}(S \cup \{j\}) \Rightarrow \phi_i = \phi_j$.

2. **Efficiency** - The sum of Shapley values of independent agents belonging to a certain subset (coalition) is equal to the Shapley value of a given subset. Formally, it is expressed by equation $\sum_{i \in S} \phi_i = \phi_S$.

3. **Linearity** - The Shapley value of the combined situations (games) is equal to the sum of particular Shapley values of isolated situations. Formally, it is expressed by equation $\phi(\text{model}_1 + \text{model}_2, \vec{x}_1 + \vec{x}_2) = \phi(\text{model}_1, \vec{x}_1) + \phi(\text{model}_2, \vec{x}_2)$.

4. **Null agent - dummy** - The agent with a negligible impact on all subsets of agents has a Shapley value equal (or almost equal) to zero. Formally, it is expressed by formula $\forall S \subseteq U \backslash \{i\} : \text{model}(S \cup \{i\}) - \text{model}(S) < \epsilon_1 \iff \phi_i < \epsilon_2$, where ϵ_1 and ϵ_2 are almost zero.

According to Deng and Papadimitriou [DP94], the computation of the Shapley values is in terms of the theory of complexity of the NP-hard problem. Mitchell et al. [MCFH22] propose the Monte Carlo estimation via sampling the permutations used in equation 2.10. The approximation lies in reformulating the task into

$$\overline{\phi}_i(\text{model}, \vec{x}) = \frac{1}{|U|!} \sum_{\sigma \in \Pi} [\text{model}([\sigma]_{i-1} \cup \{i\}, \vec{x}) - \text{model}([\sigma]_{i-1}, \vec{x})], \tag{2.15}$$

where $\Pi \subset P$ is a subset of uniformly sampled permutations.

Mitchell et al. propose an approach of sampling permutations of the length d by a relaxation to the Euclidean hypersphere

$$\mathcal{S}^{d-2} = \{\vec{x} \in \mathbb{R}^{d-1} : \ \|\vec{x}\| = 1\}, \tag{2.16}$$

which is similar to the approach presented by Plis et al. [PLC10]. Relaxation simplifies the problem of selection of uniformly distributed samples. The method lies in mapping the hypersphere points to the nearest permutation represented by a vertex of a Cayley graph [Cay78] inscribing the hypersphere.

The basis of the method is to determine the mapping of permutations $\sigma \in P_d$ of length d to the Euclidean space $\mathbb{R}^d$ in a following way:

$$\begin{aligned} &\vec{p_i} = \sigma^{-1}(i), \forall i \in \{1, 2, ..., d\} \\ &\sigma \in P_d \\ &\vec{p} \in \mathbb{R}^d \end{aligned} \tag{2.17}$$

Algorithm 3 Sampling permutations on permutahedron (Mitchell et al.)

1. `The permutahedron is shifted and scaled.`

$$\hat{\sigma}^{-1} = \frac{\sigma^{-1} - \mu}{\|\sigma^{-1}\|}, \tag{2.18}$$

`where`

$$\mu = \frac{1}{2}(d+1, d+1, ...) \tag{2.19}$$

`is the mean vector of all permutations.`

2. `Uniformly selected vector` $\vec{x} \in \mathbb{R}^{d-1}$ `and` $\|\vec{x}\| = 1$ `from the surface` $\mathcal{S}^{d-2}$.
3. `Project` $\vec{x}$ `to hyperplane in` $\mathbb{R}^d$.

$$\hat{\vec{x}} = \hat{M}^T \vec{x} \tag{2.20}$$

`where` $\hat{M}$ `is the matrix`

$$\begin{pmatrix} 1 & -1 & 0 & \dots & 0 \\ 1 & 1 & -2 & \dots & 0 \\ & \vdots & & \ddots & \\ 1 & 1 & 1 & \dots & -(d-1) \end{pmatrix} \tag{2.21}$$

`normalised by rows.`

4. `The nearest permutation` $\hat{\sigma}^{-1}$ `is found by maximising the dot product.`

$$\hat{\sigma}^{-1} = \underset{\vec{s}}{\operatorname{argmax}}(\hat{\vec{x}} \cdot \vec{s}) \tag{2.22}$$

`The vector` $\vec{s}$ `is just a reordering of the same constants, which implies that the argmax can be simplified into the sorting`

$$\sigma = \texttt{argsort}(\hat{\vec{x}}) \tag{2.23}$$

`such that`

$$\hat{\vec{x}}_{\sigma_0} \leq \hat{\vec{x}}_{\sigma_1} \leq \ ... \ \leq \hat{\vec{x}}_{\sigma_n}. \tag{2.24}$$

The mapping of all permutations of length d creates the vertices of the permutahedron polytope, which is a $d-1$ dimensional object in d dimensional space. The inversion of permutations gives a Cayley graph of the symmetric group. Each vertex in the permutahedron has $d-1$ neighbours and each differs in one transposition. The process of sampling the permutations is described in Algorithm 3.

The Neuron Shapley framework introduced by Ghorbani and Zou [GZ20] is an application of the mentioned principle and method based on the calculation of Shapley numbers. Their contribution is in quantifying and estimating the performance of the layers and neurons of the deep network. Their work resulted in the finding that removing just thirty filters corresponding to the highest Shapley numbers completely destroys the accuracy of Inception-v3 on ImageNet. Based on this, it is possible to very well mark the important parts of the network and cut off the confusing ones. Ghorbani and Zou solve the high asymptotic complexity of calculation Shapley numbers by approximating the calculation and introducing a multi-armed bandit algorithm for detecting agents (neurons, filters, layers) with the highest Shapley numbers.

2.2 Communities detection

In this subsubsection, the detection of communities on graphs is introduced. The goal is to introduce approaches that can be used to simplify computationally demanding methods of explainability of neural networks. Furthermore, an approach to creating topologies of neural networks based on the output of the detection of communities of features in input samples is outlined here.

Since within one community, there are parts of the data or entities that carry similar information. Explainability can be done at the level of communities and not directly at the level of individual data features. In the case that the input to the neural network is a vector with a dimensionality, at least in the higher tens, the evaluation of the Shapley method is extremely computationally demanding. The same is in the case of investigating the trainable parameters of a neural network. Models usually have at least on the order of thousands of parameters, but quite commonly, millions. Generating all Shapley subsets would thus lead to an incomputable problem. On the other hand, community detection, whether at the level of input data or model layers or components, or at the level of trainable parameters, can significantly reduce the amount of generated subsets due to the aggregation of entities with similar information, and thus enable Shapley to be evaluated in a reasonable time.

Based on the detection of communities over the input data, from the point of view of community features of the sample, it is possible to gain better insight into the solved problem and thus better design the topology of the model. Thus, it allows the model to be better grasped to process parts of data carrying similar information.

2.2.1 Terminology in the communities detection

Before the declaration of the terms below, there is an assumption of knowledge of the theory from N. Deo: Graph Theory with Applications to Engineering and Computer Science [Deo94]. Formalisation is defined according to Réka and Barabási [AB02] and M. Newman [New10].

- Complex network - graph representing interdependent entities and displaying non-trivial topological features defined as

$$\mathcal{N} := \mathcal{N}(\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{E}}), \tag{2.25}$$

where

$$\boldsymbol{\mathcal{A}} := \boldsymbol{\mathcal{A}}(\mathcal{A}_1, ..., \mathcal{A}_n), \tag{2.26}$$

where $\mathcal{A}_i$ is the i^{th} agent and

$$\boldsymbol{\mathcal{E}} := \{(\mathcal{A}_i, \mathcal{A}_j) | \mathcal{A}_i, \mathcal{A}_j \in \mathcal{N} \wedge \mathcal{A}_i \neq \mathcal{A}_j\} \tag{2.27}$$

is the set of interactions (edges) between them. The edges together define patterns of entity connections that are not purely random. The information could be carried by degree distribution (mostly in unweighted graphs), edge weights (such as correlation), directions of edges (reciprocity of entities), or by a hierarchical structure which reflects dependencies between entities. While analysing the complex networks, we must ask which properties we are looking for, which entities are more important than others, how to distribute entities between communities and look for some repeating pattern in the network. The characteristics properties are degree heterogeneity and the existence of bridge vertices and small worlds.

- Network agent - node

  $$\mathcal{A}_i := \mathcal{A}_i(\mathcal{P}_i) \in \mathcal{N} \tag{2.28}$$

  representing the entity of the problem with properties $\mathcal{P}_i$ related to the network $\mathcal{N}$. The agent could carry various types of information and may be connected to other agents (for example, by correlation of the carried information). From the game theory point of view, we can speak about cooperative agents in case they are mostly correlated with each other. The community of these agents can be considered for the analysis as one superagent.

- Agent ranking - mapping

  $$m : (\mathcal{A}_i, \boldsymbol{\mathcal{E}}) \to \mathbb{R} \tag{2.29}$$

  assigning significance to the particular agent (degree of the vertex in the graph, weighted degree, median degree of the neighbours, the modularity contribution - change of the modularity after removing the vertex).

- Community - a group of entities (vertices) with common properties, meaning, or cooperative influence. Radicchi et al. [RCC$^+$04] state that the community is part of a network where the internal connections are denser than the external ones. They provide two definitions of community, **strong** and **weak**.

  The **strong sense** means that the community is the subgraph $\mathcal{C}_s$ if

  $$\mathcal{A}_i^{in} < \mathcal{A}_i^{out},\ \forall \mathcal{A}_i \in \boldsymbol{\mathcal{A}}. \tag{2.30}$$

  The above definition says that each agent has more connections within the community than with the agents in other communities.

The **weak sense** means that the community is the subgraph $\mathcal{C}_w$ if

$$\sum_{\mathcal{A}_i \in \boldsymbol{\mathcal{A}}} \mathcal{A}_i^{in} > \sum_{\mathcal{A}_i \in \boldsymbol{\mathcal{A}}} \mathcal{A}_i^{out}. \tag{2.31}$$

The **weak sense** definition says that the sum of all degrees within the community is larger than the sum of degrees with the agents in other communities.

As mentioned by Radicchi et al., their definitions do not represent the only possible choice, and at the same time, they are not completely general. So it is not easy to formally define the concept of community. For that reason, we present here the definition, which is built on the idea of mapping between spaces. The formalisation is following:

$$d: \ \boldsymbol{\mathcal{A}} \times \boldsymbol{\mathcal{E}} \longrightarrow \boldsymbol{\mathcal{C}}, \tag{2.32}$$

where d is the mapping of the agent space and their interactions to the community space $\boldsymbol{\mathcal{C}}$. At the same time as the above-mentioned mapping, the objective function loss_d is optimised during community detection, which determines the quality of the dividing agents into communities according to a predefined criterion. From the above, the set of communities is formally defined as

$$\boldsymbol{\mathcal{C}} = \underset{d}{\text{argmin}}(\text{loss}_d(\mathcal{N}))(\boldsymbol{\mathcal{A}} \times \boldsymbol{\mathcal{E}}). \tag{2.33}$$

Based on the chosen representation of d, fundamentally different community detection outputs can be achieved. Mapping can enable or disable overlapping communities for which:

$$\exists \, \mathcal{C}_i, \mathcal{C}_j \in \boldsymbol{\mathcal{C}}: \ \mathcal{C}_i \cap \mathcal{C}_j \neq \emptyset, \tag{2.34}$$

or hierarchical-based, where

$$\exists \, \mathcal{C}_i, \mathcal{C}_j \in \boldsymbol{\mathcal{C}}: \ \mathcal{C}_i \subseteq \mathcal{C}_j. \tag{2.35}$$

As a ranking objective function loss_d, various methods can be used in network graphs. The connectivity inside communities could be described by density (the best connections are cliques, or the problem can be relaxed to dense subgraphs). It can also be described by the degree of vertices (communities of the same-degree neighbours or communities of one high-degree vertex and its low-degree neighbours) or by modularity maximisation. Based on the task, the point of interest can be centrality vertices, outliers, or bridge vertices (vertices that connect two or more communities).

2.2.2 Communities detection approaches

The fundamental problem of clustering on large-scale networks is the time complexity. Due to that, there is usually looked for some approximation and not precisely the optimal solution. There are three fundamental concepts of clustering on graphs: *divisive* algorithms (finds bridge connections between communities and remove them) by Girvan and Newman [GN02], *agglomerative* (merge similar communities from the one-agent-community to the more-agents) by Pons and Latapy [PL06] and the *optimization* (maximization or minimization of the objective function).

The most direct way to find communities in graphs and networks is to transform the problem into finding cliques and their close subgraphs. A clique of the graph (network) is the most robust community in terms of mutual connection. However, the fundamental disadvantage of such an idea is that finding a maximal clique in a graph is an NP-hard problem from the point of view of Complexity Theory. The decision variant of such a problem is NP-complete - it is one of Richard Karp's original 21 problems shown NP-complete [Kar72]. The **Bron-Kerbosch algorithm** [BK73] can be used to find the maximum cliques. It is an enumerative algorithm on undirected networks that sequentially searches all subsets of agents and checks whether the current subset is a clique. The fundamental disadvantage of the Bron-Kerbosch is the exponential time complexity of $\mathcal{O}(3^{N/3})$, where N is the number of vertices (agents). The asymptotic time complexity can be reduced by relaxing the problem from maximal cliques to any. For this, a greedy approach can be used based on the aggregation of agents so that the partial component is a clique. Finding a clique starting epoch of the algorithm from each agent by the greedy approach has an asymptotic complexity of $\mathcal{O}(N^2)$ (linear asymptotic time for each agent). The clique methods have got the disadvantage of finding relatively small communities. Searching for dense subgraphs (near clique) is presented by Khuller and Saha in [KS09]. They focus on developing fast polynomial time algorithms. Their work deals with variations algorithms in searching dense subgraphs for directed and undirected graphs. The **agglomerative Khuller and Saha greedy approaches** are shown in Algorithms 4 and 5. The advantage of near-clique methods is the polynomial complexity, but the algorithms do not guarantee an optimal solution.

Another possible approach to finding communities on graphs is the approach using **entropy-based clustering**. This method belongs to the methods using the optimization of the objective function. Le and Kim [LK15] applied that approach to find protein structures and protein-protein interactions. Their approach consisted of determining locally optimal clusters by aggregating from an initial cluster by combining selected seeds while minimising the cluster's entropy. The approach is described as follows. The entropy e of

Algorithm 4 Khuller and Saha: Densest-Subnetwork greedy

```
Input: $\mathcal{N}$.
Output: error matrix $\boldsymbol{E}_{avg} \in \mathbb{R}^{w \times f}$ (where $w$ is the size of the input window
step).

  $n \leftarrow |\boldsymbol{\mathcal{A}}|$
  $H_n \leftarrow \mathcal{N}$
  for $i$ = $n$ to 2 do
     Let $\mathcal{A}_i$ be an agent in $H_i$ of minimum degree.
     $H_{i-1} \leftarrow H_i \backslash \{\mathcal{A}_i\}$
  end for
  return $H_j$ which has the maximum density over all $H_i$, $i \in \{1, ..., n\}$.
```

Algorithm 5 Khuller and Saha: Densest-at-least-k

```
Input: $\mathcal{N}$, $k$.
Output: error matrix $\boldsymbol{E}_{avg} \in \mathbb{R}^{w \times f}$ (where $w$ is the size of the input window
step).

  $D_0 \leftarrow \emptyset$
  $N_0 \leftarrow \mathcal{N}$
  $i \leftarrow 1$
  while $V(D_i) < k$ do
     $H_i \leftarrow$ maximum-density-subgraph$(N_{i-1})$
     $D_i \leftarrow D_{i-1} \cup H_i$
     $N_i \leftarrow$ shrink$(N_{i-1}, H_i)$
     $i \leftarrow i + 1$
  end while
  for each $D_i$ do
     Add an arbitrary set of max($k$ - $\boldsymbol{\mathcal{A}}(D_i)$, 0) vertices to form $D_i^{'}$
  end for
  return $D_j^{'}$ which has the maximum density among all $D_i^{'}$.
```

a community $\mathcal{C}$ is defined as:

$$e(\mathcal{C}) = \sum_{\mathcal{A}_i \in \boldsymbol{\mathcal{A}}} -p_i(\mathcal{A}_i) log_2 p_i(\mathcal{A}_i) - (1 - p_i(\mathcal{A}_i)) log_2 (1 - p_i(\mathcal{A}_i)), \tag{2.36}$$

where

$$p_i(\mathcal{A}_i) = \frac{n_i}{m_i}, \tag{2.37}$$

where n_i is a count of neighbouring agents of the agent $\mathcal{A}_i$ inside the community $\mathcal{C}$ and m_i is the total count of the neighbouring agents of the agent $\mathcal{A}_i$. The Le and Kim Entropy-based clustering algorithm is performed in four steps:

1. Select two seeds as an initial community.

2. Add all neighbours if the adding will decrease community entropy.
3. Remove agents added in the previous step if the removing will decrease community entropy.
4. Repeat steps 2 and 3 until there is no choice in entropy decreasing. Shape the community, remove agents from the network and repeat the process from step one.

Zhao et al. [ZLW21] dealt with community detection using subgraph compression on large-scale social networks, which is a hierarchically agglomerative approach. The **CDEP** method consists of four parts.

1. Compressing - produces a compressed network obtained by iteratively merging agents with a degree 1 or 2 into their neighbours of a higher degree.
2. Seeding determination - defines density and quality of agents and computes the probability of agents being community seeds.
3. Expansion - iteratively expanding the community. Each community is initialized on agents designated as seeds. Agents are assigned to communities based on the calculation of community membership scores.
4. Propagation - the community results are propagated to the original network based on the compressing affiliation.

But the fundamental problem of this approach lies in the assumption that several almost isolated agents in the network have only one or two neighbours. The approach is therefore designed especially for large-scale social networks, where people can expect just such behavioural interactions. However, on strongly connected networks, this approach fails and produces only one community as an input network - the method has nothing to catch in the first step on strongly connected networks.

The **Louvain method** is a greedy approach for community detection on graphs with log-linear time complexity which is its main advantage. The method uses the principle of graph modularity optimization; it was first used in identifying language communities in a Belgia mobile network by Blondel et al. [BGLL08]. The method is used to detect non-overlapping communities, and its main advantage lies in the fact that it outperforms other methods in computation time.

The modularity is a rating of the quality of the communities partition. It is a scalar between -1 and 1, defined as

$$Q = \frac{1}{2m} \sum_{i,j} \left[A_{ij} - \tfrac{k_i k_j}{2m} \right] \delta(c_i, c_j), \tag{2.38}$$

where A_{ij} is a weight of edges between i and j agents of the network $\mathcal{N}$,

$$k_i = \sum_j A_{ij} \tag{2.39}$$

is the sum of the weights of the edges attached to the agent i, c_i is the community of the i^{th} agent, $\delta(a, b)$ is 1 if $u = v$ and 0 otherwise, m is the sum of all the weights of the edges. Maximising modularity improves the quality of dividing network agents into communities. According to Brandes et al., [BDG$^+$06], the disadvantage of direct maximization of modularity is the high time complexity - it is an NP-complete problem that cannot be solved in polynomial time. The Louvain method solves this problem by introducing a hierarchical approach to dividing agents into communities and calculating only the change in the modularity of the graph, not the entire value. The change in modularity is only affected by the calculation on the agent, which changes the community and its neighbours. The change in modularity when assigning the i^{th} agent to the community $\mathcal{C}_c$ is defined as:

$$\begin{aligned}\Delta Q = \left[\tfrac{\sum_{in} + k_{i,in}}{2m} - \left(\tfrac{\sum_{tot} + k_i}{2m}\right)^2\right] - \\ \left[\tfrac{\sum_{in}}{2m} - \left(\tfrac{\sum_{tot}}{2m}\right)^2 - \left(\tfrac{k_i}{2m}\right)^2\right],\end{aligned} \tag{2.40}$$

where $\sum_{in}$ is the sum of the edges weights within the community $\mathcal{C}_c$, the $\sum_{tot}$ is the sum of edges weights incident to the community $\mathcal{C}_c$, k_i is the edges weights sum incident to i^{th} agent, $k_{i,in}$ is the edges weights sum between i^{th} agent and agents inside the node $\mathcal{C}_c$ and m is the sum of all edge weights. The method works as described in Algorithm 6.

Algorithm 6 Louvain method

```
Input: $\mathcal{N}$.
Output: the tree graph of hierarchical communities clustering.

  for each each agent $\mathcal{A}_i$ do
    1. The neighbours $\mathcal{A}_j$ of $\mathcal{A}_i$ are considered.
    2. The gain of modularity for moving $\mathcal{A}_i$ to the community of $\mathcal{A}_j$.
    3. The agent $\mathcal{A}_i$ is placed into the community for which the gain is maximum,
       but only if the gain is positive.
  end for
  Build new network $\mathcal{N}'$ whose agents are communities found during the first phase
  and repeat the whole process until there is only one-super-agent-community.
```

The result of the Louvain method is the dendrogram - hierarchical tree representing a successive division of the agent network into communities.

White and Smyth in [WS05] presented a **spectral clustering approach** to finding communities in graphs. Their approach lies in optimizing the modularity function, which is reformulated as a spectral relaxation problem. The key idea is to turn the modularity function into a spectral problem in which the graph can be embedded into Euclidean space. On a graph represented in a Euclidean space, fast clustering algorithms such as K-mean, which is based on geometric principle, can then be used to identify the clusters. Compared to the Louvain method, the disadvantage of the spectral method is the time complexity $\mathcal{O}(n^3)$ during computing eigenvectors.

2.3 Conventional neural networks models

This subsection contains an overview of the conventional neural network models widely used in many typical machine learning tasks such as computer vision, sequence prediction, and regression or classification problems.

2.3.1 Dense NN

A fully connected (dense) neural network, also known as a multilayer perceptron (MLP as a deep learning model introduced by A. G. Ivakhnenko [ILLM67]), consists of layers in which every neuron of one layer is connected to every neuron in the following layer; thus, it is a feedforward neural network. Each neuron performs a linear combination of its input data followed by an activation function in a way:

$$\vec{y_i} = \vec{\alpha} \cdot \vec{x}_i + b_i, \tag{2.41}$$

where $\vec{y_i}$ is the output of the i^{th} layer, $\vec{\alpha}$ and b_i are trainable parameters of the corresponding neuron and $\vec{x}_i$ is the input to the i^{th} layer (output of the $(i-1)^{\text{th}}$ layer). The output of the last layer could be a scalar value, a probability distribution, or a multidimensional vector of class labels. Dense neural networks are useful for any arbitrary task of supervised machine learning. According to Danishvar et al., [DDS$^+$21], the main advantage of Dense NN is that there are no assumptions about specific structure, data handling and processing - the network is structure agnostic. Although fully connected networks are very broadly applicable and simply implementable, such networks tend to perform weaker than special-purpose networks built accordingly to the structure of a solved problem. Often the limitation of dense networks is a tendency to overfitting, especially if the amount of the trainable parameters (directly proportional to the count and dimensionality of the hidden layers) is large in relation to the size of the dataset.

The dense network schema shows Figure 2.2 generated by the NN-SVG tool [LeN19]. The example schema contains four layers - one input, two hidden, and one output.

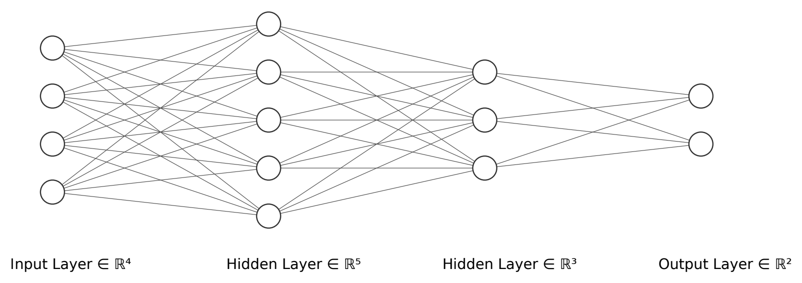


Figure 2.2: Example of Dense NN architecture

2.3.2 Convolutional neural network

Convolutional neural networks are a type of artificial neural network that is used primarily for pattern recognition and signal processing (Lo et al. [LCL+95]). Compared to fully connected networks, the CNN architecture solves the dimensionality problem of processed data samples. Each network layer processes the data flow by applying a convolutional mask to the blocks of data. The length of the step to move the mask to the next block of data is called a stride. The amount of masks used in parallel on the same layer is referred to as the dimensionality of the filter. The behaviour of convolution masks on the first layers of the model typically corresponds to edge detectors; the deeper layers are usually responsible for detecting abstract data patterns and capturing different aspects of data features. The output of the last convolutional layer is passed into the chain of a few fully connected layers (or recurrent layers) that process the learnt features or newly generated hidden features. According to Kiranyaz et al., [KAA+19], CNN is essential for segmentation and detection in image processing, speech processing and synthesis, and thus time-dependent series in general.

Although convolutions are mainly used in computer vision applications, they are also effective for classification or regression tasks in processing time-dependent data. The architecture of such an application is based on one-dimensional convolutions. The main challenge of time-dependent data processing is capturing the temporal dependencies in the sequences, which can be handled by properly specifying kernel mask size. The one-dimensional kernel mask filter is applied along the time dimension of the data sequence. Long-size masks are designed to capture long-term and global trends, and smaller masks, especially in hidden layers, capture short-term feature dependencies.

The general schema of the 1D convolution chain is shown in the picture 2.3.

The main components of the convolutional network architecture are as follows:

- **Convolutional layer** applies a kernel mask (convolutional tensor) on an input

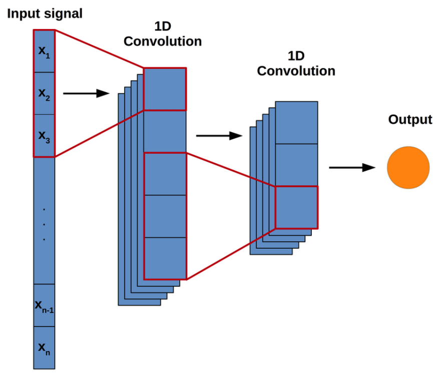


Figure 2.3: General 1D-conv chain [SH20]

tensor in a way (1D convolution):

$$g(x, a, b) = \text{bias}(x) + \sum_{dx=1}^{a} \mu(dx) \cdot f(x + dx), \tag{2.42}$$

where $f(x, y)$ is an input feature map, $\mu(x)$ is a kernel mask of size (a).

- **Activation function** adds a nonlinearity into the network.

According to Hosseini et al., [HP17], the fundamental problem with convolutional neural networks is the adversarial problem and generally poor noise immunity. Furthermore, there is a significant problem when dealing with irregularly sampled input data of different lengths.

2.3.3 Recurrent and LSTM neural networks

Recurrent neural networks (RNN) are types of artificial neural networks mainly used for pattern recognition in sequences, regression and encoder-decoder problems of time-dependent data. Compared to feedforward networks (fully-connected, convolutional), whose principle of data processing is a uniform flow of information, RNNs contain loops in their architecture. According to Schmidt [Sch19], the existence of loops within the architecture allows the network to remember contextual information about the flow of data and to make decisions based on aggregated information from (temporally) previous inputs and not just from the one currently being processed. The formal expression of processing the context is as follows:

$$\begin{aligned} \vec{h}_t &= \phi_1(\boldsymbol{U}\vec{x}_t + \boldsymbol{W}\vec{h}_{t-1} + \vec{b}_h), \\ \vec{y}_t &= \phi_2(\boldsymbol{V}\vec{h}_t + \vec{b}_y), \end{aligned} \tag{2.43}$$

where $\vec{h}_t$ is the contextual vector of the hidden state at step t, $\boldsymbol{U}$, $\boldsymbol{V}$, $\boldsymbol{W}$, $\vec{b}_h$ and $\vec{b}_y$ are coefficients that are shared temporally and ϕ_i are activation functions. Amidi [Ami19] states that the main advantage of recurrent neural networks is the possibility of processing input of any length, while the number of trainable parameters of the model does not increase with the increasing range of the processed sample. However, the implementation of the RNN network architecture itself, as it was defined, has the disadvantage of forgetting information about the context of the sequence. If an event at its beginning characterises a long sequence, it will be forgotten during processing and this information will not be transferred to the network output. The general phenomena that are often encountered in the context of recurrent neural networks are vanishing/exploding the gradient[2].

The Long short-term memory, introduced by Hochreiter & Schmidhuber, 1997 [HS97], are recurrent models described by Chien et al. [CTB+21], that exploit gating mechanisms to support the retention of information for longer inputs. LSTM cells deal with the vanishing gradient problem and improve the recurrent neural networks concept, and have the ability to train and remember long addictions of information sequences. The neural network is built with a chained repeating cell, as is visible in Figure 2.4. The LSTM cell

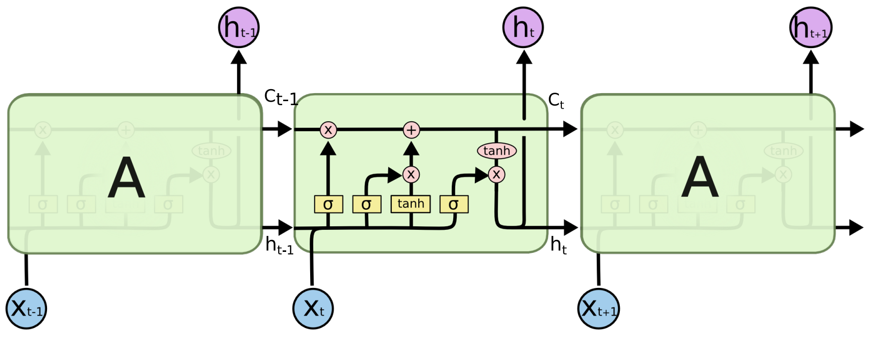


Figure 2.4: LSTM chain [Ola15].

has three entry instances: previous internal cell state vector $\vec{c}_{t-1}$, previous output vector $\vec{h}_{t-1}$ and actual input vector $\vec{x}_t$. Two resulting instances[3] are the current internal cell state vector $\vec{c}_t$ and the current output vector $\vec{h}_t$.

The first sigmoid on the left of the cell diagram 2.4 is called the *forget gate layer*, which decides how the previous internal state will be treated in the current cell. The second sigmoid from the left and the hyperbolic tangent block represent the *input gate*

[2]According to the Pascanu et al. [PMB13], if the gradient becomes too small and almost zero, it does not occur to update the early layers. It leads to a state where a model is only learning the last layers of the network - parameters (of the last few layers) are adapting to the training data. The problem typically occurs while using hyperbolic tangent activation functions. On the contrary, the gradient explosion occurs when the error gradients are accumulated and result in very large gradients. According to J. Brownlee [Bro19], the explosion can result in an unstable network that cannot learn from training data.

[3]For all experimental settings of models containing LSTM layers, the initial context cell state vector c_0 was experimentally set to the zero vector, and after the evaluation of the layer, the context vector was reset.

layer through which the cell's internal state is updated. The right part of the cell with the last sigmoid acts as an output gate layer for calculating the output vector $\vec{h}_t$. According to Salem [Sal18], the result depends on the modified cell internal state vector and the previous inputs.

The LSTM equations according to Figure 2.4 are following:

$$\begin{aligned}
\vec{p}_t &= \sigma(\vec{x}_t U^p + \vec{h}_{t-1} W^p + \vec{b}^p)\\
\vec{q}_t &= \sigma(\vec{x}_t U^q + \vec{h}_{t-1} W^q + \vec{b}^q)\\
\vec{r}_t &= \tanh(\vec{x}_t U^r + \vec{h}_{t-1} W^r + \vec{b}^r)\\
\vec{s}_t &= \sigma(\vec{x}_t U^s + \vec{h}_{t-1} W^s + \vec{b}^s)\\
\vec{C}_t &= \sigma(\vec{p}_t \odot \vec{C}_{t-1} + \vec{q}_t \odot \vec{r}_t)\\
\vec{h}_t &= \tanh(\vec{C}_t) \odot \vec{s}_t
\end{aligned} \tag{2.44}$$

where $\vec{b}$ are bias vectors, W are weights matrices and $\odot$ is a *element-wise (Hadamard) product.*

According to Chien et al., [CTB$^+$21], the advantage of networks based on LSTM cells is the ability to work with time-dependent data and resistance to vanishing gradient (unlike RNN and convolutional neural networks). The disadvantage is mainly the mechanism of forgetting, which results in an exponential loss of information over time and a limitation of the ability to capture information in the long term.

A possible solution to the aforementioned shortcoming of the LSTM network is to consider as output not only the last output vector h_{last}, but all (or at least a certain part) of the vectors h_t. The h_{last} contains condensed information, but, in contrast, considering more h_t vectors enables us to pay attention to the previous states of the sequence. The attention mechanism, by Bahdanau et al. [BCB15], enables the network to learn which hidden cell state attends to information and how much. Figure 2.5 shows the difference between the usage of the vanilla encoder-decoder of the RNN / LSTM architecture, where there is only one connection between layers (which is subject to forgetting) and the architecture of the attention mechanism where the layers are inserted the dense layer that enables the transfer of information from all hidden states.

In addition to the attention mechanism, there is another improvement of the LSTM-based networks - bidirectional LSTM (BLSTM) described by Cui et al. [CKPW20] and [CKW18]. In such cases, the layer is exploited to capture spatial features and bidirectional temporal dependencies from historical data. The LSTM block processes the data in a forward direction and backwards by another block. The outputs of both directions are

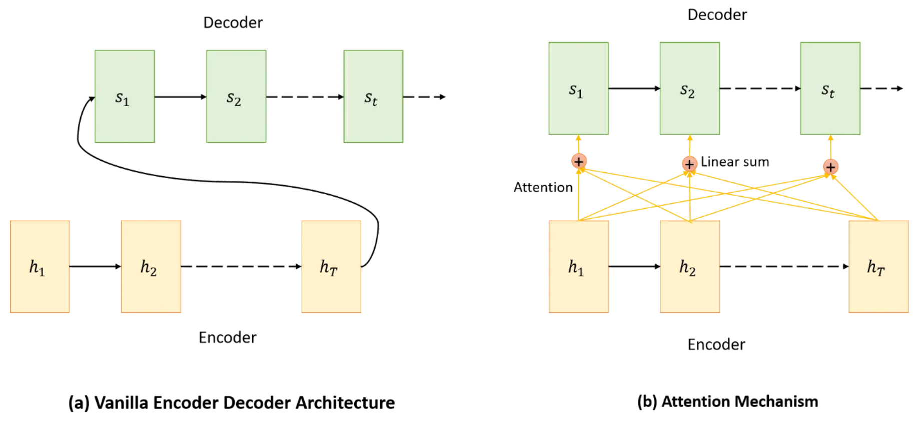


Figure 2.5: Vanilla and attention encoder-decoder architectures [Man21]

merged or aggregated. The formal notation of the bidirectional LSTM layer is as follows:

$$\texttt{blstm}(\boldsymbol{x}, f) = f(\sum_{i=1}^{n} l_1(\vec{x}_i), \sum_{i=1}^{n} l_2(\vec{x}_{n-i})), \tag{2.45}$$

where l_1 and l_2 are LSTM cells, f is a merging-aggregation function and

$$\boldsymbol{x} \in \mathbb{R}^{n \times m}, \tag{2.46}$$

where n is the time series dimension, m is a feature dimension, and

$$\vec{x}_i \in \mathbb{R}^m. \tag{2.47}$$

The BLSTM architecture is shown in the picture 2.6.

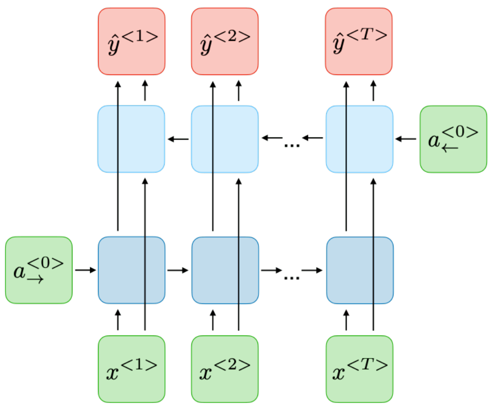
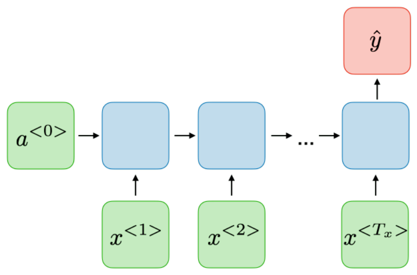


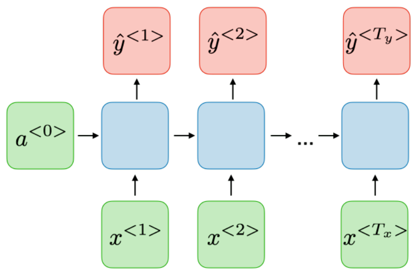
Figure 2.6: BLSTM architecture, image taken from [Ami19]

Figure 2.7 shows typical topological architectures of recurrent neural networks.

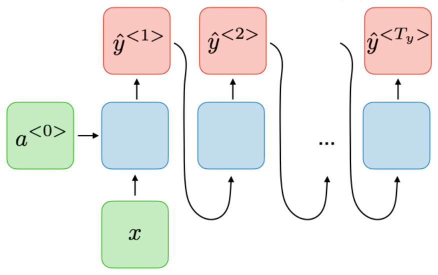
- **many-to-one** used for sequence classification, but subject to the forgetting problem mentioned above.



(a) Many-to-one architecture



(b) Many-to-many architecture



(c) One-to-many architecture

Figure 2.7: LSTM networks architectures, images taken from [Ami19]

- **many-to-many** used for recognition or classification and is prepared for using the attention mechanism to solve the above-mentioned forgetting problem.

- **one-to-many** used for sequence generation or prediction.

2.3.4 Attention neural network

Attention is a technique to reduce the computational complexity of the neural network model by introducing a block that focuses on specific features or regions of the input data rather than uniformly covering the entire sample. The mechanism was introduced by Bahdanau et al. [BCB15] in a machine translation model. The technique is used in different fields of machine learning, such as image recognition, natural language processing, or time-series processing. The following formulation can be introduced for the attention mechanism. The input sample is called a sentence of words. Each word is characterized by its letters. Attention mapping highlights letters, and thus words, which are essential for the solved task. Non-important, irrelevant or noisy words are assigned low-ranked weights, and thus values of their letters are lowered. Ideally, irrelevant words are zeroed by the zero weights. Important words have assigned high-rank weights, which increase their letter values. The setting of the weights is realized by gradient descent in the same way as in other parts of the neural network.

The Bahdanau et al.'s attention mechanism consists of three steps:

1. Alignment score calculation. The actual score $e_{t,i}$ is calculated by the model m based on the previous output s_{t-1} and the current hidden state h_i in a way

$$e_{t,i} = m(s_{t-1}, h_i). \tag{2.48}$$

2. The weights are computed by the softmax function from the scores.

$$\alpha_{t,i} = \text{softmax}(e_{t,i}) \tag{2.49}$$

3. The context vector c_t is a scalar product of weights and hidden states.

$$c_t = \sum_i \alpha_{t,i} h_i \tag{2.50}$$

The above attention mechanism corresponds to recurrent (LSTM) models in returning all hidden vectors and their processing with a dense layer.

The general attention mechanism was introduced by the GoogleBrain team [VSP$^+$17], the concept is used in the transformer neural network blocks. The general mechanism is described by D. Soydaner [Soy22] and Brauwers and Frasincar [BF23]. In the general approach, the input data are divided into three components, queries Q, keys K, and values V. The attention weights corresponding to the data features are computed from these components. The queries are generalisations of the vector s_{t-1} in the Bahdanau approach; the values and keys are related to the vectors h_i. Keys represent the most relevant features for computing the attention weights, queries correspond to the features that are being attended to, and values are actual attendee information. The steps of the general attention mechanism are the following:

1. Query vectors are matches against keys. Scores are scalar products of queries and keys.

$$e_{q,k_i} = q \cdot k_i \tag{2.51}$$

2. The weights are computed by the softmax function from the scores.

$$\alpha_{t,i} = \text{softmax}(e_{t,i}) \tag{2.52}$$

3. The generalised attention is a weighted sum of the value vectors with the corresponding keys.

$$\text{attention}(q, \boldsymbol{K}, \boldsymbol{V}) = \sum_i \alpha_{q,k_i} v_{k_i} \tag{2.53}$$

The approach allows the neural network to better target the direction of the gradient to those features that carry information from the point of view of the given issue. Also, because the approach can be applied to a wide range of problems, it is very flexible. A significant advantage of the attention mechanism is its direct explainability. The obtained attention weights can be visualised to provide insight into how the model is processing the input data and which features are relevant - the features of the input data with assigned attention weight around zero represent noisy/irrelevant information.

Chapter 3

Methods, proposed approaches and results

This part provides an overview of the methods used. The neural network models and algorithms for their explainability are listed with detailed summaries to facilitate replication of the experimental evaluations.

As mentioned in the introduction, the main task solved is to estimate four parameters characterizing the quality of the material from the curve of dependence of load on a deflection. This task was divided into two subtasks.

In the first subtask, called *partial curve problem*, only the first twenty points (of the total sixty-one) of the curve are considered as the input sample. Figure 3.1 shows the problem schema. The twenty points enter the *c2c* (curve to curve) neural network model, which calculates the remaining 41 points. This model generates the rest of the curve, which enters the main model, which determines the four material parameters. The main model and *c2c* model were trained separately. The reason for the cutting of the curve at the twentieth point is the fact that the breakpoints (peaks) occur only after this boundary, and such an approach allows use without knowing exactly the breakpoint. So it is a question of determining the parameters of the material without its deformation occurring.

The second subtask, called *full curve problem*, considers the input sample to be a complete curve of all sixty-one points, which enters the main model that produces the material parameters. This approach considers information about the breaking point and the subsequent behaviour of the material after breaking but requires deformation of the object, which can be problematic and expensive in industrial practice.

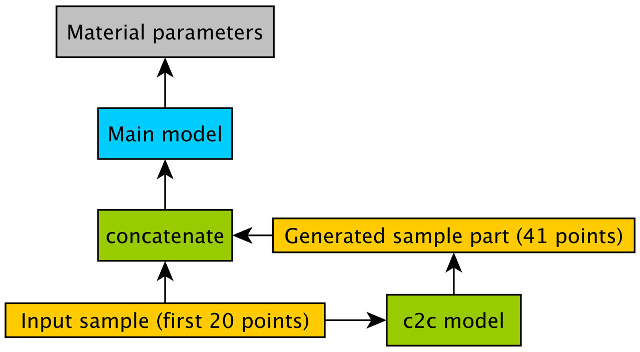

Material parameters
Main model
concatenate
Generated sample part (41 points)
Input sample (first 20 points)
c2c model

Figure 3.1: Schema of the *partial curve problem*.

3.1 Used conventional models

This subsection contains summaries of the used conventional neural network models. The first of them, the *c2c*, is a model used to generate the full curve in the *partial curve problem*. The rest are models used to solve the problem of material parameters in both *partial curve problem* and *full curve problem*.

3.1.1 c2c model

A *c2c* (curve to curve) model based on LSTM and Dense layers was used to generate the surrogate curve. At the input of the model was the vector of the first twenty points of the curve, and the model added the next forty-one points to the complete curve of the given sample. The model contained 101 173 trainable parameters. The specific layers of the chain were as follows:

1. LSTM layer with an attention mechanism,
2. dense layer: input dimension 20, output dimension 200,
3. LSTM layer with an attention mechanism,
4. dense layer: input dimension 200, output dimension 400,
5. LSTM layer with an attention mechanism,
6. dense layer: input dimension 400, output dimension 41.

3.1.2 CNN

The built convolutional neural network consisted of five convolutional layers (with linear activation function), one flattened layer and one fully connected dense layer. The network contained 23 904 trainable parameters. The specific layers of the chain were as follows:

1. convolutional layer: mask dimension 30, filters (2, 10),
2. convolutional layer: mask dimension 15, filters (10, 20),
3. convolutional layer: mask dimension 10, filters (20, 30),
4. convolutional layer: mask dimension 5, filters (30, 40),
5. convolutional layer: mask dimension 5, filters (40, 40).
6. flatten layer.
7. dense layer: input dimension 40, output dimension 4.

The LeNet schema of the network is in the picture 3.2.

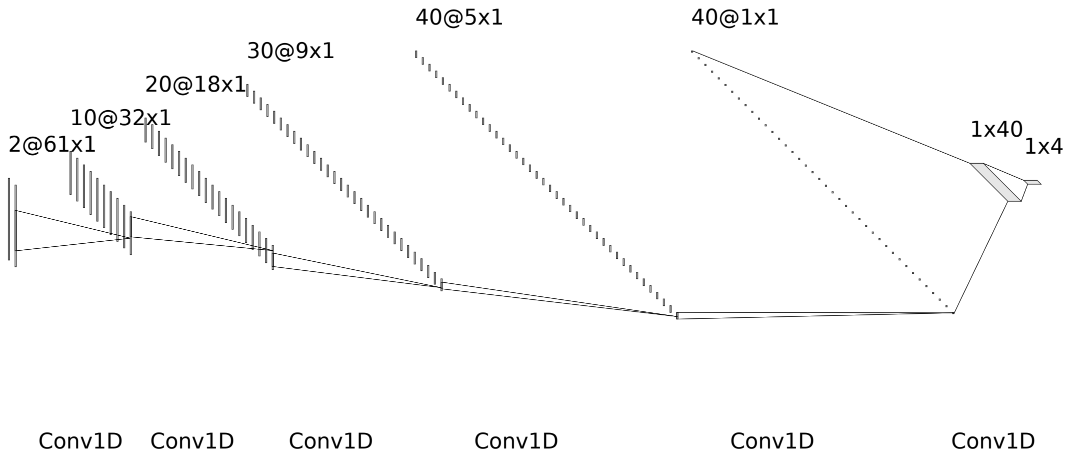


Figure 3.2: Used CNN, the image generated by the NN-SVG tool [LeN19]

3.1.3 Dense NN

The built Dense neural network consisted of five layers. The model contained 71 204 trainable parameters. The specific layers of the chain were as follows:

1. flatten layer.
2. dense layer: input dimension 122, output dimension 300,
3. dense layer: input dimension 300, output dimension 100,
4. dense layer: input dimension 100, output dimension 40,
5. dense layer: input dimension 40, output dimension 4.

3.1.4 LSTM NN

The built LSTM-based neural network consisted of six layers. The element-wise summation was used as the merging-aggregation function in the BLSTM layers. The model contained 6 762 trainable parameters. The specific layers of the chain were as follows:

1. dense layer: input dimension 61, output dimension 61.
2. bidirectional LSTM layer with an attention mechanism.
3. dense layer: input dimension 61, output dimension 40.
4. bidirectional LSTM layer with an attention mechanism.
5. flatten layer.
6. dense layer: input dimension 80, output dimension 4.

3.2 Proposed approaches

This part contains an overview of the proposed approaches of neural network models and explainability.

3.2.1 Proposed models

The goal was to design models of neural networks, which with their architecture, will reflect the distribution of information in the data, processing input samples in parts carrying common information. For that reason, the first step was to build a similarity (unoriented, weighted) graph - the vertices of such a graph represent the points of the input curve and the weights of the edges of the measure carrying information together.

The similarity graph was constructed based on the Dynamic time warping[1] of the curves in the dataset in the following way. Let

$$\begin{aligned}\vec{a} &:= (a_1, a_2, ..., a_N),\\ \vec{b} &:= (b_1, b_2, ..., b_N)\end{aligned} \tag{3.1}$$

[1]**Dynamic time warping** is an algorithm to measure the similarity between two time-dependent sequences which differ in speed. The algorithm dates back to the 1960s; for the purposes of this work, a review by P. Senin was used [Sen09]. Dynamic time warping (DTW) calculates optimal mapping (match) between sequences with the following constraints:

- Every point in each sequence must be matched with at least one point from the other sequence.
- The mapping between sequences points is monotonically increasing.
- First points in sequences must be mutually matched.
- Last points in sequences must be mutually matched.

be (curves) samples of length N. The warping path denoting the mapping of points between curves is a sequence of pairs $P = (p_1, ..., p_L)$. The pair $p_i = (p_{i,1}, p_{i,2})$ means that the $p_{i,1}^{\text{th}}$ point of the curve $\vec{a}$ is mapped to the $p_{i,2}^{\text{th}}$ point of the curve $\vec{b}$. The quality of the DTW path is given by optimising (minimising) the aggregated sum of Euclidean distances between mutually mapped points[2]. Define the mapping matrix $\boldsymbol{D}^{a,b}$ in the following way:

$$\boldsymbol{D}_{i,j}^{\vec{a},\vec{b}} = \begin{cases} 1, & \text{if } \vec{a}_i \text{ is mapped to } \vec{b}_j \\ 0, & \text{otherwise.} \end{cases} \tag{3.2}$$

The weights matrix $\boldsymbol{W}$ of the similarity graph is obtained as

$$\boldsymbol{W} = \sum_{\vec{a} \in \mathcal{X}} \sum_{\substack{\vec{b} \in \mathcal{X} \\ \vec{a} \neq \vec{b}}} \boldsymbol{D}^{\vec{a},\vec{b}}, \tag{3.3}$$

where $\mathcal{X}$ is set of samples and n is the number of points in the input sample ($n = 61$).

On the obtained similarity graph, communities were detected using the Louvain method. Figure 3.3 shows the log-scale similarities of points in curves[3]. On the diagonal of the heatmap, there is a line with black points (please notice that the line is not continuous and spaces are highlighted by green vertical lines). The segments of such line show points belonging to common communities. The first ten points of the curves were evaluated as ten isolated communities by the DTW-Louvain methods. Since such communities would have one point each, they were grouped into one community. It can be seen from the heatmap that the division into communities is densest between the twentieth and fortieth points, where the most breaking (peak) points are also located.

[2]Considering that these are the distances of points in space, where the base vectors correspond to different physical quantities (and therefore also units), the values of the coordinates of the points were normalized by components.

[3]Similarity of two points in given by the weight of the edge between corresponding vertices of the similarity graph.

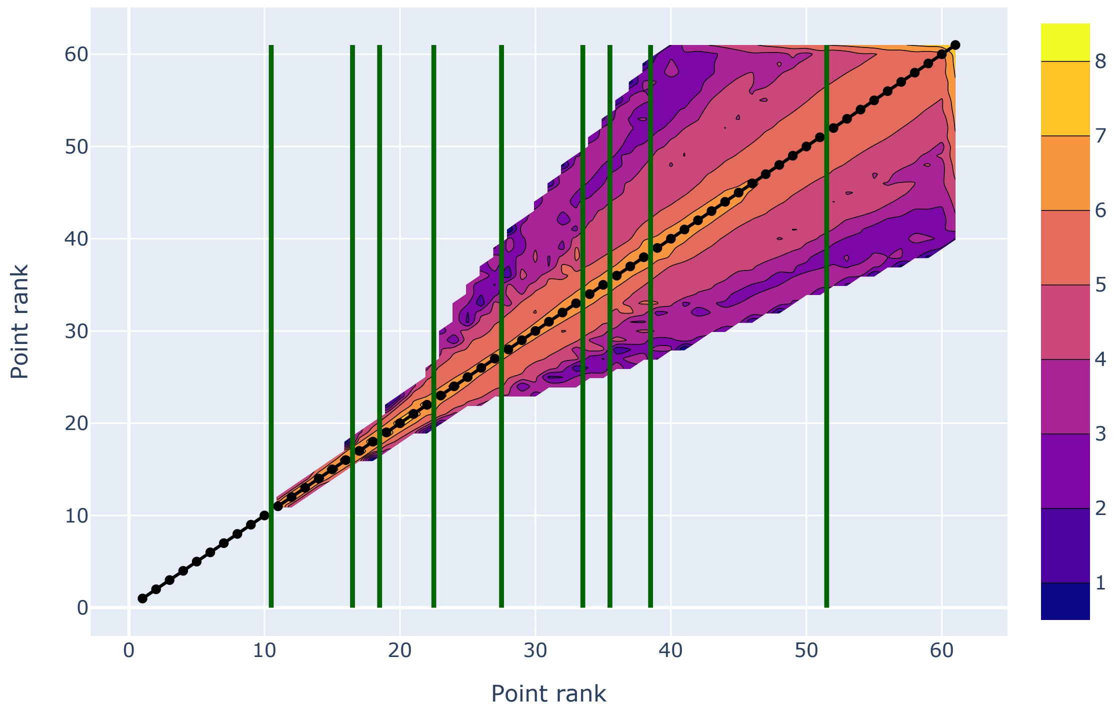


Figure 3.3: Similarity contour heatmap (natural logarithm scale) on A-dataset. The axes represent the order of the points on the curves.

Three architectures of neural network models were created based on the distribution of sample points. Their architecture was implemented in the tool Maen (Multiple agents ecosystem network) developed by V. Drahý [Dra23].

In the following diagrams of the neural network model, the input sample's orientation (relative to the order of the curve points) is indicated by a dashed arrow.

Figure 3.4 shows the topology schema of the **Feedforward Maen** (**F-Maen**) neural network model. The model is composed of the following components layers:

1. *InputAgents* components marked by yellow color 🟨 - components that represent input data divided into communities. The InputAgents transform two-dimensional points of the input sample into vectors of complex numbers.

2. *HiddenAgents* components marked by orange colour 🟧 - perform geometrical transformations by applying dense layers with complex numbers parameters.

3. *HiddenAgents* components marked by blue colour 🟦 - vertically concatenate the results of two previous neighbouring orange Hidden Agents to pass the information of neighbouring communities forward and apply a dense layer with complex numbers

parameters. Based on this, the blue layer considers not only isolated community information but also the context of the immediate surroundings of the communities. At the same time, within the network, all information between components is forwarded between layers (layers marked with colours). There is no information transfer within a single layer. For that, the network is called Feedforward Maen.

4. *HiddenAgents* components marked by green colour 🟩 - perform concatenation of results of the blue layer from the vector of complex numbers into a vector of real numbers. Next, apply three dense layers with real numbers parameters (output dimension is a dimension of labels).

5. *OutputAgent* marked by red colour 🟥 concatenates results of the green layer components and performs one single dense layer resulting in a dimension of labels.

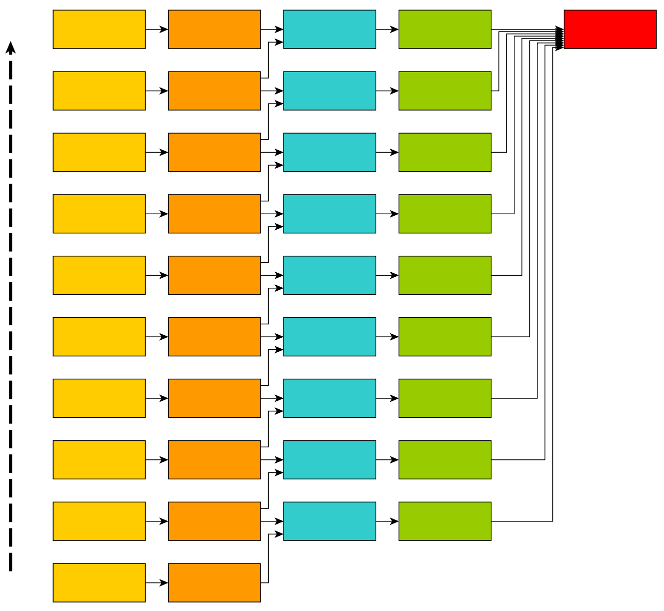


Figure 3.4: Feedforward Maen (F-Maen).

Figure 3.5 shows the topology schema of the **LSTM Maen** (**L-Maen**) neural network model. The model is composed of the following components layers:

1. *InputAgents* components marked by yellow color 🟨 - components that represent input data divided into communities.

2. *HiddenAgents* components marked by orange colour 🟧 - contain bidirectional LSTM cells to process time dependencies within each community.

3. *HiddenAgents* components marked by blue colour 🟦 - the results of three adjacent components from the orange layer are concatenated and processed by one dense

layer in the HiddenAgents components of the blue layer. Based on this, the blue layer considers the extended surroundings of the information processed in the orange layer.

4. *HiddenAgents* components marked by green colour 🟩 - components containing bidirectional LSTM cells to process time dependencies of newly generated hidden features from blue layer components (output dimension is a dimension of labels).

5. *OutputAgent* marked by red colour 🟥 concatenates results of the green layer components and performs one single dense layer resulting in a dimension of labels.

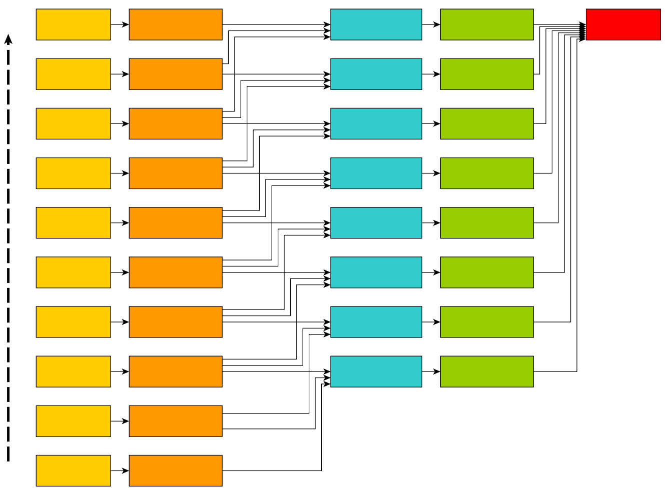


Figure 3.5: LSMT Maen (L-Maen).

Figure 3.6 shows the topology schema of the **LSTM-Attention Maen** (**L-A-Maen**) neural network model. The model is composed of the following components layers:

1. *InputAgents* components marked by yellow color 🟨 - components that represent input data divided into communities.

2. *HiddenAgents* components marked by purple color 🟪 - components with a general attention mechanism. Their goal is to highlight parts of the data according to the focus of the attention module.

3. *HiddenAgents* components marked by green colour 🟩 - contain bidirectional LSTM cells to process time dependencies within each community. Furthermore, these components transmit to each other (in the direction of the order of the points of the input curve) the context vector of LSTM cells. Thus, time-dependent information is processed within each component, and time-dependent context is passed between components.

4. *HiddenAgents* components marked by orange colour 🟧 - their role is identical to that of the green colour components, with the only difference being that they move in the opposite (relative to the order of the input curve points) direction.

5. *HiddenAgents* components marked by blue colour 🟦 - perform concatenation of results of green and orange corresponding components and applies dense layer (output dimension is a dimension of labels).

6. *OutputAgent* marked by red colour 🟥 concatenates results of the blue layer components and performs one single dense layer resulting in a dimension of labels.

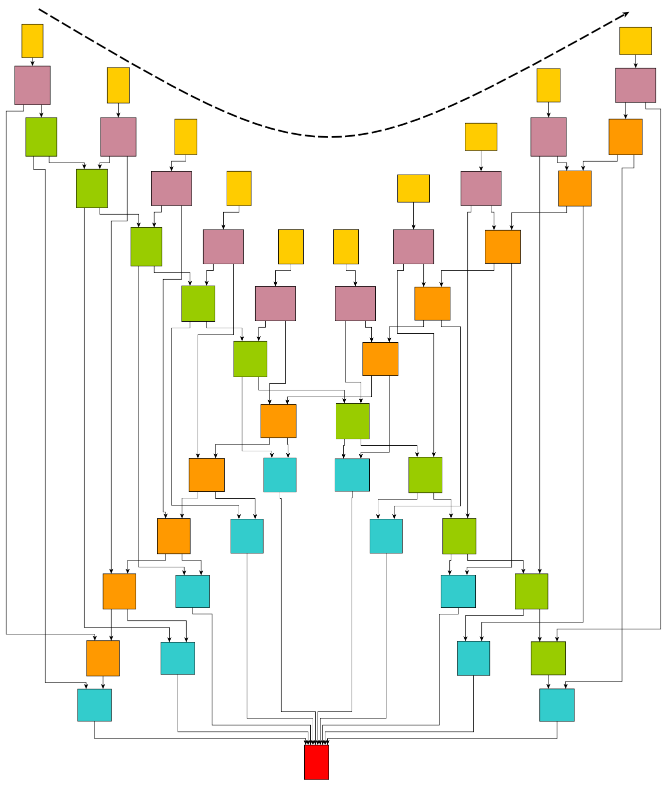


Figure 3.6: LSMT-Attention Maen (L-A-Maen).

3.2.2 Proposed explainability methods

Two approaches are presented here to determine the usefulness of neural network model components. The first one is based on monitoring the gradient during model training. The second approach identifies statistically significant correlations between component outputs on the one hand and expected labels on the other.

Tracking gradient is based on the idea that the optimizer searches the space intending to get the parameter to the optimum value as the network learns and shrinks the gradient. Thus, the gradient with respect to the trainable parameter typically decreases exponentially. If the parameter fails to learn, its value oscillates. For non-converging parameters, gradient values are not reduced. Formally expressed as follows.

$$f := \texttt{loss}(\texttt{model}(\vec{x}, \vec{p})) \tag{3.4}$$

The f is defined as a result of the loss function of the neural network model with input vector $\vec{x}$ and vector of trainable parameters of the model $\vec{p}$. The sequence s_g is obtained by following the gradient with respect to j^{th} parameter during model training. The sequence s_p by following parameter changes generates.

$$\begin{aligned} s_g &:= \left(\frac{\partial f_i}{\partial p_j}\right)_{i=1}^{I}, \\ s_p &:= \Big(|p_{j,i-1} - p_{j,i}|\Big)_{i=2}^{I}, \end{aligned} \tag{3.5}$$

where I is the number of iterations while training, $p_{j,i}$ is a value of j^{th} parameter in i^{th} iteration of training. Both sequences should converge to zero when training the given parameter correctly. Suppose they do not converge, oscillate, or settle to a non-zero value. In that case, it means that the optimizer cannot find the correct value of the given parameter - this implies that the given parameter is confusing for the model.

Identifying statistically significant correlations between component outputs and expected labels is based on the idea that the outputs of the relevant components (neurons) of the model should carry information with an influence on the model's outputs, and thus the component's outputs are correlated with corresponding labels. Depending on the significance level of the statistical correlation test, the threshold of how much the given component is related to the label can be changed. The t-test (Pearson's correlation coefficient follows Student's t-distribution; Rahman, N.A. [Rah68]) determines the statistical significance of the correlation between two signals.

Chapter 4

Experimental results and evaluations

4.1 Experimental results of neural networks

In the experimental setting, the *mean square error* was always set as a loss function for training neural networks, as the Adam optimizer with a preset learning rate of 0.001 and a decay of momentums of 0.9 and 0.999. The median relative error was determined as a measure of quality.

Due to the fact that the experiments took place on three different (especially in terms of the number of samples) datasets and the compared models differed greatly in the number of trainable parameters, the efficiency of the parameter p_e) was introduced to compare the performance of the networks in the following way:

$$p_e = \frac{1}{n \cdot m}, \tag{4.1}$$

where n is the count of trainable parameters of the model and m is the median relative error.

Training and evaluation of the models were carried out by cross-validation in five epochs. The distribution of the dataset in each epoch was 64% samples for training, 16% for validation and 20% for testing, with no overlap in test data between epochs. The models were learned on the training data after epochs, and the final set of trainable parameters was chosen for the iteration with the best model quality on the validation data; test data was evaluated for this parameter setting.

From the results (median relative errors) presented in Tables 4.1 to 4.12, it can be seen that lower median relative errors are achieved on all datasets when considering the complete curve as an input sample, which means that the neural networks consider information about the peak breaking point of the monitored object to be important. Furthermore, it can be concluded from such a finding that the subsequent behaviour of the object after breaking is also essential for the decision-making of the models - in general, two behaviour scenarios occur - complete deformation and a rapid decrease in the observed load or partial deformation with a slow decrease in the load, when the material is still able to withstand a certain pressure. According to the obtained results, it can be said that the absence of accurate information about this behaviour worsens the results of neural networks.

The results show that the parameter F_c is determined as the worst, while E_c is the best. Therefore, it can be said that the value of F_c does not share the informational connection with the input data, regardless of the neural network model, dataset, and number of samples.

As the number of samples in the datasets increased, the quality of the model results improved significantly. It, therefore, makes sense to create large datasets for the given task, which currently do not exist, as the systematic generation of datasets is very computationally intensive, and real physical measurement is very expensive due to the fact that the measured objects must be destroyed.

The required maximum 20% relative error was achieved on the A-dataset (in the test data) only for the parameter E_c, on the B-dataset as well as the C-dataset for the parameters E_c, F_t and G_f.

Furthermore, it can be observed from the results that for a small number of samples, it is worthwhile to create a specific, complex topology of a neural network, which, even on small data, can set the parameters for solving the problem. This can be observed in the test results of *L-A-Maen* on the A-dataset. In contrast, a neural network architecture with a large number of trainable parameters and a topologically simple structure cannot absorb the problem and generalize it on small data, but with the growing number of samples in the dataset, the behaviour of the large-scale network improves and the quality exceeds that of small, topologically special networks, which can be observed on the *Dense NN* results.

	CNN	Dense NN	LSTM NN	L-Maen	L-A-Maen	F-Maen
E_c	0.148	**0.087**	0.144	0.126	0.114	0.120
F_c	0.314	0.346	0.287	0.273	**0.232**	0.296
F_t	0.234	**0.198**	0.236	0.221	0.216	0.209
G_f	0.241	0.226	0.250	0.242	**0.209**	0.245
Mean	0.234	0.214	0.229	0.216	**0.193**	0.218
Model params count	23904	71204	6762	12308	2600	33028
p_e	0.018	0.007	0.065	0.038	**0.200**	0.014

Table 4.1: Partial curve: training (A-dataset)

	CNN	Dense NN	LSTM NN	L-Maen	L-A-Maen	F-Maen
E_c	0.167	0.131	0.176	0.171	0.149	**0.128**
F_c	0.380	**0.326**	0.379	0.380	0.394	0.353
F_t	0.325	**0.267**	0.302	0.329	0.292	0.275
G_f	0.357	0.364	0.357	**0.318**	0.362	0.326
Mean	0.307	0.272	0.304	0.300	0.299	**0.271**
Model params count	23904	71204	6762	12308	2600	33028
p_e	0.014	0.005	0.049	0.027	**0.129**	0.011

Table 4.2: Partial curve: test (A-dataset)

	CNN	Dense NN	LSTM NN	L-Maen	L-A-Maen	F-Maen
E_c	0.203	**0.082**	0.166	0.128	0.136	0.156
F_c	0.373	0.274	0.337	**0.205**	0.254	0.355
F_t	0.237	**0.144**	0.210	0.175	0.199	0.249
G_f	0.188	**0.144**	0.164	0.158	0.161	0.311
Mean	0.250	**0.161**	0.219	0.166	0.188	0.268
Model params count	23904	71204	6762	12308	2600	33028
p_e	0.017	0.009	0.067	0.049	**0.205**	0.011

Table 4.3: Full curve: training (A-dataset)

	CNN	Dense NN	LSTM NN	L-Maen	L-A-Maen	F-Maen
E_c	0.221	0.169	0.187	**0.136**	0.152	0.177
F_c	0.480	0.424	0.394	0.326	**0.272**	0.541
F_t	0.378	0.348	0.264	0.317	**0.207**	0.271
G_f	0.238	0.251	0.237	0.219	**0.204**	0.334
Mean	0.329	0.298	0.271	0.250	**0.209**	0.331
Model params count	23904	71204	6762	12308	2600	33028
p_e	0.013	0.005	0.055	0.033	**0.184**	0.009

Table 4.4: Full curve: test (A-dataset)

	CNN	Dense NN	LSTM NN	L-Maen	L-A-Maen	F-Maen
E_c	0.123	**0.052**	0.129	0.133	0.133	0.054
F_c	0.332	0.321	0.326	**0.302**	0.325	0.314
F_t	0.234	0.169	0.203	0.289	0.272	**0.147**
G_f	0.237	0.153	0.220	0.251	0.255	**0.151**
Mean	0.231	0.174	0.220	0.244	0.246	**0.166**
Model params count	23904	71204	6762	12308	2600	33028
p_e	0.018	0.008	0.067	0.033	**0.156**	0.018

Table 4.5: Partial curve: training (B-dataset)

	CNN	Dense NN	LSTM NN	L-Maen	L-A-Maen	F-Maen
E_c	0.122	**0.055**	0.116	0.139	0.142	0.056
F_c	0.333	0.322	0.333	**0.312**	0.333	0.334
F_t	0.249	0.170	0.185	0.291	0.261	**0.153**
G_f	0.266	**0.157**	0.221	0.260	0.258	0.158
Mean	0.242	0.176	0.214	0.250	0.249	**0.175**
Model params count	23904	71204	6762	12308	2600	33028
p_e	0.017	0.008	0.069	0.032	**0.155**	0.017

Table 4.6: Partial curve: test (B-dataset)

	CNN	Dense NN	LSTM NN	L-Maen	L-A-Maen	F-Maen
E_c	0.108	**0.058**	0.096	0.149	0.086	0.068
F_c	0.297	**0.275**	0.293	0.317	0.285	0.321
F_t	0.230	**0.120**	0.216	0.263	0.296	0.212
G_f	0.169	**0.112**	0.159	0.246	0.179	0.161
Mean	0.201	**0.141**	0.191	0.244	0.211	0.191
Model params count	23904	71204	6762	12308	2600	33028
p_e	0.021	0.010	0.078	0.033	**0.182**	0.016

Table 4.7: Full curve: training (B-dataset)

	CNN	Dense NN	LSTM NN	L-Maen	L-A-Maen	F-Maen
E_c	0.121	0.068	0.105	0.150	0.087	**0.065**
F_c	0.320	0.310	0.309	0.334	**0.283**	0.347
F_t	0.241	**0.156**	0.245	0.272	0.258	0.224
G_f	0.185	**0.146**	0.174	0.262	0.181	0.202
Mean	0.217	**0.170**	0.208	0.255	0.202	0.210
Model params count	23904	71204	6762	12308	2600	33028
p_e	0.019	0.008	0.071	0.032	**0.190**	0.014

Table 4.8: Full curve: test (B-dataset)

	CNN	Dense NN	LSTM NN	L-Maen	L-A-Maen	F-Maen
E_c	0.141	**0.061**	0.163	0.144	0.135	0.071
F_c	0.341	**0.314**	0.341	0.338	0.325	0.315
F_t	0.276	0.154	0.300	0.292	0.284	**0.128**
G_f	0.234	**0.120**	0.233	0.222	0.225	0.136
Mean	0.248	**0.162**	0.259	0.249	0.242	0.163
Model params count	23904	71204	6762	12308	2600	33028
p_e	0.017	0.009	0.057	0.033	**0.159**	0.019

Table 4.9: Partial curve: training (C-dataset)

	CNN	Dense NN	LSTM NN	L-Maen	L-A-Maen	F-Maen
E_c	0.147	**0.063**	0.171	0.146	0.150	0.070
F_c	0.348	0.321	0.350	0.338	0.332	**0.310**
F_t	0.281	0.154	0.288	0.297	0.282	**0.134**
G_f	0.225	**0.120**	0.225	0.212	0.228	0.138
Mean	0.250	0.164	0.258	0.248	0.248	**0.163**
Model params	23904	71204	6762	12308	2600	33028
p_e	0.017	0.009	0.057	0.033	**0.155**	0.019

Table 4.10: Partial curve: test (C-dataset)

	CNN	Dense NN	LSTM NN	L-Maen	L-A-Maen	F-Maen
E_c	0.109	0.061	0.106	0.099	0.095	**0.059**
F_c	0.322	**0.290**	0.322	0.315	0.295	0.324
F_t	0.221	**0.127**	0.226	0.222	0.200	0.180
G_f	0.178	**0.118**	0.158	0.171	0.166	0.127
Mean	0.207	**0.149**	0.203	0.202	0.189	0.172
Model params	23904	71204	6762	12308	2600	33028
p_e	0.020	0.009	0.073	0.40	**0.204**	0.018

Table 4.11: Full curve: training (C-dataset)

	CNN	Dense NN	LSTM NN	L-Maen	L-A-Maen	F-Maen
E_c	0.119	0.066	0.111	0.103	0.101	**0.063**
F_c	0.331	0.304	0.336	0.317	**0.296**	0.322
F_t	0.231	**0.138**	0.230	0.239	0.208	0.184
G_f	0.179	**0.128**	0.168	0.170	0.173	0.138
Mean	0.215	**0.159**	0.211	0.207	0.194	0.177
Model params	23904	71204	6762	12308	2600	33028
p_e	0.019	0.009	0.070	0.039	**0.198**	0.017

Table 4.12: Full curve: test (C-dataset)

4.2 Experimental results of explainability

In the experimental part, the proposed approaches for output correlation, gradient tracking and model parameter changes are first presented. The approaches are presented first with simple examples for easy demonstration. Subsequently, they are applied to models related to the task of bridges. Next, there are presented the result of the experimental evaluation of the Game theory approach with the usage of Shapley values on Maen models.

4.2.1 Experimental examples of proposed explainability approaches

Tracking the gradient and parameters - simple example

A simple example of gradient tracking is presented on a task of linear regression with the usage of a neural network. The model is following:

$$\texttt{model}(x) = d_1(x) + d_2(u), \tag{4.2}$$

where d_1 and d_2 are linear functions (dense layers with bias), x is the input scalar to the model and u is a Gaussian noise (scalar which is for each call of the model different). From the mentioned model, it can be assumed that the function d_1 will be significant for its functioning, while the function d_2 will be confusing, and the gradient will not be able to learn its parameters. Figure 4.1 show such behaviour.

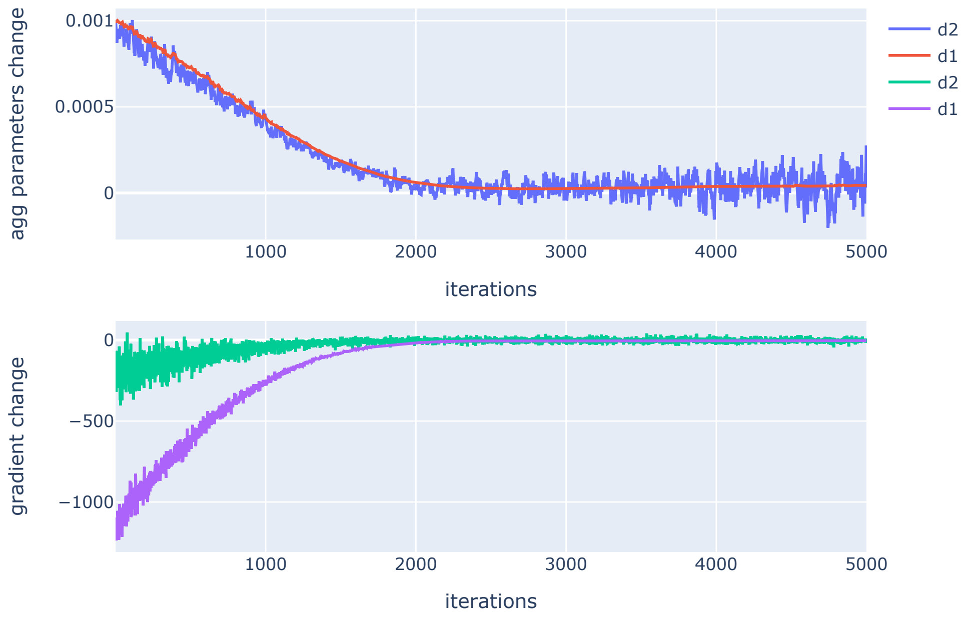


Figure 4.1: Gradient tracking (example model)

The dependence of the aggregated changes of the parameters of the model functions on the iterations shows that the function (d_1), into which the relevant data enters, does not change its parameters from the three thousandth iteration, while the second function (d_2), into which the noisy irrelevant data enters, performs, after converging, small oscillations around a steady value. Similar behaviour can also be seen in the dependence of the size of the gradients on the iterations, where it can be seen that the gradient relative to the function with a noisy input oscillates around zero. From the above observation, an assumption can be made about the usefulness of the model components.

Identifying statistically significant correlations - simple example

Here, we present a simple example of identifying statistically significant correlations. The task is to interpolate two-dimensional points (data are visible as blue points in Figure 4.4). The model is a Dense neural network with the five following dense layers:

1. dense layer: input dimension 1, output dimension 32,
2. dense layer: input dimension 32, output dimension 64,
3. dense layer: input dimension 64, output dimension 128,
4. dense layer: input dimension 128, output dimension 10,
5. dense layer: input dimension 10, output dimension 1.

The input to the neural network is the first coordinate x of data points, and the second coordinate y is an expected result to be interpolated. The fact that it makes sense to be concerned with the correlation of the neuron outputs with the expected outputs of the model is demonstrated in Figure 4.3, where it is shown that for the 27th neuron of the first layer of the model, its outputs are correlated with the expected outputs of the model (y) for positive expected outputs, while for the 5th neuron, there is visible a dependency for negative expected outputs (y) of the model.

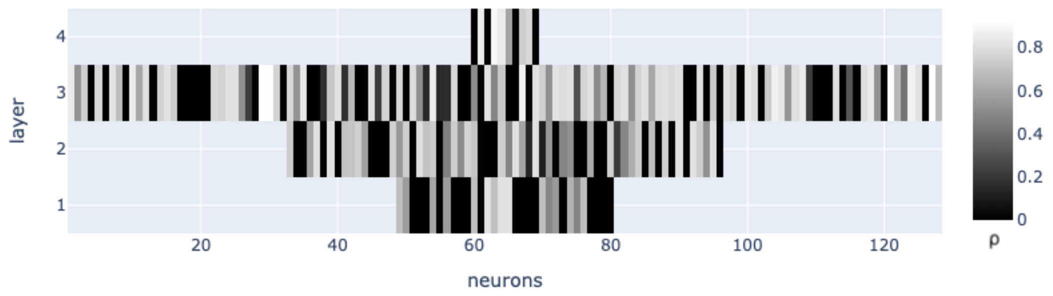


Figure 4.2: Correlations with the expected output for all neurons outputs.

Figure 4.2 shows the heatmap of Pearson correlations between neurons outputs and expected outputs of the model. It is evident, that there are many neurons whose outputs are not correlated with the expected output of the neural network according to the dataset points.

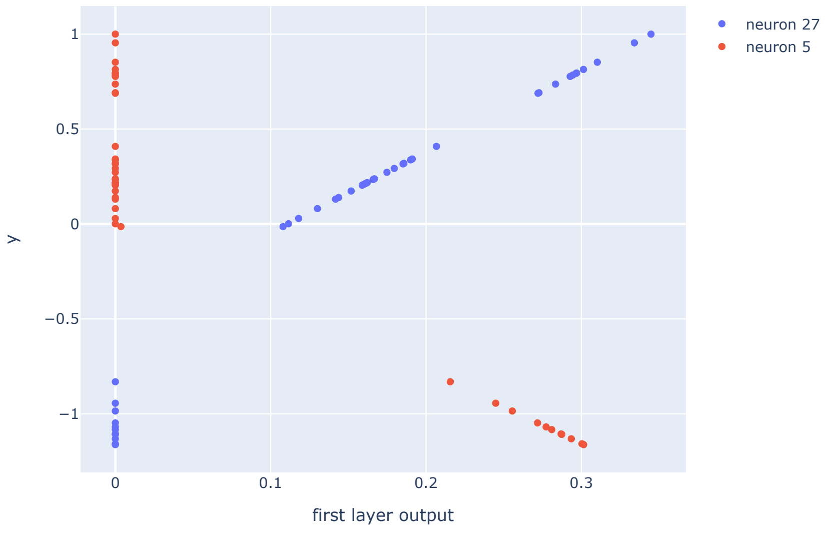


Figure 4.3: Correlations with the expected output for two neurons in the first layer.

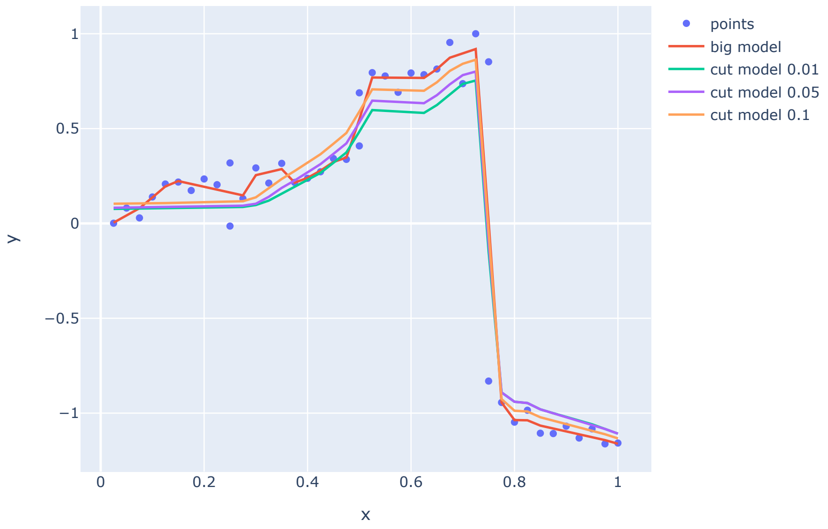


Figure 4.4: The comparison of dataset points, original *big model* and newly generated *cut models* with the level of significance.

Based on the knowledge about correlations of neurons outputs with model expected outputs there was performed model's pruning. The original model, in which all neurons of all layers are contained, is named *big model.* Models with cut layers and removed neurons are marked as *cut model. Cut models* were created from the original *big model* by removing neurons whose outputs at a certain level of significance are not correlated with the expected outputs of the entire model. The significance level of the correlation statistic test determines the threshold affecting how many neurons will be removed from the model. Along with lowering this threshold, there is a higher loss of information when removing neurons from the model. The model created on the basis of applying a significance level of 0.1 contains 64% of the parameters of the original model; in the case of a significance level of 0.01, it is 62%. Large losses of interpolation information occurred with the further lowering of the threshold. Figure 4.4 shows a comparison of the interpolation of individual models against the dataset points.

4.2.2 Experimental evaluations of explainability approaches on bridges task

Identifying statistically significant correlations - bridges task

The experimental evaluation was performed between Dense NN model neuron outputs, and material parameters are presented as follows. There was no point in dealing with the A-dataset due to its small size and distribution. Removing neurons from a model trained on a small dataset resulted in an immediate loss of transfer information and a significant increase in relative errors. On the B-dataset, it made sense to deal with the correlations to the material parameters E_c, F_t and G_f, for which there were no significant increases in the relative error when reducing the number of neurons. The experiment was evaluated according to the procedure of performing correlation tests of the outputs of all neurons against one of the material parameters. The results are presented in Table 4.13. On the C-dataset, it only made sense to deal with the correlations to the material parameters E_c and G_f. The results are presented in Table 4.14.

Material parameter	Significance level	Relative error	Parameters reduction
E_c	0.15	0.20	9.2%
F_t	0.15	0.19	9.5%
G_f	0.005	0.2	11.22%

Table 4.13: Reduction of trainable parameters (B-dataset)

Material parameter	Significance level	Relative error	Parameters reduction
E_c	0.003	0.17	6.6%
G_f	0.001	0.16	11.3%

Table 4.14: Reduction of trainable parameters (C-dataset)

Tracking the gradient and parameters - bridges task

The approach was applied to the L-A-Maen model for attention modules. The parameters of each of the modules were aggregated by summation of their absolute values into one scalar. Figure 4.5 shows the dependence of such aggregated parameter changes on iterations during gradient training of the model (Figure is generated by evaluation on the A-dataset, but the trends were very similar for all three datasets). It is noticeable that the optimizer was looking for a path for approximately the first hundred iterations. Only after the hundredth iteration can we talk about the gradual convergence of the parameters. It can be seen from the graph that approximately every fifty iterations, the optimizer tries to step out of the local minimum and find a different path while searching the space, which is reflected in the curves by periodic peak locations. Between these peaks, all attention blocks, except blocks four and ten, have only small parameter changes. At block ten, the parameter changes are damped only from the two-hundred-and-fiftieth iteration, which means that prematurely terminating the model training would mean that the parameters of this block will not be set correctly. On the other hand, the fourth block apparently starts to relearn from this iteration and the changes in its parameters increase. Overall, based on the trends in the graph, it can be stated that the attention modules are catching on to the data since the hundredth iteration and converge. The modules are not redundant.

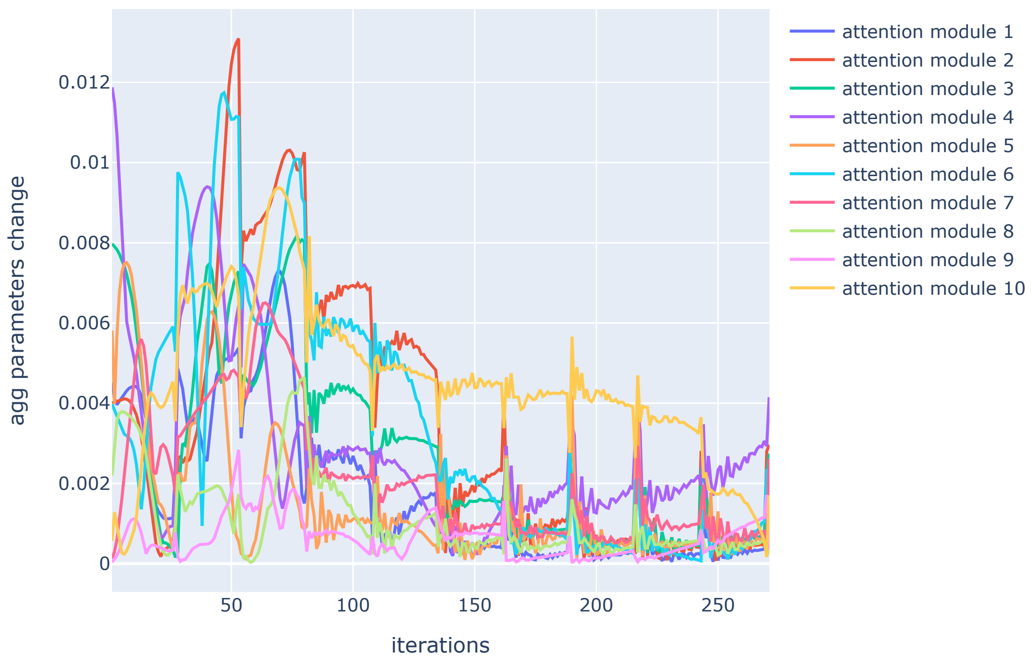


Figure 4.5: L-A-Maen model - tracking aggregated changes of parameters of attention modules.

Shapley values - Maen models

Explainability at the level of individual components of Maen models was performed using Shapley values. A full Shapley calculation examined the input components (yellow colour) - that is, by performing the analysis over all subsets without approximation. These are ten components corresponding to ten communities on each data sample, which was feasible from the point of view of the time required for the calculation. In contrast, on the hidden components of the model, the analysis was performed using approximation. From the point of view of the practical implementation of the method, the outputs of the components that were discarded in the given epoch were zeroed in the case of input components and multiplied by Gaussian noise in the case of hidden components.

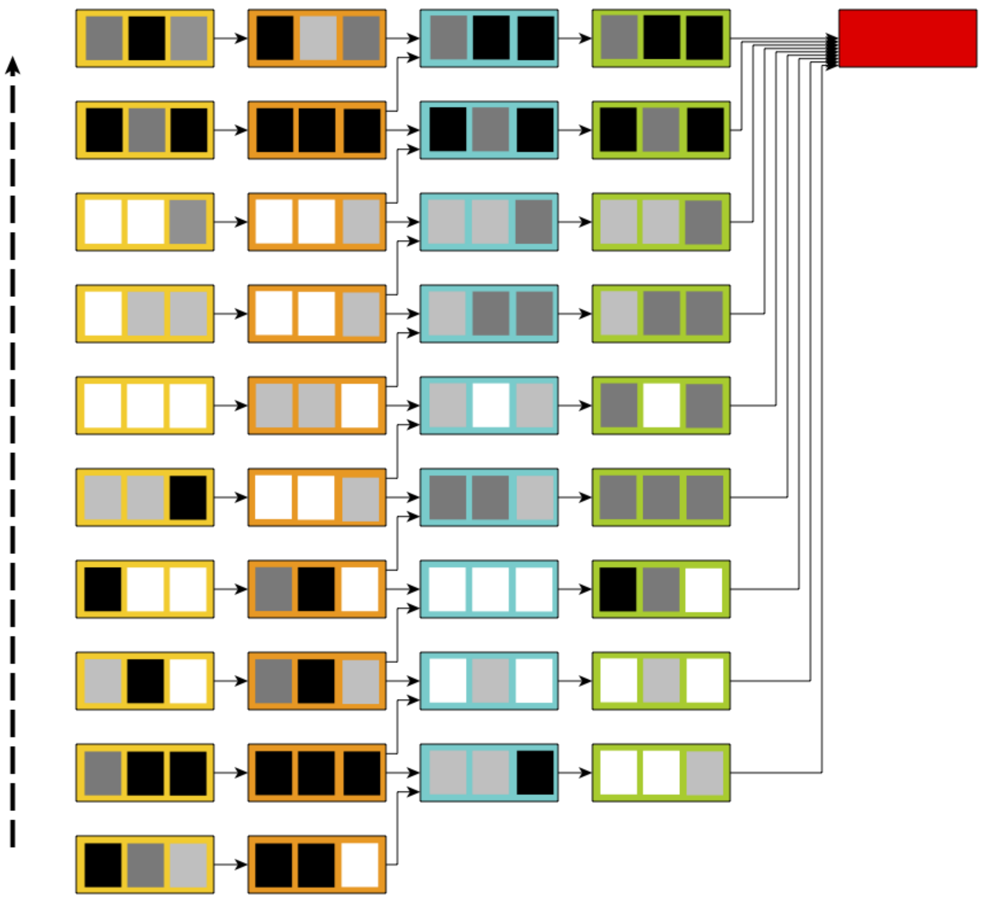


Figure 4.6: F-Maen model with Shapley values explanation over all datasets.

Figures 4.6, 4.7 and 4.8 show the significance of the components of the models according to Shapley numbers - depicted by grayscale distributions according to quantiles. The first quantile, corresponding to the smallest Shapley numbers, is represented by white rectangles, and the fourth quantile by black. Quantile distributions were applied separately for the input and hidden components. Three rectangles corresponding to the results on datasets A, B and C are assigned to each component.

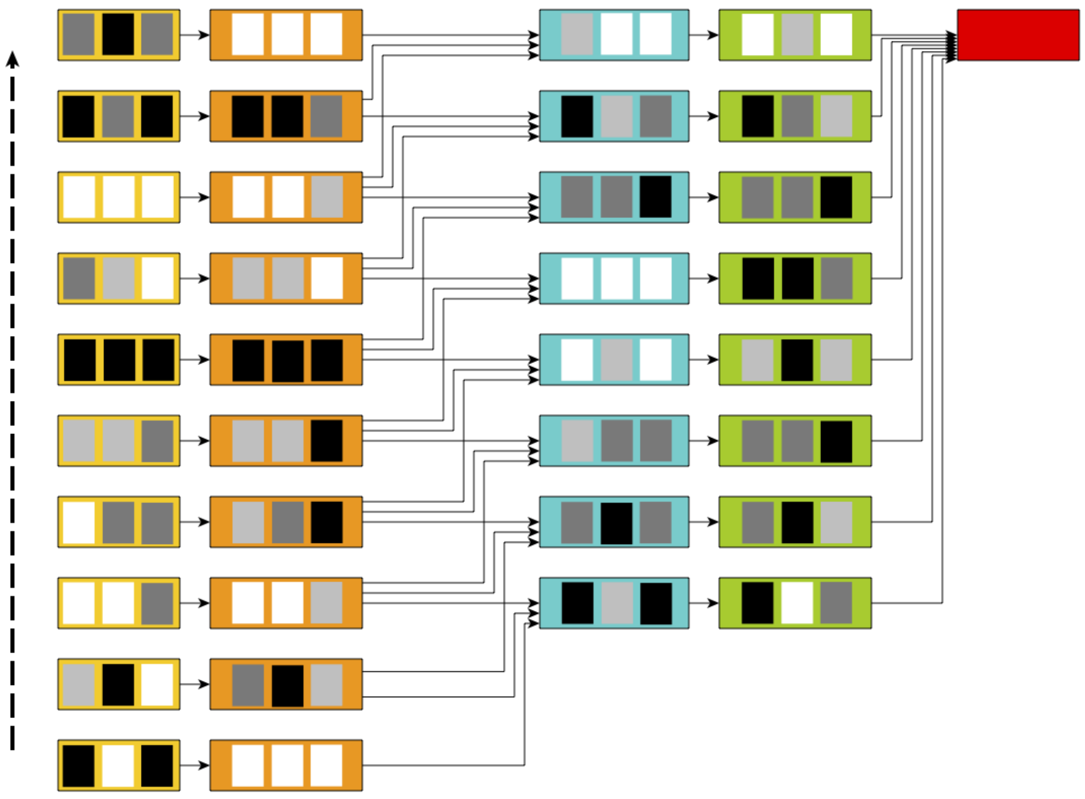


Figure 4.7: L-Maen model with Shapley values explanation over all datasets.

In the case of analyzing the explainability of the input components of all three Maen models, due to the evaluations of all datasets, it can be concluded that the first two components (communities of sample data properties), then one to two middle components and the last two components are important for functioning. From the point of view of interpretability explainability, it can be concluded from these experimental observations that the models draw information from the parts of the curves corresponding to the elastic deformation, as well as the behaviour of the curve in the area just after the break and the deformation of the material object. The last part of the sample carrying important information is the end state of the curve - that is, information about the behaviour of the material after the propagation of cracks and fracture defects has stabilized. For the F-Maen and L-Maen models, the hidden components approximately take over the significance of the input components to which they are linked. In the case of the explainability of the L-A-Maen model, an interesting observation is the cascading transfer of information in orange components. The left part of the model is assigned lower Shapley values than for the right part, which is especially evident on the blue and green layers. In general, it can be said about this architecture that the last (red) component, in the form of the decision-making layer, takes into account more the right side of the diagram than the left. The results of the explainability of the L-A-Maen model for all three datasets came out almost identically, which means that for this model, the Shapley values approximation does not cause excessive loss of explainability information.

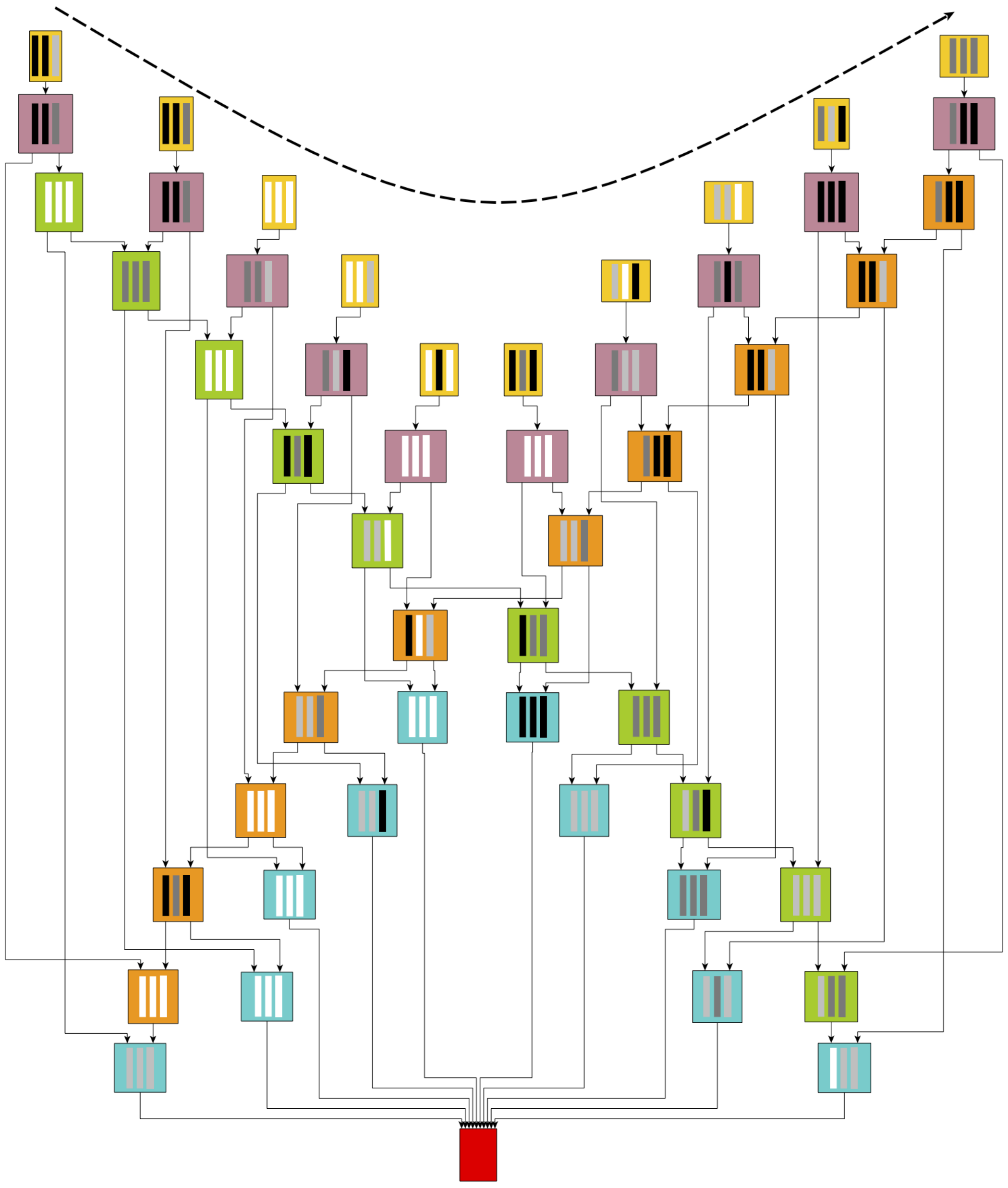


Figure 4.8: L-A-Maen model with Shapley values explanation over all datasets.

Chapter 5

Conclusion

This thesis introduced several approaches to the explainability of machine learning models. There were proposed neural network models for solving a task from the field of material engineering. Subsequently, the explainabilities of the approaches were performed. In addition to the explainability of the components of neural networks, the interpretability of the achieved results was carried out both from the point of view of explainability and from the point of view of the accuracy of the functioning of the given models.

On the basis of experimental evaluations of methods and approaches, it has been shown that it makes sense to deal with thorough pre-processing of data, including the detection of communities over the properties of data samples. On the basis of data preprocessing, architectures of neural network models can subsequently be created, which can accommodate the solved problem while simultaneously significantly reducing the computational complexity of the model through the reduction of trainable parameters of the neural network.

On the basis of the presented metric of the efficiency of the model parameters, the L-A-Maen model works best, which, thanks to its specific architecture, was able to solve the problem, even on a very small scale of the dataset. The conventional fully connected Dense NN was able to accommodate the solved problem with a sufficiently large dataset, due to its large number of trainable parameters. From the point of view of the accuracy of the models, it is, therefore, a trade-off between a dataset size on the one hand, and the necessity of data preprocessing on the other.

The creation of specific architectures of the neural network also has a big bonus for its explainability in terms of clear aggregation of neurons into components, which can significantly reduce the computational complexity of explainability methods (Shapley values). At the same time, the components of the topology, unlike only fully connected Dense layers or convolutions, can also be assigned human-understandable behaviour, for example, in the form of processing physical phenomena or transmitting contextual information of time-dependent events.

A comparison of approaches for explainability can be said to mainly depend on the use case. For the examination of larger functional blocks, it is advisable to use the Shapley number method or, due to the time complexity, its approximation. To examine individual trainable model parameters, it is more computationally efficient to monitor their convergence during model training. In the case of the possibility of easy or effective visualization of data, especially in the field of computer vision, it is advisable to use one of the methods based on the Grad-CAM principle.

The following conclusions can be drawn for the solved problem from the field of materials engineering. It is necessary to address the critical lack of dataset samples by collecting additional data. Using a larger dataset significantly improved model training. Given that the input curves carried information through values corresponding to the elastic deformation (and partially inelastic), it would be appropriate to increase the sampling during data collection at this stage of the loads on deflection measurement. The curves also carried information through the behaviour after stabilization of the propagation of cracks, which, however, from a practical point of view, is poorly usable information, if it is not possible to allow the destruction of the material object. Therefore, it is necessary to focus on the area of the curve before the phenomenon of material destruction (peak point of the sample curve). In the current datasets that were available, this is only 20 points out of a total of 61. Increasing the sampling during data collection would allow for better capture of the course of the curve, including the formation and propagation of small cracks even before the curve breaks. Small cracks form in the material during the transition between elastic and inelastic deformation, without fatal fractures and destruction, but these phenomena can tell about the character of the material and should be paid attention to. Contrary to original expectations, the most important information for solving the problem was not information about the breaking point, where the destruction of the material will occur. This can be explained by the fact that these points were widely scattered (especially on the A-dataset) and so poorly grasped by neural network models.

Bibliography

[AB02] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Rev. Mod. Phys.*, 74:47–97, Jan 2002.

[ACÖG19] Marco Ancona, Enea Ceolini, Cengiz Öztireli, and Markus Gross. *Gradient-Based Attribution Methods*, pages 169–191. Springer International Publishing, Cham, 2019.

[Ami19] Shervine Amidi. Cs 230 — deep learning. https://stanford.edu/~shervine/teaching/cs-230/, 2019.

[BAB20] Shane Barratt, Guillermo Angeris, and Stephen Boyd. Minimizing a sum of clipped convex functions. *Optimization Letters*, 14, 11 2020.

[BCB15] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[BDG$^+$06] U. Brandes, D. Delling, M. Gaertler, R. Goerke, M. Hoefer, Z. Nikoloski, and D. Wagner. Maximizing modularity is hard, 2006.

[BF23] Gianni Brauwers and Flavius Frasincar. A general survey on attention mechanisms in deep learning. *IEEE Transactions on Knowledge and Data Engineering*, 35(4):3279–3298, apr 2023.

[BFL$^+$17] David Balduzzi, Marcus Frean, Lennox Leary, J. P. Lewis, Kurt Wan-Duo Ma, and Brian McWilliams. The shattered gradients problem: If resnets are the answer, then what is the question? *CoRR*, abs/1702.08591, 2017.

[BGLL08] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, oct 2008.

[BK73] Coen Bron and Joep Kerbosch. Algorithm 457: Finding all cliques of an undirected graph. *Commun. ACM*, 16:575–577, 1973.

[Bro19] Jason Brownlee. A gentle introduction to exploding gradients in neural networks, Aug 2019.

[BS08] Susanne C. Brenner and Larkin R. Scott. *The Mathematical Theory of Finite Element Methods*, volume 15 of *Texts in Applied Mathematics*. Springer, 2008.

[Cas19] Isaac Castro. Gradcam-keras. https://github.com/isaaccasm/GradCAM-keras, 2019.

[Cay78] Professor Cayley. Desiderata and suggestions: No. 2. the theory of groups: Graphical representation. *American Journal of Mathematics*, 1(2):174–176, 1878.

[CKPW20] Zhiyong Cui, Ruimin Ke, Ziyuan Pu, and Yinhai Wang. Stacked bidirectional and unidirectional lstm recurrent neural network for forecasting network-wide traffic state with missing values, 2020.

[CKW18] Zhiyong Cui, Ruimin Ke, and Yinhai Wang. Deep bidirectional and unidirectional LSTM recurrent neural network for network-wide traffic speed prediction. *CoRR*, abs/1801.02143, 2018.

[CTB$^+$21] Hsiang-Yun Sherry Chien, Javier S. Turek, Nicole Beckage, Vy A. Vo, Christopher J. Honey, and Theodore L. Willke. Slower is better: Revisiting the forgetting mechanism in LSTM for slower information decay. *CoRR*, abs/2105.05944, 2021.

[DDS$^+$21] Morad Danishvar, Sebelan Danishvar, Francisco Souza, Pedro Sousa, and A. Mousavi. Coarse return prediction in a cement industry's closed grinding circuit system through a fully connected deep neural network (fcdnn) model. *Applied Sciences*, 11:1361, 02 2021.

[Deo94] Narsingh Deo. *Graph theory : with applications to engineering and computer science*. Prentice-Hall of India ; Prentice-Hall International, 1994.

[DP94] Xiaotie Deng and Christos H. Papadimitriou. On the complexity of cooperative solution concepts. *Mathematics of Operations Research*, 19(2):257–266, 1994.

[Dra23] Vojtěch Drahý. Maen. https://github.com/drvojtex/Maen/, 2023.

[FvZ22] Warren Freeborough and Terence van Zyl. Investigating explainability methods in recurrent neural network architectures for financial time series data. *Applied Sciences*, 12(3), 2022.

[Gib66] J. J. Gibson. *The senses considered as perceptual systems*. Houghton Mifflin, Boston, 1966.

[GN02] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, jun 2002.

[GZ20] Amirata Ghorbani and James Zou. Neuron shapley: Discovering the responsible neurons, 2020.

[HP17] Hossein Hosseini and Radha Poovendran. Deep neural networks do not recognize negative images. *CoRR*, abs/1703.06857, 2017.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, nov 1997.

[ILLM67] A.G. Ivakhnenko, V.G. Lapa, V.G. Lapa, and R.N. McDonough. *Cybernetics and Forecasting Techniques.* Modern analytic and computational methods in science and mathematics. American Elsevier Publishing Company, 1967.

[KAA+19] Serkan Kiranyaz, Onur Avci, Osama Abdeljaber, Turker Ince, Moncef Gabbouj, and Daniel J. Inman. 1d convolutional neural networks and applications: A survey, 2019.

[Kar72] Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.

[Koz21] Vojtěch Kozel. Hierarchical models of network traffic — dspace.cvut.cz. https://dspace.cvut.cz/handle/10467/94650, 2021. [Accessed 30-Mar-2023].

[KS09] Samir Khuller and Barna Saha. On finding dense subgraphs. pages 597–608, 01 2009.

[LCL+95] Shih-Chung B. Lo, Heang-Ping Chan, Jyh-Shyan Lin, Huai Li, Matthew T. Freedman, and Seong K. Mun. Artificial convolution neural network for medical image pattern recognition. *Neural Networks*, 8(7):1201–1214, 1995. Automatic Target Recognition.

[LeN19] Alexander LeNail. Nn-svg: Publication-ready neural network architecture schematics. *Journal of Open Source Software*, 4(33):747, 2019.

[LK15] Viet-Hoang Le and Sung-Ryul Kim. Using entropy cluster-based clustering for finding potential protein complexes. volume 9043, 04 2015.

[Man21] Manu. A simple overview of rnn, lstm and attention mechanism. https://medium.com/swlh/a-simple-overview-of-rnn-lstm-and-attention-mechanism-9e844763d07b, Feb 2021.

[MCFH22] Rory Mitchell, Joshua Cooper, Eibe Frank, and Geoffrey Holmes. Sampling permutations for shapley value estimation, 2022.

[Mil83] H.D. Mills. *Software Productivity.* Little, Brown Library of Radiology. Little, Brown, 1983.

[New10] M. E. J. Newman. *Networks: an introduction.* Oxford University Press, Oxford; New York, 2010.

[Ola15] Christopher Olah. Understanding lstm networks. *Understanding LSTM Networks – colah's blog*, Aug 2015.

[PL06] Pascal Pons and Matthieu Latapy. Computing communities in large networks using random walks. *Journal of Graph Algorithms and Applications*, 10(2):191–218, 2006.

[PLC10] S. Plis, Terran Lane, and Vince D. Calhoun. Permutations as angular data: Efficient inference in factorial spaces. *2010 IEEE International Conference on Data Mining*, pages 403–410, 2010.

[PMB13] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1310–1318, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[PS22] Guilherme Dean Pelegrina and Sajid Siraj. Shapley value-based approaches to explain the robustness of classifiers in machine learning, 2022.

[Rah68] N.A. Rahman. *A Course in Theoretical Statistics: For Sixth Forms, Technical Colleges, Colleges of Education, Universities*. Griffin books on statistics and mathematics. Griffin, 1968.

[RCC$^+$04] Filippo Radicchi, Claudio Castellano, Federico Cecconi, Vittorio Loreto, and Domenico Parisi. Defining and identifying communities in networks. *Proceedings of the National Academy of Sciences*, 101(9):2658–2663, feb 2004.

[RPSM21] Mohammad Rahimzadeh, Soroush Parvin, Elnaz Safi, and Mohammad Reza Mohammadi. Wise-srnet: A novel architecture for enhancing image classification by learning spatial resolution of feature maps. *CoRR*, abs/2104.12294, 2021.

[SAAA21] Mehmet Hakan Satman, Shreesh Adiga, Guillermo Angeris, and Emre Akadal. Linregoutliers: A julia package for detecting outliers in linear regression. *Journal of Open Source Software*, 6(57):2892, 2021.

[Sal18] Fathi M. Salem. Slim lstms. https://arxiv.org/abs/1812.11391, 2018.

[SCD$^+$19] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. *International Journal of Computer Vision*, 128(2):336–359, Oct 2019.

[Sch19] Robin M. Schmidt. Recurrent neural networks (rnns): A gentle introduction and overview. *CoRR*, abs/1912.05911, 2019.

[Sen09] Pavel Senin. Dynamic time warping algorithm review. 01 2009.

[SGGZ18] Julian Stier, Gabriele Gianini, Michael Granitzer, and Konstantin Ziegler. Analysing neural network topologies: a game theoretic approach. *Procedia Computer Science*, 126:234–243, 2018. Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 22nd International Conference, KES-2018, Belgrade, Serbia.

[SH20] Alex Shenfield and Martin Howarth. A novel deep learning model for the detection and identification of rolling element-bearing faults. *Sensors (Basel, Switzerland)*, 20, 09 2020.

[Sha51] Lloyd S. Shapley. *Notes on the N-Person Game &mdash; II: The Value of an N-Person Game*. RAND Corporation, Santa Monica, CA, 1951.

[Soy22] Derya Soydaner. Attention mechanism in neural networks: where it comes and where it goes. *Neural Computing and Applications*, 34(16):13371–13385, may 2022.

[STY17] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. *CoRR*, abs/1703.01365, 2017.

[SYK$^+$22] Rabia Saleem, Bo Yuan, Fatih Kurugollu, Ashiq Anjum, and Lu Liu. Explaining deep neural networks: A survey on the global interpretation methods. *Neurocomputing*, 513:165–180, 2022.

[TG19] Erico Tjoa and Cuntai Guan. A survey on explainable artificial intelligence (XAI): towards medical XAI. *CoRR*, abs/1907.07374, 2019.

[VSP$^+$17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[WDYZ19] Haofan Wang, Mengnan Du, Fan Yang, and Zijian Zhang. Score-cam: Improved visual explanations via score-weighted class activation mapping. *CoRR*, abs/1910.01279, 2019.

[WS05] Scott White and Padhraic Smyth. A spectral clustering approach to finding communities in graph. volume 5, 04 2005.

[YGG$^+$13] Shang Yang, Sheng-Uei Guan, Shu Guo, Lin Zhao, Wei Li, and Hong Xue. Neural network output partitioning based on correlation. *Journal of Clean Energy Technologies*, pages 342–345, 01 2013.

[ZKL$^+$15] Bolei Zhou, Aditya Khosla, Àgata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. *CoRR*, abs/1512.04150, 2015.

[ZLW21] Xingwang Zhao, Jiye Liang, and Jie Wang. A community detection algorithm based on graph compression for large-scale social networks. *Information Sciences*, 551:358–372, 2021.