Graph Neural Networks for Model-Based Reinforcement Learning

Graph Neural Networks für Model-Based Reinforcement Learning
Bachelor thesis by Marius Zöller
Date of submission: February 25, 2021

1. Review: M.Sc. Michael Lutter
2. Review: Prof. Dr. Jan Peters
Darmstadt

TECHNISCHE UNIVERSITÄT DARMSTADT

Erklärung zur Abschlussarbeit gemäß §22 Abs. 7 und §23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Marius Zöller, die vorliegende Bachelorarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Fall eines Plagiats (§38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung gemäß §23 Abs. 7 APB überein.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

[illegible]

Darmstadt, February 25, 2021

Marius Zöller

Abstract

Graph Neural Networks (GNNs) are neural networks with a promising architecture that enables them to operate on graph structured data. This structure can represent the decomposition of physical scenes into entities and their relationships. This work examines the suitability of this architecture to learn physical systems and function as a simulator. To this end experiments are conducted that examine the generalisation capability and physical understanding of GNNs. The inner workings of the model are explored via a system identification approach in a partially supervised scenario. The used GNN models were able to generalise well to different system sizes and performed on par or outperformed the chosen baselines.

Zusammenfassung

Graph Neural Networks (GNNs) sind neuronale Netzwerke mit einer vielversprechenden Struktur, die es ihnen erlaubt auf Daten mit einer Graphstruktur zu arbeiten. Diese Struktur kann die Aufteilung physikalischer Szenen in einzelne Einheiten und ihre Verbindungen repräsentieren. Die vorliegende Arbeit untersucht die Eignung dieser Architekur für das Lernen physikalischer Systeme um als Simulator dieser zu agieren. Dafür werden Experimente durchgeführt, die die Fähigkeit zur Generalisierung und dem Verstehen von physikalischen Gegebenheiten von GNNs untersuchen. Die Funktionsweise der Modelle wird untersucht indem in einem teilüberwachten Szenario das betreffende System identifiziert und analysiert wird. Die benutzten GNN Modelle haben starke Generalisierungsfähigkeiten in Bezug auf verschiedene Größen von System gezeigt und haben so gut wie die Vergleichsmodelle oder bessere Ergebnisse bewiesen.

Contents

Figures and Tables

List of Figures

Abbreviations, Symbols and Operators

List of Abbreviations

Notation	Description
CNN	Convolutional Neural Network
GNN	Graph Neural Network
GRU	Gated Recurrent Unit
MLP	Multilayer Perceptron
SRNN	Symplectic Recurrent Neural Network

List of Symbols

Notation	Description
$\mathbf{e}_{ij}$	vector of edge attributes of the edge going from vertex i to vertex j.
$\mathbf{e}_{i*}$	Set of outgoing edges from vertex i

$\mathbf{e}_{*i}$	Set of incoming edges to vertex i
E	set of all edges
$\mathbf{g}$	vector of global attributes
$\mathcal{N}_i$	set of neighbouring vertices of vertex i
$\boldsymbol{\theta}$	vector of parameters of a neural network
$\mathbf{v}_i$	vector of vertex attributes of vertex i
V	set of all vertices

List of Operators

Notation	**Description**	**Operator**
$\|\|$	Vector concatenation	$(\bullet) \|\| (\bullet)$

1. Introduction

Autonomous robots are transitioning from being confined to factories to supporting humans in their everyday life. Simple service robots such as Roombas, autonomous robotic vacuum cleaners, have been used in households for more than a decade and have been well accepted [1]. Robots that perform more challenging tasks such as object manipulation via grasping have so far not arrived in the domestic domain due to the complexity of those tasks [2]. Search and rescue environments have been another area where there is a lot of research interest [3].

What all those scenarios have in common is that the robot has to deal with different environments and must be able to accomplish its task. This requires the robotic system to learn strategies that it can employ regardless of the environment. Reinforcement Learning is such a technique that allows the learning of general strategies, so called policies, that react to different environments. Learning a policy that can perform well in a range of environments is a challenging task with a high sample complexity, i.e. a huge amount of data is needed for the training process [4]. Generating this data from physical robots is often prohibitively expensive time wise, while also risking damage to the robot and its environment.

Virtual physical environments, so called simulators, speed up the training process and prevent damage to equipment. This allows reinforcement learning algorithms to be feasible. Furthermore simulators can also provide additional sensory information that the physical system might not yield, as well as gradient information for a more efficient policy search. Since the reinforcement learning model should still be applied in the real world, the simulator itself needs to portray the real world as precisely as possible. Errors in the simulation can be abused by the policy to achieve the goal in a way that is not possible in the reality and may harm the robot.

Instead of using traditional physics based simulators such as Mujuco [5] the usability of neural network based simulators which learn from data is examined in this work. The focus is put on how well the simulator learns the physics that govern the system instead of interpolating between seen data. To bias the learning into that direction specific architecture choices are made. Physical scenes can often be interpreted as a group of entities that interact with each other. The entities can arise either naturally as point masses in simple simulations or as components of robots or as the result of discretisation schemes where bodies are divided into virtual parts. Those entities and their interactions can be well translated into a graph and as such processed by a GNN [6].

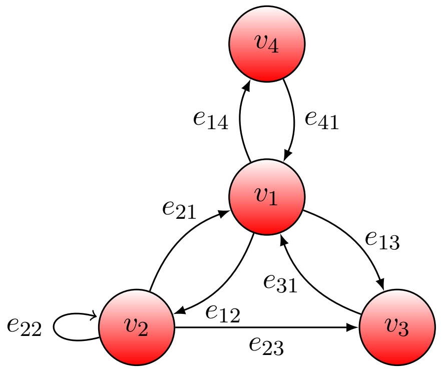
To provide background information on GNNs chapter 2 provides information about the history and workings of this archetype. Different interpretations are highlighted and the implementation that is used in this work is explained. In chapter 3 the experimental setup is presented that is meant to examine the capabilities of GNNs. The results of those experiments and the insights gained from them are discussed in chapter 4. In chapter 5 the understanding gained from the examination of GNNs is summed up, as well as some problems that exist. An outlook on further research topics in this area and possible further steps is provided in chapter 6.

2. Foundations

In this work GNNs will be used as architecture for the models. This describes a class of machine learning algorithms that take graph structured data as input and compute arbitrary, task dependent outputs. The architecture leverages the fact that each vertex should be treated equally which is helpful for generalisation as well as sparseness of parameters. The following chapter gives an overview over the history of GNNs and the inner workings of the archetype.

2.1. Nomenclature

First an appropriate nomenclature needs to be established. Graphs in this context consist of attributed vertices $\mathbf{v}_i$ and attributed edges $\mathbf{e}_{ij}$ connecting them. Input in this form will be referred to as graph structured data. The vertices represent entities and the edges represent the relationships between those entities. In addition a graph can have a global state $\mathbf{g}$ that is also taken as input. The connectivity structure of the graph, i.e. which vertices are connected by edges, is given as an adjacency matrix $\mathbf{A}$. In this matrix zeros indicate the absence and ones indicate the presence of an edge. If the edge $\mathbf{e}_{ij}$ is present, the matrix element a_{ij} is one. An example can be seen in Fig. 2.1. The example graph is directed and contains a self-loop in $\mathbf{e}_{22}$. Outgoing edges of $\mathbf{v}_i$ are written as $\mathbf{e}_{*i}$ and incoming edges as $\mathbf{e}_{i*}$. The neighbourhood $\mathcal{N}_i$ of a vertex $\mathbf{v}_i$ is defined here as the vertices to which $\mathbf{v}_i$ is directly connected by an edge, i.e. the one-hop neighbourhood. The set of all vertices is referred to as V and the set of all edges as E. The combination of those sets characterise complete graph as $\mathcal{G} = (\mathbf{g}, V, E)$. The dimensionality of the edges, vertices and the global state vectors is task and architecture dependent and components can be zero dimensional if there is no data available in the data set.



(a) A directed graph with 4 vertices.

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

(b) The corresponding adjacency matrix $\mathbf{A}$.

Figure 2.1.: Example of a directed graph and its adjacency matrix.

The task-dependent outputs can be divided into the classical machine learning classes of regression and classification, depending on whether the labels that should be predicted are real numbers or belong to a fixed set of integers. The tasks can be further classified based on the level where the predictions are made in the graph. This can be on the vertex, edge or global level.

2.2. History

The first description of this archetype as a way to do machine learning on graph structured data was in Scarselli et al., 2009 [7]. It focuses on node-level tasks. This work defines GNNs as a recurrent scheme. It utilises a transition function $\mathbf{v}_i = f\left(\mathcal{N}_i, \mathbf{e}_{i*}; \theta\right)$ that is used until convergence. Then a prediction is made by an output function $\mathbf{o}_i = g\left(\mathbf{v}_i\right)$. The transition function f is required to be a contraction map which needs to satisfy the corresponding condition

$$|f(\mathbf{\hat{v}_i}) - f(\mathbf{v_i})| < \mu|\mathbf{\hat{v}_i} - \mathbf{v_i}| \quad \exists\mu \quad \text{with} \quad 0 < \mu < 1 \quad \forall\mathbf{\hat{v}_i}, \mathbf{v_i} \quad ,$$

where $\mathbf{\hat{v}_i}$ is the current approximation of the solution and $\mathbf{v_i}$ the true value. A repeated application of the transition function causes the result to converge to the solution due to the steadily decreasing distance to the solution.

This model deals with the varying number of neighbours and corresponding incoming edges by decomposing the transition function

$$f\left(\mathcal{N}_i, \mathbf{e}_{i*}; \boldsymbol{\theta}\right) = \sum_{\mathbf{v}_x \in \mathcal{N}_i} h(\mathbf{v}_x, \mathbf{e}_{x*}; \boldsymbol{\theta})$$

into a sum of functions on neighbour-edge pairs. This approach is later called an aggregation function. Training is done as usual for machine learning applications by defining an objective function, computing gradients w.r.t the weights $\boldsymbol{\theta}$ and then performing a variant of gradient descent.

The restriction of the transition function to a contraction map limits the expressivity of the network. The transition function needs to be either designed to respect this restriction by choosing appropriate activation functions that limit the range of results or adding a penalty term to the objective function. In addition long range dependencies are difficult to model since each propagation step only causes an exchange of information with the immediate neighbourhood of each vertex. After n steps each vertex has received information from its n-hop neighbourhood. Due to the contraction map property the information content decays exponentially, which leads to vertices receiving less information from vertices the further they are away from each other. This problem can be alleviated by using gated graph neural networks that employ Gated Recurrent Units (GRUs) [8] for a fixed number of iterations instead of iterating the transition function until it converges [9]. This approach can also be used to output sequences instead of single predictions by utilising a readout function after each recurrent step.

A different interpretation of GNNs was introduced in the physics context of dynamical systems [10]. The goal here was the prediction of the evolution of states of particles. In this context the components of a graph have the natural interpretation of particles as vertices and pairwise particle interactions, i.e. forces, as edges. This interpretation led to a more explicit modelling of those interactions as functions of the sender and receiver vertices as well as the corresponding edge attributes. The interactions are afterwards used as input for the vertex update function as preprocessed input instead of directly inputting the edge attributes. The evolution of the states is a temporal development that is achieved by repeatedly applying the GNN to the output of the previous step. This can be seen as a reinterpretation of the recurrent transition where the intermediate outputs are states on the evolution trajectory. This way, in each time step interactions travel only one hop through the network so the length of the time step and the graph structure need to be aligned.

Building on top of this a general framework was introduced in which most types of GNNs can be formulated [11]. In this work the different levels of the graph get explicit intermediate representations by adding edge, vertex and global level update functions. The update functions are called in this order and the result from the previous function is taken into account. The update functions all take a set of relevant edges, vertices and the global attribute as input. For the edge function the input is just the edge itself, the corresponding sender and receiver, and the global attribute. For the vertex function this is the vertex itself, the set of all incoming edges and the global attribute. For the global function this is the set of all edges, all vertices and the global attribute. To deal with the operation on sets aggregation functions are needed that reduce the set to a single vector, so in addition to the three update functions three aggregation functions are needed. These are required to be commutative and associative to preserve the permutation invariance that is inherent to sets. Element-wise sum is the most popular operator but other operators that fulfil this requirement are also possible, such as the mean, minimum or maximum operator. Different aggregators capture different properties of the sets which can lead to improvements depending on the task. Further details are described in section 2.3.

The update functions are not restricted in their form and are usually parametrised by small neural networks. The group of those six functions makes up what is called a graph network block. The order of the update function can be changed depending on the level on which predictions are made. This work also deals with the combination of multiple graph network blocks. This can be done by chaining multiple blocks, i.e. using the output of one block as the input for the next block. The interpretation of this depends on whether the blocks share parameters. If they share parameters this can be viewed akin to recurrent networks whereas blocks with individual parameters can be seen as multiple layers of a neural network. The advantages and disadvantages are also similar to those analogous types. Parameter sharing provides stronger regularisation, whereas individual parameters increase the capacity of the model more. Individual parameters also allow for more flexibility in the design of the network architecture since the blocks are not limited to have the same input as output size. This building block can be utilised in an encoder, processor, decoder architecture [12]. The encoder and decoder steps project the input graph in and out of latent space, respectively. In the processor step multiple graph network blocks are used to update the states.

2.3. Variants of aggregation functions

The expressive power of GNNs is limited by the equal treatment of each neighbourhood [13]. The non-utilisation of statistics of the neighbourhoods such as the (element-wise) minimum, maximum, or the size of the neighbourhood limits the power of the model. Introducing scalers based on the size of the neighbourhood and stacking multiple aggregators preserves more information that can be used in creating more informative representations. Instead of treating the aggregator choice as a hyper parameter, they can also be learned [14]. Those aggregators utilise an attention mechanism [15]. For each vertex in the neighbourhood of the considered vertex $\mathbf{v}_i$ a pairwise attention coefficient is calculated by using a Multilayer Perceptron (MLP). Those attention coefficients are normalised in each neighbourhood by using a softmax function. The normalised coefficients are then used as weights for a weighted sum aggregation of the neighbourhood.

Sets can also be dealt with in alternative ways. Different weight matrices can be used for the update function based on the degree of the vertex on top of a sum aggregation function [16]. This idea allows for more individual treatment of vertices if their degree is a strong indicator for the type of vertex. The downside to this strategy is, that it does not generalise to unseen data that contains higher vertex degrees. This strategy can be employed if domain knowledge about the highest possible vertex degree is available. Janossy pooling [17] is a way of constructing permutation invariant functions. For this pooling strategy all possible permutations of the set that is to be processed are built and input into a permutation variant function. The mean of the outputs is taken as the result. Calculating the full set of permutations and backpropagating through this step is often prohibitive due to the superexponential complexity of $\mathcal{O}(!n)$ and the size of the neighbourhood sets. The proposed solution is taking a subset of a fixed size of the neighbourhood set instead by random sampling or enforcing an order.

2.4. Alternative interpretations of graph networks

An interpretation of GNNs that is closer to traditional graph algorithms is GNNs as a form of message passing networks [18]. This means that each node generates a specific message for each of its neighbours that it sends to them. Those nodes in turn aggregate those messages and update their own state. This is akin to the belief propagation algorithm for probabilistic graphical models [19]. Graphical models represent a probability distribution of random variables by modelling the variables as vertices and the conditional structure as the edges. Belief propagation is used for inference in those models, more specifically calculating the marginals in regards to the variables. This can be seen as a node level task. Here messages are also generated based on the neighbourhood and send to them. This process is iterated until convergence. This is similar to the original GNN [7] because in both cases the message passing is iterated until convergence [9]. The replacement of the convergence criterion by a fixed number of recurrent steps in the gated graph neural network can be seen analogue to truncated belief propagation [20] where the belief propagation is also limited to a fixed number of iterations.

GNNs can also be viewed as a generalisation of Convolutional Neural Networks (CNNs) [21],[22],[23] to generalised neighbourhoods. CNNs are most often used for processing 2D data such as visual images. Those images have a regular grid structure due to being made of pixels. In natural images there is a strong correlation between pixels that are spatially close to each other which decreases the farther away the pixels are. This strong local coherence is leveraged by using convolutions with a kernel and the fixed, square neighbourhood of a pixel, e.g. the eight pixels directly surrounding the centre pixel for a 3x3 convolution. This convolution can be seen as a weighted sum of the neighbouring pixels and the centre pixel. This operation is similar to a GNN that is only using node level updates, the identity function as update function and an attention mechanism as aggregation function. The attention mechanism would have to take the position of the neighbours in relation to the centre pixel into account by either using attributed, directed edges or position attributes of the pixels. The neighbourhood structure depends on the size of the convolution kernel. For a 3x3 convolution each vertex needs to be connected to the eight vertices surrounding it. The constant size of the neighbourhoods as well as the ordering imposed on the neighbourhood set by the spatial interpretation allows CNNs to have this simpler update structure.

2.5. Implementation

In this work a network based on graph network blocks [12] is utilised. It is shown in algorithm 1. The graph is input in its component sets V, E and $\mathbf{g}$. Additionally the adjacency matrix $\mathbf{A}$ that represents the connectivity structure of the graph. The fetching of vertices and edges that belong together is done via this adjacency matrix. Variables utilising the hat-notation are intermediate representations or approximations to the true value. From this general formulation different networks can be derived by omitting components, e.g. not keeping a global state or choosing different implementations for the encode, decode and integrator functions. For the aggregation function a sum aggregator is chosen. The exact details can be found in appendix A.1.

Training

The networks are trained in a supervised fashion by giving the network an input state and letting it predict the next N steps. The loss is calculated by comparing the predictions to the ground truth via a weighted MSE in the form of

$$\text{loss} = \left[\sum_{i=0}^{N-1} |\mathbf{v}_i - \hat{\mathbf{v}}_i|_2^2 \right] \tag{2.1}$$

$\hat{v}_i$ is the vector of all states in time step i. Training is done for 100 epochs unless stated otherwise. The training is done using mini batches of size 256. The loss is minimised by backpropagating from there and using the gradient information for optimisation with Adam [24]. The learning rate is initially set to $\eta = 4 \times 10^{-3}$. A scheduler is employed to reduce the learning rate dynamically based on the performance on the validation set. If the validation error stagnates for 15 epochs the learning rate is reduced to $\eta = 0.7\eta$.

Algorithm 1 Graph Neural Network (GNN)

```
1: **Input:** node information $V$, edge information $E$, adjacency matrix $\mathbf{A}$, time step $\Delta t$
2:
3: # Encoding
4: $\hat{\mathbf{v}}_i \leftarrow$ ENCODE_NODES$(\mathbf{v}_i ; \boldsymbol{\theta}_e)$
5: $\hat{\mathbf{e}}_{ij} \leftarrow$ ENCODE_EDGES$(\mathbf{e}_{ij} ; \boldsymbol{\theta}_v)$
6: # Processing
7: for each network layers $l$ do
8:     for each edge $\hat{\mathbf{e}}_{ij} \in E$ do
9:         $(\hat{\mathbf{v}}_s , \hat{\mathbf{v}}_r) \leftarrow$ Get corresponding sender and receiver node
10:        $\hat{\mathbf{e}}_{ij} \leftarrow$ UPDATE_EDGE$(\hat{\mathbf{e}}_{ij} , \hat{\mathbf{v}}_s , \hat{\mathbf{v}}_r ; \boldsymbol{\theta}_{e,l})$
11:    end for
12:    for each node $\mathbf{v}_i \in V$ do
13:        $\hat{\mathbf{e}}_{i*} \leftarrow$Aggregate incoming edges of $\hat{\mathbf{v}}_i$
14:        $\hat{\mathbf{v}}_i \leftarrow$ UPDATE_NODE$(\hat{\mathbf{e}_{i*}} , \hat{\mathbf{v}}_i ; \boldsymbol{\theta}_{v,l})$
15:    end for
16:    $\hat{\mathbf{e}}_{*,*} \leftarrow$ Aggregate all edges
17:    $\hat{\mathbf{v}}_* \leftarrow$ Aggregate all nodes
18: end for
19:
20: # Decoding
21: for each node $\hat{\mathbf{v}}_i$ do
22:     $\hat{\mathbf{e}}_{i*} \leftarrow$ Aggregate incoming edges of $\hat{\mathbf{v}}_i$
23:     $\Delta\mathbf{v}_i \leftarrow$ DECODE$(\hat{\mathbf{e}_{i*}} , \hat{\mathbf{v}}_i ; \boldsymbol{\theta}_d)$
24: end for
25:
26: # Integration
27: $\hat{\mathbf{v}}_i \leftarrow$ INTEGRATOR$(\hat{\mathbf{v}}_i , \Delta\mathbf{v}_i , \Delta t)$
```

2.6. System identification

The richness of observation data of physical systems is limited by the amount of sensors and positioning of those sensors. Not all physical quantities can be measured at each position due to limited spaces or other limiting conditions.
System identification describes the process of estimating latent parameters of a system based on observations of that system [25]. This can either be done to get a more complete description of the dynamical system or to use this information for downstream tasks that utilise the observation data enriched with the estimated parameters. The estimation additionally helps for generalisation to similar systems. For experiment 3.5 the second way will be employed since the parameters of the system will be treated as unobserved as opposed to observed in the other experiments. For this the GNN described in section 2.5 will be extended as described in algorithm 2. A vertex and an edge matrix for each system are created and filled with parameters that need to be estimated. The information to which system a vertex or edge belongs is given as input. Before the GNN is called the given input data is enriched with the estimated data. The training is gradient-based by backpropagating the supervised loss of the GNN back to the system identification parameters.

Algorithm 2 Graph Neural Network (GNN) with system identification

```
1: **Input:** node information $V$, time step $\Delta t$
2:
3: # Retrieval
4: **for** each node $\mathbf{v}_i \in V$ **do**
5:     $\hat{\mathbf{v}}_{i,\text{param}} \leftarrow$ Retrieved learned parameters for this vertex
6:     $\mathbf{v}_i \leftarrow \mathbf{v}_i || \hat{\mathbf{v}}_{i,\text{param}}$
7:     Add $\mathbf{v}_i$ to $V$
8: **end for**
9: **for** each edge $\hat{\mathbf{e}}_{ij} \in E$ **do**
10:     $\hat{\mathbf{e}}_{ij,\text{param}} \leftarrow$ Retrieved learned parameters for this edge
11:     $\mathbf{e}_{ij} \leftarrow \mathbf{e}_{ij} || \hat{\mathbf{e}}_{ij,\text{param}}$
12:     Add $\mathbf{e}_{ij}$ to $E$
13: **end for**
14:
15: # Call GNN algorithm with estimated input
16: $\hat{\mathbf{v}}_i \leftarrow$ GRAPH NEURAL NETWORK (GNN)($V$ , $E$ , $\mathbf{A}$ , $\Delta t$)
```

3. Experiments

To examine the ability of GNNs to reason about physical systems and understand basic concepts of them, the following experiments are proposed. Each experiment highlights different aspects of GNNs properties.

3.1. Data set generation

The data set is generated by an analytic simulator of a spring-chain system that is clamped on both ends. Each part of the chain is linked to its two neighbours via a spring. The two links at the edges are connected with a spring to the clamped ground. Due to this connection at the boundaries a chain of n links has $n + 1$ springs between them in total. An example of a system can be seen in Fig.3.1. The deflection is one dimensional and in the vertical direction. This means that the chains are only affected by the spring forces and there is no gravitational force. Dissipative forces such as friction are ignored so that the system conserves its energy. This system can be likened to a discretised guitar string that is vibrating. The individual links have a one dimensional position q and momentum p as

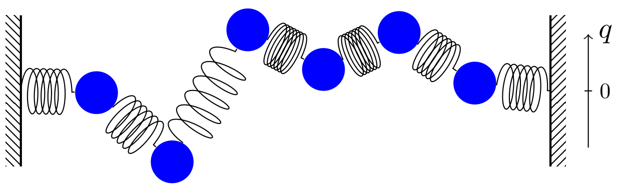


Figure 3.1.: Visualisation of a 6 link chain system. The 6 links are represented by the blue circles and connected by springs. They are regularly spaced horizontally and only have one degree of freedom in the vertical direction.

$$\mathbf{k} = \begin{bmatrix} k_1 & k_2 & 0 & 0 & 0 & 0 \\ k_2 & 0 & k_3 & 0 & 0 & 0 \\ 0 & k_3 & 0 & k_4 & 0 & 0 \\ 0 & 0 & k_4 & 0 & k_5 & 0 \\ 0 & 0 & 0 & k_5 & 0 & k_6 \\ 0 & 0 & 0 & 0 & k_6 & k_7 \end{bmatrix}$$

Figure 3.2.: The given connectivity structure with the spring constants of a 6 link chain system. The matrix is mainly populated on the first off-diagonals due to the connection of each vertex to its two neighbours. This connection is symmetric. The two elements on the main diagonal represent the connection of the first and last link to the ground.

dynamical states and a mass m as static state. The connecting springs are characterised by their spring constant k that describes the stiffness of the spring. The springs are assumed to be massless. The connectivity structure of each link having two neighbours is also given as input to the model. An example matrix showing off the symmetric connectivity structure can be seen in Fig. 3.2. The data set by Zhengdao Chen is taken from github[1] and was used in the corresponding paper [26]. Their Symplectic Recurrent Neural Network (SRNN) model uses a MLP to predict the Hamiltonian of the system, computes the partial derivatives and uses those with a numerical integrator to generate the outputs. This model is used as a baseline where appropriate. Further details can be found in appendix A.2.

Multiple scenarios are evaluated where the number of links, the initialisation of the masses and spring constants, as well as the fraction of supervised states are varied. For each scenario 130 different systems are examined. The systems display different initial states in regards to position and momentum. They are sampled from a normal distribution with a mean of $\mu = -0.5$ and a variance of $\sigma^2 = 5$. For those systems rollouts of 10000 time steps of $dt = 0.001\,\mathrm{s}$ are generated with the simulator for a total time length of 10 seconds per rollout. They will be referred to as trajectories. The trajectories are coarsened with a factor 10, i.e. every tenth time step is kept, so that the length of the trajectories is reduced to 1000. The time length stays at 10 seconds since the effective time step is

[1]https://github.com/zhengdao-chen/SRNN.git

now $dt = 0.01\,\mathrm{s}$. 100 of those trajectories are used as training data whereas the other 30 trajectories are used for evaluating the generalisation ability of the model to unseen data. For training the trajectories are further split into overlapping sequences of 10 time steps so that $(1000 - 10 + 1) \cdot 100 = 99100$ sequences of 10 time steps are utilised for training. The long trajectories are generated to evaluate generalisation to longer rollout lengths.

3.2. Generalisation to different chain lengths

For this scenario the generalisation capability of the GNN is examined. The setup of the data is the same as in the SRNN paper [26] to guarantee a fair comparison. Two chain systems are examined. The first system consists of a chain of 20 links with different masses and 21 springs. The second system consists of a chain of 6 links with different masses and 7 springs. For both systems 130 trajectories with varying initial states are generated, 100 trajectories for training and 30 for testing. The masses m and spring constants k are sampled independently from two different normal distributions. Details for reproducibility can be found in the appendix B.1. The masses and spring constants are treated as observed variables and given as input to the models. The performance of the SRNN and a GNN is compared. In addition a simple MLP is compared as baseline. One of each model is trained on each data set belonging to a system. Both, the GNN trained on the 6 link system and the GNN trained on the 20 link system, are evaluated on the 6 link system data and the 20 link system data.

3.3. Generalisation to different systems

For this scenario the generalisation capability to different systems of the examined models is evaluated. To this end 130 different systems of 6 link chain systems and 130 systems of 20 link chain systems are examined. The systems have varying masses m and varying spring constants k. The static parameters are sampled from a Gaussian mixture model with two components as described in the appendix B.2.
The static parameters are provided as input to the models.

3.4. Latent static variables

In this scenario the capability of the models to infer latent variables is examined. One chain system with 20 links is regarded. The static parameters m and k are the same as in experiment 3.2. The static parameters are not observed in this case and instead treated as latent variables.

3.5. Parameter learning

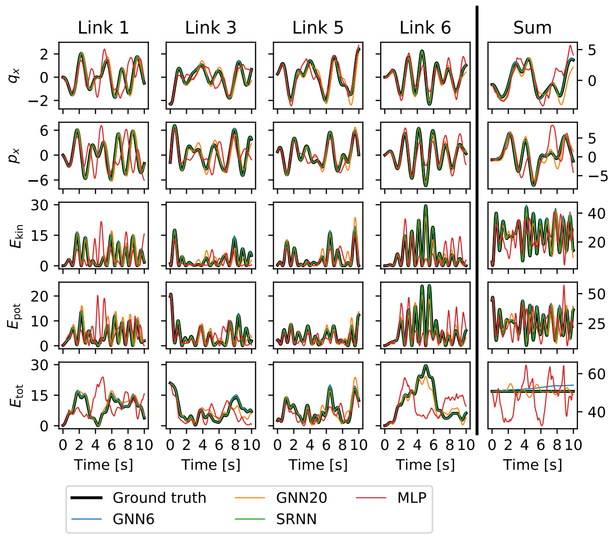
In this scenario the ability of the examined models to explicitly learn the latent variables is examined. The setup from experiment 3.3 is used with 100 different systems of 20 link chains that each generate one trajectory. The static parameters m and k are not observed and treated as the latent variables. The adjacency matrix is also not learned and instead needs to be inferred from the data. For learning the parameters the system identification approach from section 2.6 is applied by adding trainable parameters to the models. Since the parameters of those specific 100 systems are learned the test data set features those 100 systems as well. For each system a new trajectory is generated with new initial states. The performance of the models with the additional identifier is evaluated. In addition the distributions of the learned parameters is compared to the true parameters.

4. Results

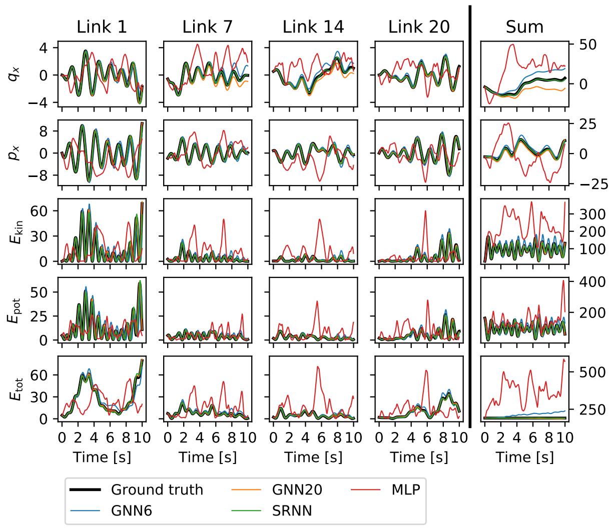
In this chapter the results from the experiments that were proposed in chapter 3 are individually presented and discussed. For all experiments the performance of the models as detailed in appendix A on a 1000 step trajectory is visualised and loss curves are shown. In addition the distributions of the learned parameters are shown for experiment 3.5.

4.1. Generalisation to different chain lengths results

In Fig. 4.1 the results of the experiment can be seen and in Fig. 4.2 the corresponding loss curves. The GNN models and the SRNN model show strong performances with near perfect predictions of the system, whereas the MLP makes worse predictions. While the GNN that was trained on a data set performs better on that data set, the GNNs are also able to make solid predictions on systems with a different number of links, especially in regards to the conservation of energy which is mostly stable in all cases. The GNN trained on the 6 link system shows signs of a slow divergence on the 20 link system. This generalisation is not possible for the baselines models since the inflexibility of MLPs in regards to input size does not allow making predictions for systems of different sizes.

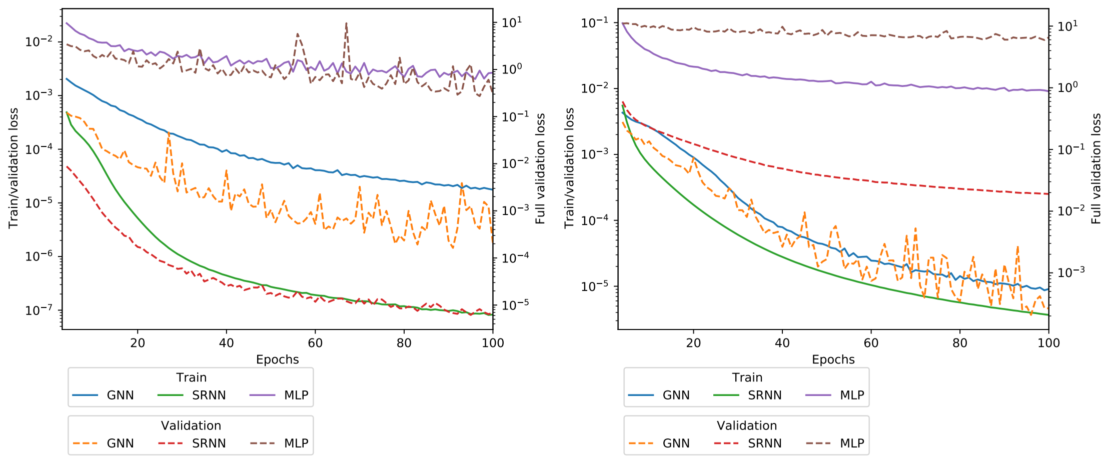


(a) Results on a chain system with 6 links.



(b) Results on a chain system with 20 links.

Figure 4.1.: Comparison of model performance on two chain systems for experiment 3.2. For a subset of the links the position, momentum, as well as the kinetic, potential and total energy is shown. GNN6 refers to the GNN trained on the 6 link system and GNN20 refers to the one trained on the 20 link system.



(a) Training on a chain system with 6 links. (b) Training on a chain system with 20 links.

Figure 4.2.: Learning curves for the models from Fig. 4.1 belonging to experiment 3.2.

4.2. Generalisation to different systems results

In Fig. 4.3 the results of the experiment can be seen and in Fig. 4.4 the corresponding loss curves. The GNNs perform very well and achieve near perfect predictions in terms of energy conservation and deviation from the ground truth positions. When comparing the losses of the GNNs on this task to the losses from the previous experiment shown in 4.2 it can be seen that the models did adapt to the harder data set very well with only small increases in the validation loss. This indicates strong generalisation capabilities to different systems. The SRNN model fell behind and does not manage to adapt to the varying system parameters as good.

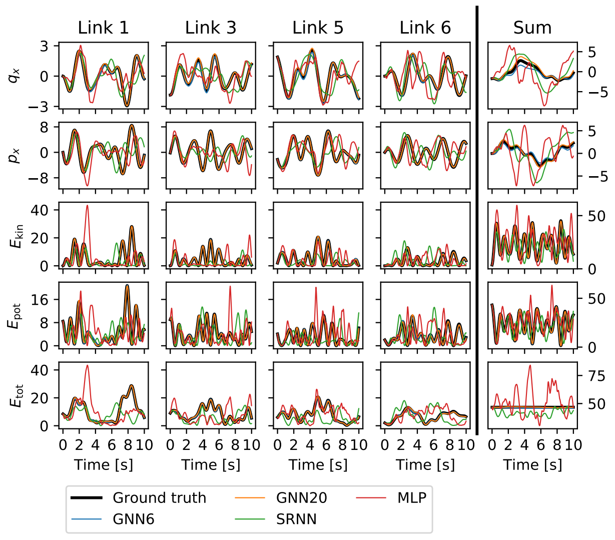
4.3. Latent static variables results

In Fig. 4.5 the results of the experiment can be seen and in Fig. 4.6 the corresponding loss curves. The MLP results are omitted here for a better overview since they not are competitive and distort the axis of the graph. The results in Fig. 4.5 differ strongly from those with a similar setup in section 4.1. The performance of the GNN has diminished whereas the SRNN is still achieving near perfect predictions. This result highlights the drawback of the basic bias GNNs have. Due to the unobserved nature of the link attributes the vertices have no unique identifier so there is no way for GNNs to distinguish between them. Thus no behaviour dependent on the individual link can be learned and instead a general model that tries to fit the dynamics of every vertex must be learned. The MLP based networks retain an order of the links so that it is possible to treat them individually and learn the dynamics based on this information. The results also indicate that the SRNN has no understanding of the meaning of the static attributes of the system since it does not benefit much from observing them. The model in Fig. 4.5 only slightly worse than the one in Fig. 4.1 although it is based on the same system and in this case more input data is given.

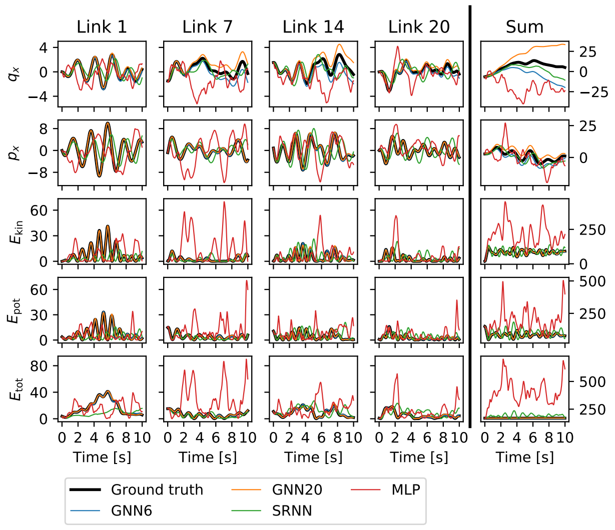
4.4. Parameter learning results

In Fig. 4.7 the results of the experiment can be seen. The combination of learnable parameters for identification of the system parameters with the models allows the GNN model to outperform the SRNN again. The GNN shows strong predictions on the predictions of the links. The predicted energy of the GNN is deviating from the true energy although it after 1000 frames still diverged less than the variance of the energy prediction of the SRNN.

In Fig. 4.8 the distribution of the learned parameters of the GNN is shown. The right column shows the complete matrix of the spring constants. The matrix is sparse due to the connectivity structure of the spring system where each link is only connected to two other links. The middle column shows the matrix after a filter mask was applied that removes the parameters, that are not in the adjacency matrix of the system. Due to the interaction between weights of the neural network and the parameters the scale of the parameters is arbitrary. Because of this only the distribution of the parameter values is examined.

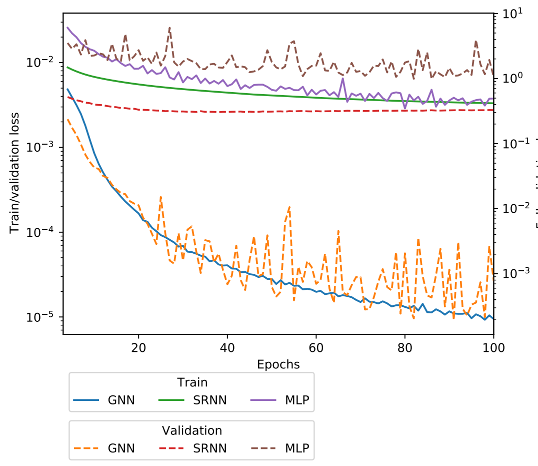


(a) A chain system with 6 links.

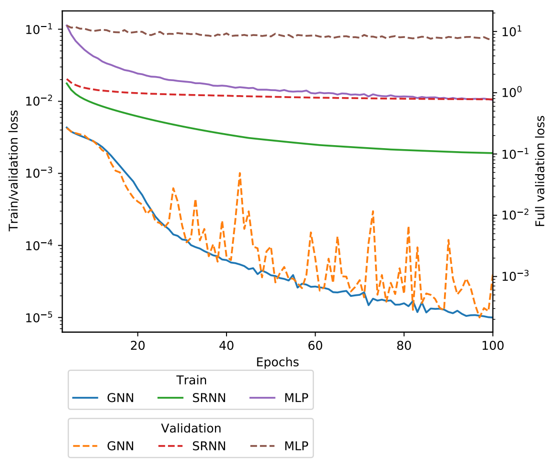


(b) A chain system with 20 links.

Figure 4.3.: Comparison of model performance on two systems for experiment 3.3. For a subset of the links the position, momentum, as well as the kinetic, potential and total energy is shown.GNN6 refers to the GNN trained on the 6 link system and GNN20 refers to the one trained on the 20 link system.



(a) Training on a chain system with 6 links.



(b) Training on a chain system with 20 links.

Figure 4.4.: Learning curves for the models from Fig. 4.3 belonging to experiment 3.3.

The mass parameters show a similar distribution as the ground truth parameters. The distribution is characterised by the two modes which are separated by a gap where there is very little probability mass. In contrast the learned mass parameters of the SRNN and MLP models shown in Fig. 4.9 do not resemble the true distribution and are closer to a unimodal normal distribution.

A similar distribution can be seen for the spring constants. Two strong, separated modes can be distinguished in the middle column. Two additional, smaller peaks can be seen in the positive region. Values from those peaks are found in the places of the matrix where springs connect to the ground, i.e. the first and last element of the matrix. In the right column one can see that the learned matrix is also sparse and is mainly populated by values that are three orders of magnitude smaller than the other values. The symmetry of the matrix is also a property that is approximately preserved through the learning process as can be seen in Fig. 4.10. The values on the two first off-diagonals mirror each other with a mean deviation of less than 1% over all systems as shown in appendix B.3. The non-zero values on the main diagonal show learned self-loops. Adding an additional edge seems to be helpful for learning the self-dynamics.The fact that the distribution of the learned parameter values of the GNN is similar to the true distribution indicates that the model uses those extra parameters for identifying the parameters of the systems.

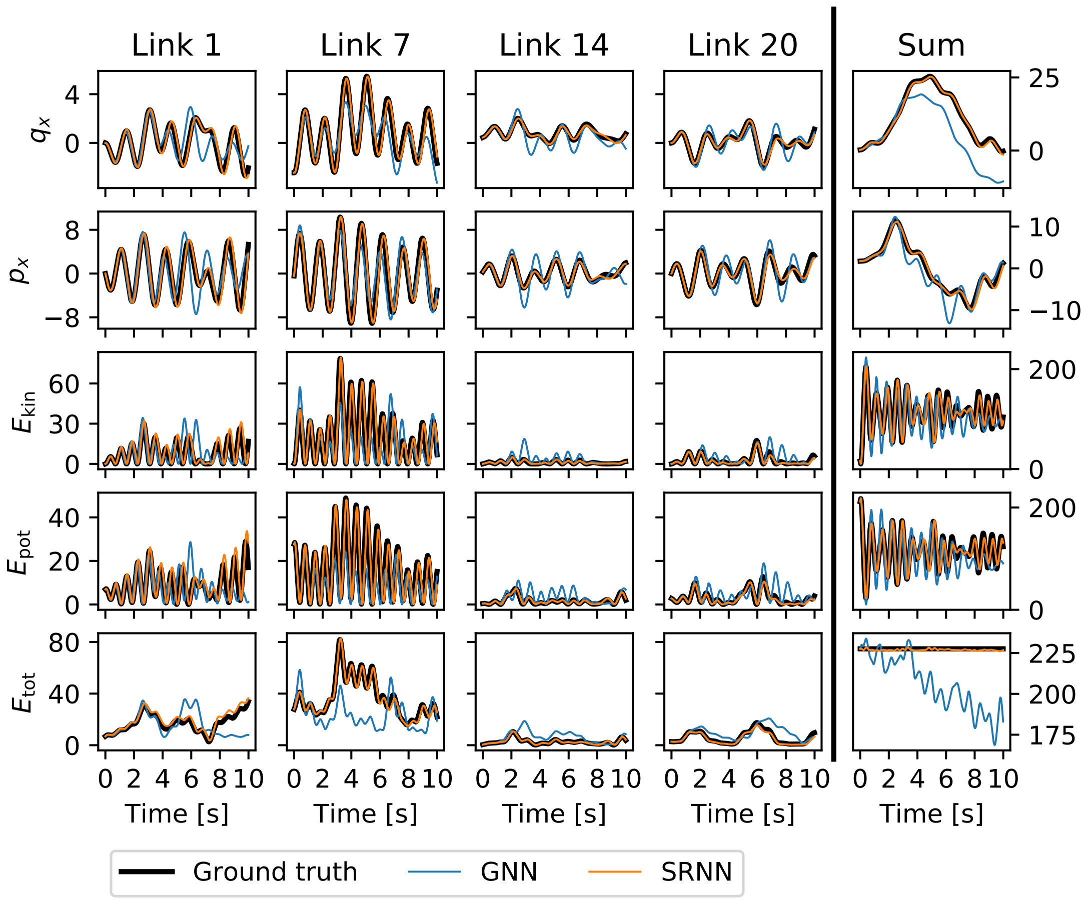


Figure 4.5.: Comparison of model performance on a 20 link system for experiment 3.4. For a subset of the links the position, momentum, as well as the kinetic, potential and total energy is shown.

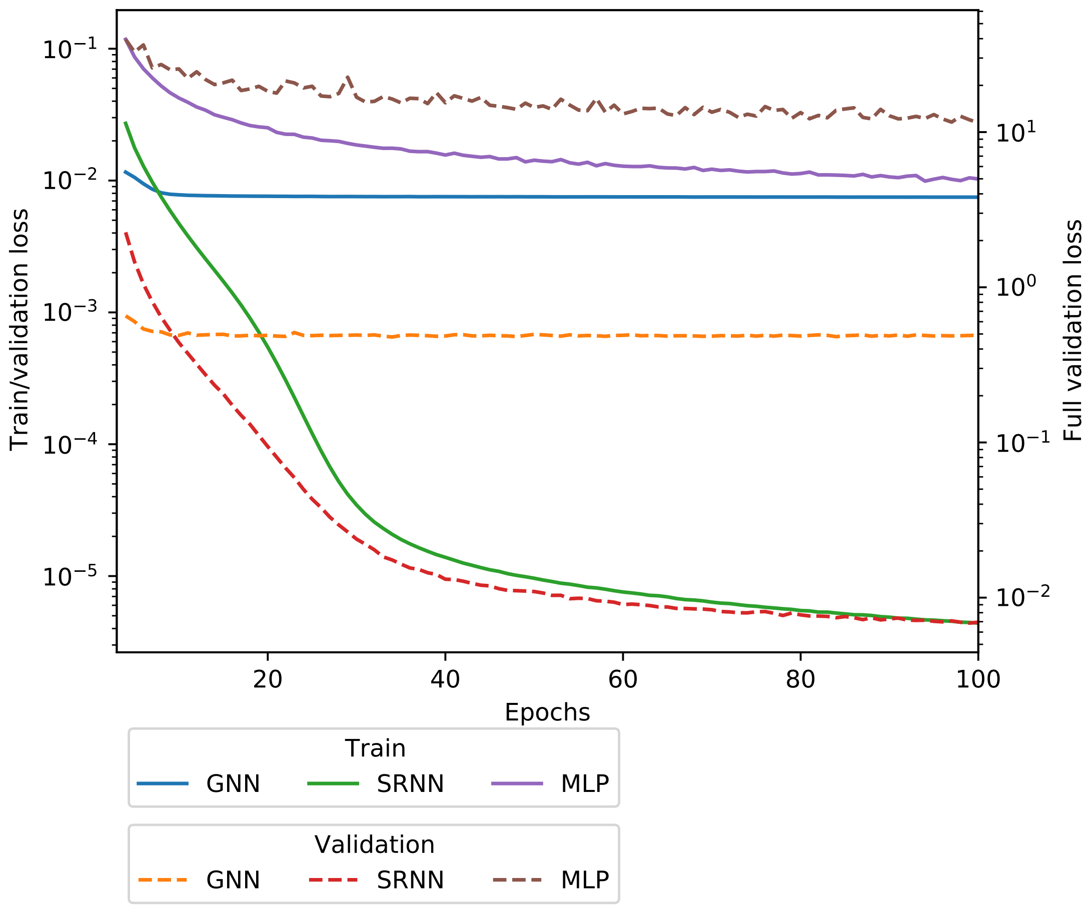


Figure 4.6.: Learning curves for the models from Fig. 4.5 belonging to experiment 3.4.

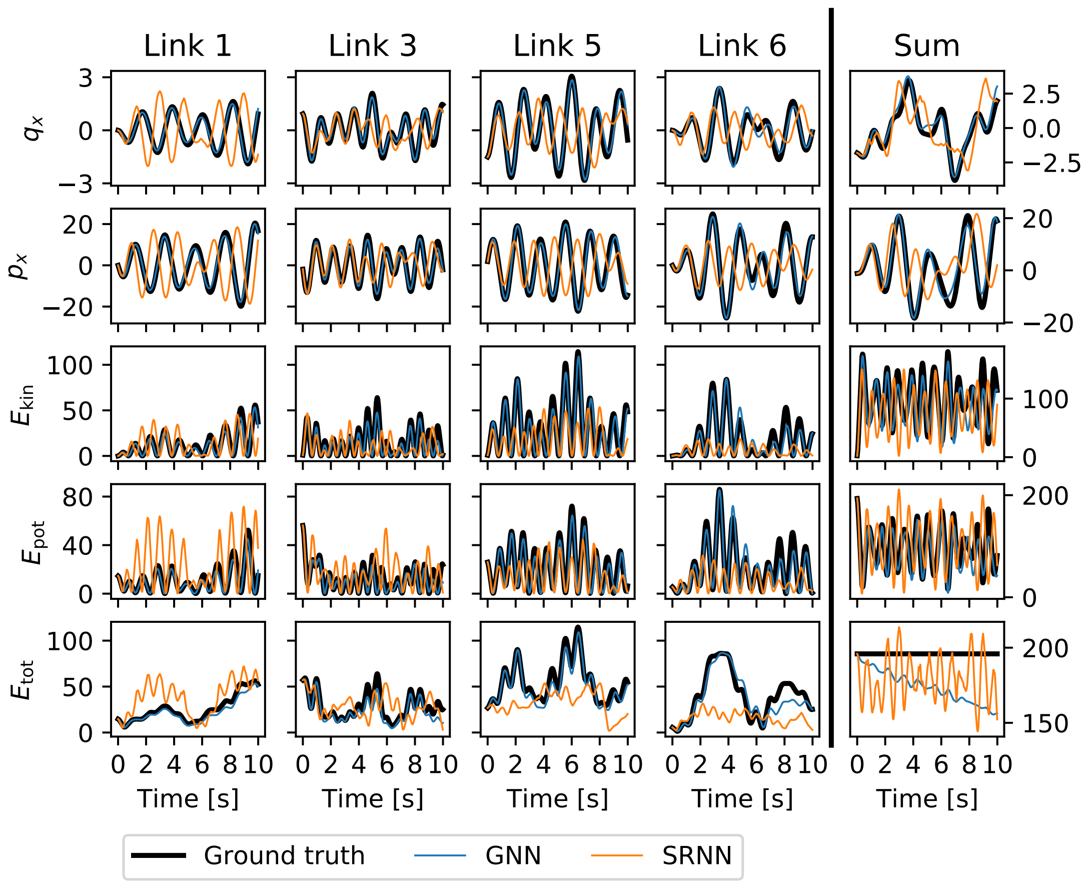


Figure 4.7.: Comparison of model performance on a 6 link system for experiment 3.5. For a subset of the links the position, momentum, as well as the kinetic, potential and total energy is shown.

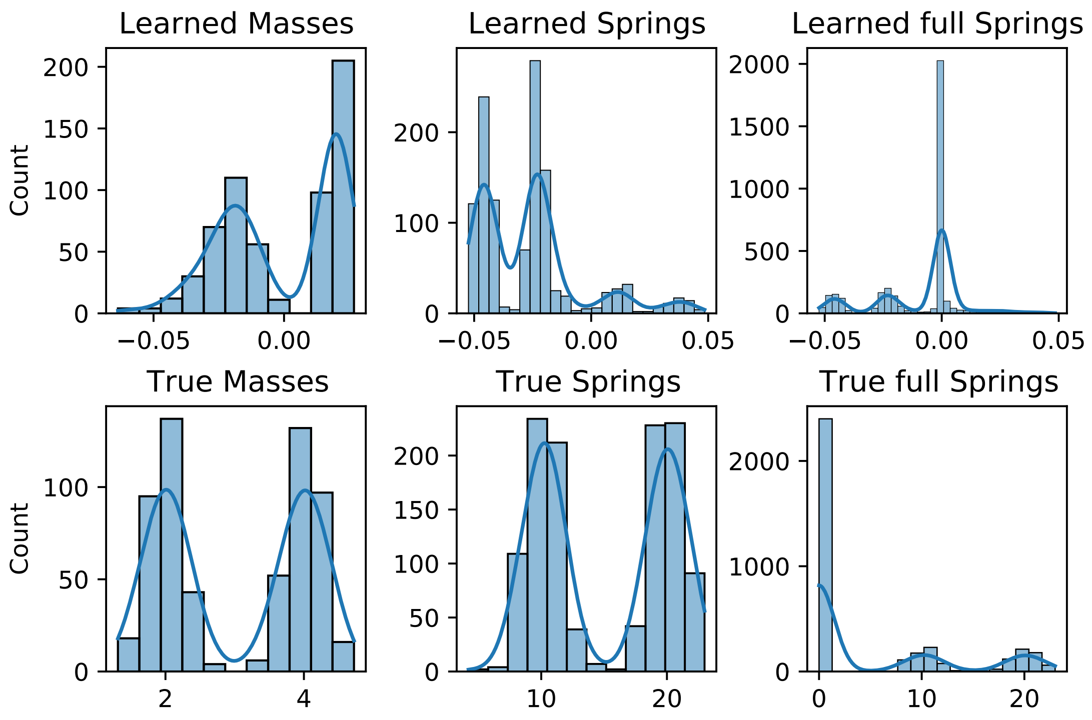
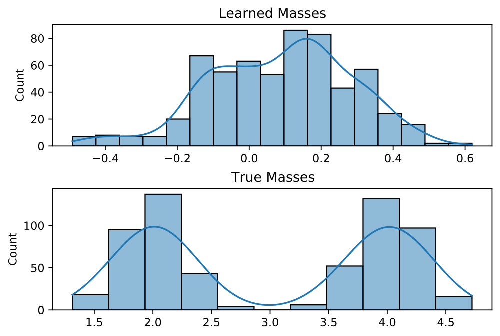
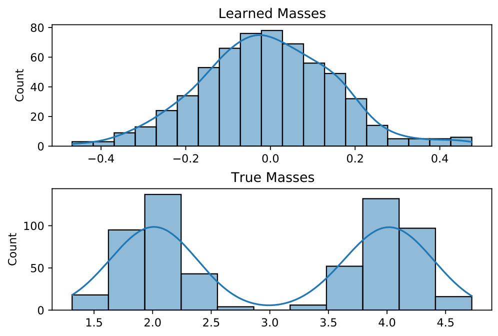


Figure 4.8.: Histogram and estimated probability density of the distribution of the learned static parameters for all systems in comparison to the ground truth of the systems.





(a) Histogram of the learned masses for SRNN. (b) Histogram of the learned masses for MLP.

Figure 4.9.: Histogram and estimated probability density of the distribution of the learned mass parameters for all systems in comparison to the ground truth of the systems.

$$\mathbf{k} = \begin{bmatrix} 19.93 & 9.94 & 0 & 0 & 0 & 0 \\ 9.94 & 0 & 19.80 & 0 & 0 & 0 \\ 0 & 19.80 & 0 & 20.40 & 0 & 0 \\ 0 & 0 & 20.40 & 0 & 11.19 & 0 \\ 0 & 0 & 0 & 11.19 & 0 & 9.73 \\ 0 & 0 & 0 & 0 & 9.73 & 20.12 \end{bmatrix}$$

$$\hat{\mathbf{k}} \cdot 10^2 = \begin{bmatrix} 1.38 & -2.27 & 0 & 0 & 0 & 0.01 \\ -2.29 & -0.04 & -4.5 & -0.01 & 0.01 & -0.01 \\ -0.03 & -4.53 & 2.36 & -4.67 & 0 & -0.01 \\ 0 & 0.02 & -4.71 & 0.43 & -2.57 & 0 \\ -0.01 & -0.03 & 0.04 & -2.56 & -2.04 & -2.22 \\ 0.02 & 0.02 & -0.02 & 0.03 & -2.20 & 1.38 \end{bmatrix}$$

Figure 4.10.: Comparison of the learned spring parameters $\hat{\mathbf{k}}$ for the GNN for the system shown in Fig. 4.7 to the ground truth parameters $\mathbf{k}$. Note the scaling factor for the learned parameters.

5. Discussion

During the research process the search for an appropriate data set was a major hurdle. The data set needed to be accessible for deep learning approaches and scalable in ways that allowed show casing the generalisation capabilities of GNNs. In addition the generating system for this system needed to be simple enough for analytic descriptions to be able to compute the energy terms for the system to evaluate the physical correctness of the models. Earlier experiments were conducted on data sets such as a dynamical billiards system that involved collisions between the billiards and a gravitational n-body problem system. The instantaneous nature of the collisions of the billiards system hindered the learning process and lead to subpar results. A similar phenomenon occurred in the n-body problem system due to the rapid acceleration of bodies that are close to each other that is triggered by the hyperbolic nature of the gravitational potential that governs the system. The usual solution of clipping the gravitational effect was not an option since the physical correctness of the data set needed to be guaranteed. Models that integrated Hamiltonian or Lagrangian formulations as bias performed worse than GNN models without this bias. This effect could not be explained satisfactory. Due to those problems this string of research was discontinued and results from this phase are omitted in this work.

Nevertheless this work has been able to show that GNNs are a valuable class of neural networks with strong generalisation capabilities in regards to the size of the input graph and different system parametrisations. The inductive bias of operating on graph structured data allows the incorporation of prior knowledge in the domain via the connectivity structure of the entities. GNNs are well suited for areas where there is strongly structured data available such as physical systems. Sparse interactions between the entities are common such as in kinematic chains which is beneficial for GNNs since it saves computational cost. In this domain the interpretation of data sets as entities, be it particles or components, that interact seems natural. The same treatment of each entity covers a fundamental concept of the physical world that each entity is governed by the same physical laws and the effect is dependent on the attributes of the entity.

GNNs show this behaviour by being able to learn governing laws on smaller systems and apply those laws to systems of different sizes as was shown in experiment 4.1. In addition they develop an understanding for the importance of the attributes of the entities as shown by the learned parameters that resemble the true parameters as seen in experiment 4.4. If the entities do not provide enough attributes to distinguish between them effectively, GNNs fail as could be seen in experiment 4.3.

6. Outlook

In this work the system identification approach was restricted to learning parameters for a limited amount of systems. Instead of learning parameters, identifiers for those parameters could be learned that generalise to other similar systems. An additional GNN that identifies a system based on a sequence of frames can be employed for this after which longer trajectories can be generated [27]. Due to the low number of attributes of the data set, a system identification based on partial observations was not possible but might be feasible in richer data sets. Either more complex simulations can be chosen such as the ones from the DeepMind Control Suite [28] or real world data may be explored. Working with visual data may be also explored which requires computer vision preprocessing since in this case graph structured data is not directly available. Combining a CNN for the encoding of the visual scene with a GNN for the dynamics prediction and another CNN for the decoding has been done [29] [30]. The intermediate representation that is generated by the encoding network has not been studied in those works and might give interesting insights into the physical scene understanding capabilities of GNNs similar to the structural insights gained from experiment results 4.4. To further research the capabilities of GNNs to simulate physical processes the analogy from edge attributes and forces can be examined [31]. Symbolic regression [32] allows the approximation of complicated, learned MLP based update mechanisms with simpler symbolic equations. With this technique the symbolic approximation to the GNN can be compared to the true generating equations and the relationship between the acting forces in the system and the edge attributes. In addition to the relational bias inherent to GNNs due to the focus on graph structured data, more physics inspired biases may be introduced to improve adherence to physical laws. This can either be done by encouraging the model to learn solutions that fulfil certain criteria via penalty terms in the cost function or by making architecture choices that directly incorporate those biases. If conserved quantities of systems are known, such as total energy and momentum in the case of the used spring chain system, deviations can be penalised.

Classical mechanical systems are governed by differential equations that allow solving in both temporal directions [33]. Due to this property training sequences that are given in the classic forward fashion should be able to be reversed and predicted from the last frame to the first when using a negative time step. Calculating the supervised loss in both directions can help to be more sample efficient as well as depicting the physics more accurately. This approach can be taken one step further by replacing the data based state in the reverse direction by the predicted trajectory. This will lead to problems due to floating point errors and errors of the integration scheme accumulating. Errors of the integration scheme can be reduced by either using higher order integrators from the Runge-Kutta family or using symplectic integrators [34].

The combination of those further research topics, raw visual data as input and adherence to physical laws, is a very important area for smart robotics. If GNNs continue to show promise in those areas, they may become an important building block for making the vision of household robots come true one day.

Bibliography

[1] J. Forlizzi and C. DiSalvo, “Service robots in the domestic environment: a study of the roomba vacuum in the home,” in *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*, pp. 258–265, 2006.

[2] C. Wang, K. V. Hindriks, and R. Babuska, “Active learning of affordances for robot use of household objects,” in *2014 IEEE-RAS International Conference on Humanoid Robots*, pp. 566–572, 2014.

[3] F. Niroui, K. Zhang, Z. Kashino, and G. Nejat, “Deep reinforcement learning robot for search and rescue applications: Exploration in unknown cluttered environments,” *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 610–617, 2019.

[4] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, “A brief survey of deep reinforcement learning,” *arXiv preprint arXiv:1708.05866*, 2017.

[5] E. Todorov, T. Erez, and Y. Tassa, “Mujoco: A physics engine for model-based control,” in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, IEEE, 2012.

[6] B. M. Chang, T. Ullman, A. Torralba, and B. J. Tenenbaum, “A compositional object-based approach to learning physical dynamics,” *ICLR*, 2017.

[7] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, “The graph neural network model,” *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2008.

[8] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, “Learning phrase representations using RNN encoder–decoder for statistical machine translation,” in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, (Doha, Qatar), pp. 1724–1734, Association for Computational Linguistics, Oct. 2014.

[9] Y. Li, R. Zemel, M. Brockschmidt, and D. Tarlow, “Gated graph sequence neural networks,” in *Proceedings of ICLR’16*, April 2016.

[10] P. Battaglia, R. Pascanu, M. Lai, D. J. Rezende, and K. kavukcuoglu, “Interaction networks for learning about objects, relations and physics,” in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pp. 4509–4517, 2016.

[11] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, *et al.*, “Relational inductive biases, deep learning, and graph networks,” *arXiv preprint arXiv:1806.01261*, 2018.

[12] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, and P. Battaglia, “Learning to simulate complex physics with graph networks,” in *International Conference on Machine Learning*, pp. 8459–8468, PMLR, 2020.

[13] G. Corso, L. Cavalleri, D. Beaini, P. Liò, and P. Veličković, “Principal neighbourhood aggregation for graph nets,” *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[14] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, “Graph attention networks,” *arXiv preprint arXiv:1710.10903*, 2017.

[15] D. Bahdanau, K. H. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” in *3rd International Conference on Learning Representations, ICLR 2015*, 2015.

[16] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, “Convolutional networks on graphs for learning molecular fingerprints,” in *Advances in Neural Information Processing Systems* (C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, eds.), vol. 28, Curran Associates, Inc., 2015.

[17] R. Murphy, B. Srinivasan, V. Rao, and B. Riberio, “Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs,” in *International Conference on Learning Representations (ICLR 2019)*, 2019.

[18] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, “Neural message passing for quantum chemistry,” in *Proceedings of the 34th International Conference on Machine Learning* (D. Precup and Y. W. Teh, eds.), vol. 70 of *Proceedings of Machine Learning Research*, (International Convention Centre, Sydney, Australia), pp. 1263–1272, PMLR, 06–11 Aug 2017.

[19] J. Pearl, *Reverend Bayes on inference engines: A distributed hierarchical approach*. Cognitive Systems Laboratory, School of Engineering and Applied Science ..., 1982.

[20] J. Domke, “Parameter learning with truncated message-passing,” in *CVPR 2011*, pp. 2937–2943, IEEE, 2011.

[21] K. Fukushima and S. Miyake, “Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition,” in *Competition and cooperation in neural nets*, pp. 267–285, Springer, 1982.

[22] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, “Backpropagation applied to handwritten zip code recognition,” *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[24] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2015.

[25] K. J. Åström and P. Eykhoff, “System identification—a survey,” *Automatica*, vol. 7, no. 2, pp. 123–162, 1971.

[26] Z. Chen, J. Zhang, M. Arjovsky, and L. Bottou, “Symplectic recurrent neural networks,” in *International Conference on Learning Representations*, 2019.

[27] A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell, and P. Battaglia, “Graph networks as learnable physics engines for inference and control,” in *International Conference on Machine Learning*, pp. 4470–4479, PMLR, 2018.

[28] Y. Tassa, Y. Doron, A. Muldal, T. Erez, Y. Li, D. de Las Casas, D. Budden, A. Abdolmaleki, J. Merel, A. Lefrancq, *et al.*, “Deepmind control suite,” *arXiv preprint arXiv:1801.00690*, 2018.

[29] A. Santoro, D. Raposo, D. G. Barrett, M. Malinowski, R. Pascanu, P. Battaglia, and T. Lillicrap, “A simple neural network module for relational reasoning,” in *NIPS*, 2017.

[30] J. Kossen, K. Stelzner, M. Hussing, C. Voelcker, and K. Kersting, “Structured object-aware physics prediction for video modeling and planning,” in *International Conference on Learning Representations*, 2019.

[31] M. Cranmer, A. Sanchez Gonzalez, P. Battaglia, R. Xu, K. Cranmer, D. Spergel, and S. Ho, “Discovering symbolic models from deep learning with inductive biases,” *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[32] P. Orzechowski, W. La Cava, and J. H. Moore, “Where are we now? a large benchmark study of recent symbolic regression methods,” in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1183–1190, 2018.

[33] W. G. Hoover, *Time reversibility, computer simulation, and chaos*, vol. 13. World Scientific, 1999.

[34] H. Kamberaj, R. Low, and M. Neal, “Time reversible and symplectic integrators for molecular dynamics simulations of rigid molecules,” *The Journal of chemical physics*, vol. 122, no. 22, p. 224114, 2005.

A. Details of model architecture

A.1. Graph Neural Network

For a size of the vertex attribute vectors of n_v and a size of the edge attribute vectors of n_e, the size of the networks can be found in table A.1. As activation function for every layer except the last layer the $\mathrm{softplus}$-function

$$s(x) = \log\left(1 + e^x\right) \tag{A.1}$$

is used. The used integrator is explicit euler. With $n_v = 3$ and $n_e = 5$ for the chain data set this leads to a total of 39074 parameters independent of the number of links in the system.

Function	Layer sizes
ENCODE_NODES	$\begin{bmatrix} n_v & 64 & 64 \end{bmatrix}$
ENCODE_EDGES	$\begin{bmatrix} n_e & 64 & 16 \end{bmatrix}$
UPDATE_EDGE	$\begin{bmatrix} 144 & 64 & 16 \end{bmatrix}$
UPDATE_NODE	$\begin{bmatrix} 80 & 128 & 64 \end{bmatrix}$
DECODE_NODE	$\begin{bmatrix} 64 & 64 & 2 \end{bmatrix}$

Table A.1.: Layer sizes of the used GNN model.

A.2. Symplectic Recurrent Neural Networks

The SRNN architecture uses a Hamiltonian formulation that assumes separability of the kinetic and potential energy of the system. Both components are modelled by a MLP that take either the positions q or the momenta p of all entities as input. Via the Hamilton equations

$$\mathcal{H} = \boldsymbol{T}(\boldsymbol{p}) + \boldsymbol{V}(\boldsymbol{q})$$
$$\dot{q} = \frac{\partial \mathcal{H}}{\partial \boldsymbol{p}}$$
$$\dot{p} = -\frac{\partial \mathcal{H}}{\partial \boldsymbol{q}}$$

the time derivatives of the state variables are calculated using the autograd mechanism. Those derivatives are used for the integrator to calculate the next state. The used integrator is a rk4 integrator.
The two MLPs have layer sizes $\begin{bmatrix} N(n_v - 1) & 2048 & 1 \end{bmatrix}$ where N is the number of links in the system. Then $\tanh$-function is used as activation function. For $n_v = 3$ and $N = 6$ this leads to a total of 57346 parameters and for $N = 20$ to 172034 parameters.

A.3. Multilayer Perceptron baseline

This model directly uses a simple MLP to make predictions for the new state. The layer sizes used are $\begin{bmatrix} N \cdot n_v & 2048 & N(n_v - 1) \end{bmatrix}$. As activation function the $\mathrm{softplus}$-function as in eq. (A.1) is used. For $n_v = 3$ and $N = 6$ this leads to a total of 63500 parameters and for $N = 20$ to 206888 parameters.

B. Details of the data generation

B.1. Details for experiment 3.2

For the experiment 3.2 the normal distribution of the masses has a mean $\mu = 1$ and a variance $\sigma^2 = 0.25$. The normal distribution for the spring constants has a mean of $\mu = 5$ and a variance $\sigma^2 = 1.25$.
The exact values drawn for the masses of the system with 20 links are

$$\mathbf{m} = \begin{bmatrix} 0.84 & 1.02 & 0.89 & 1.13 & 0.77 & 1.25 & 0.68 & 1.01 & 1.28 & 1.25 \\ 1.40 & 0.93 & 1.03 & 1.04 & 1.04 & 0.66 & 0.67 & 0.90 & 1.22 & 0.93 \end{bmatrix}$$

and for the spring constants

$$\mathbf{k} = \begin{bmatrix} 4.64 & 5.25 & 4.15 & 3.12 & 4.95 & 4.80 & 3.72 & 4.83 & 5.13 & 5.81 \\ 4.12 & 4.45 & 5.50 & 4.60 & 6.06 & 3.15 & 5.16 & 7.94 & 4.93 & 5.56 \\ 5.90 & & & & & & & & & \end{bmatrix}.$$

For the system with 6 links the masses are

$$\mathbf{m} = \begin{bmatrix} 1.17 & 1.28 & 1.44 & 0.87 & 0.95 & 0.93 \end{bmatrix}$$

and the spring constants

$$\mathbf{k} = \begin{bmatrix} 5.13 & 6.13 & 4.74 & 3.89 & 4.36 & 3.71 & 8.42 \end{bmatrix}.$$

B.2. Details for experiment 3.3

For the experiment 3.3 Gaussian mixture models with two components are employed from which the static parameters are drawn. The probability for both components is the same. Both components share the same variance of $\sigma^2 = 0.25$ for the masses and $\sigma^2 = 1.25$ for the spring constants. The mean of the first component is $\mu_1 = 2$ for the masses and $\mu_1 = 10$ for the spring constants. The mean of the second component is $\mu_1 = 4$ for the masses and $\mu_1 = 20$ for the spring constants.
The distance between the two modes that show in the distribution is four standard deviations so that there is little overlap between the components.

B.3. Details for results 4.4

The mean deviation of the values of the first two off-diagonals is calculated as $\left(\hat{\mathbf{k}}^T - \hat{\mathbf{k}}\right) / \hat{\mathbf{k}}$ where / is elementwise division. The values are averaged over all systems. For the lower first off-diagonal the values are

$$\bar{\mathbf{k}}_{\mathrm{off-diagonal}} = \begin{pmatrix} 0.87 \\ 0.85 \\ 0.98 \\ 0.97 \\ 0.94 \end{pmatrix} \%$$

and as such all below 1 %.