LUT UNIVERSITY
LUT School of Energy Systems
LUT Mechanical Engineering
BK10A0402 Bachelor’s Thesis

LEARNING MECHANICAL SYSTEMS WITH NEURAL NETWORKS

MEKAANISTEN SYSTEEMIEN OPPIMINEN NEUROVERKKOJEN AVULLA

June 19, 2022

Juho Pitkänen

Examiner: Grzegorz Orzechowski

Supervisor: Grzegorz Orzechowski

ABSTRACT

LUT UNIVERSITY
LUT School of Energy Systems
LUT Mechanical Engineering

Juho Pitkänen

Learning mechanical systems with neural networks

BK10A0402 Bachelor's Thesis

2022

28 pages, 12 figures, 4 tables and 0 appendices.

Examiner: Grzegorz Orzechowski

Keywords: Neural network, artificial intelligence, LUT

Artificial intelligence based tools and applications are becoming more popular in many fields. Especially, use of neural networks has grown rapidly. The objective of this work was to study use of neural network based artificial intelligence tools on solving and understanding mechanical systems. For this work, two different example mechanical system models were created: mass-spring-damper system and deformable flexible multibody system. Example systems were used to generate data to train and test the neural network based data-driven model. These neural network models were specially designed for each example. The goal was to get the neural network tools to predict these systems efficiently and be able to create them with low effort. Programming was done in Python programming language using TensorFlow and Keras packages.

This work showed well how powerful neural networks can be. The results showed that created neural networks were able to predict example systems outputs with good accuracy. However, even with good accuracy in the end, tool might be ineffective if it takes great effort to get it working. This happened with the beam deformation example. In conclusion, usage of neural networks as a tool for modeling mechanical systems is feasible, but great care is needed in case of complex systems. The more experienced the user is at the start, the more effective use of the machine learning tools can be made and more complex systems can be approximated. However, tools like automated machine learning might allow easier development of neural networks even for beginners. It is easy to see these kinds of tools being popular in the future for a large variety of problems. By the results of this work, scientists and students should definitely consider using these neural network tools to understand and solve mechanical models.

TIIVISTELMÄ

LUT-YLIOPISTO
LUT School of Energy Systems
LUT Mechanical Engineering

Juho Pitkänen

Mekaanisten systeemien oppiminen neuroverkkojen avulla

kandidaatintyö

2022

28 sivua, 12 kuvaa, 4 taulukkoa ja 0 liitettä.

Tarkastaja: Grzegorz Orzechowski

Hakusanat: Neuroverkko, tekoäly, LUT

Tekoälypohjaiset työkalut ja sovellukset ovat yleistymässä alasta riippumatta. Erityisesti neuroverkkojen käyttö on kasvanut huimasti. Tämän työn tarkoitus oli tutkia neuroverkkopohjaisten tekoälytyökalujen käyttöä mekaanisten systeemien ratkaisemisessa ja ymmärtämisessä. Työtä varten luotiin kaksi erilaista mekaanista esimerkki systeemiä: massa-jousi-vaimennin systeemi sekä taipuvan palkin malli. Näitä systeemejä käytettiin datan luomiseen neuroverkko työkalujen kouluttamista sekä testaamista varten. Neuroverkko mallit luotiin kullekin esimerkille erikseen. Tavoitteena oli saada neuroverkot ennustamaan esimerkkien systeemejä tehokkaasti, ilman isompia ongelmia kehityksessä. Ohjelmointi osuus toteutettiin python-ohjelmointikielellä käyttäen TensorFlow ja Keras paketteja.

Tämä työ osoitti hyvin, kuinka tehokkaita neuroverkot voivat olla. Tulokset näyttivät, että luodut neuroverkot pystyivät ennustamaan esimerkkien järjestelmiä hyvällä tarkkuudella. Tarkkanakin, työkalu voi lopulta kuitenkin olla tehoton, jos sen luominen vie paljon aikaa. Näin kävi palkin taipumaa ennustavassa esimerkissä. Summattuna neuroverkkojen käyttö mekaanisia järjestelmiä opittaessa kannattaa, jos järjestelmät eivät ole liian monimutkaisia. Mitä enemmän kokemusta käyttäjällä on alussa, sitä monimutkaisempia järjestelmät voivat olla, pitämällä työkalut vielä tehokkaina ja hyödyllisinä. Kuitenkin, automoidut koneoppimis työkalut voivat mahdollistaa neuroverkkojen helpomman kehittämisen jopa alottelijoille. On helppo nähdä, että tällaiset työkalut ovat tulevaisuudessa suosittuja monien eri alojen ongelmien ratkaisussa. Tämän tutkimuksen perusteella tutkijoiden ja opiskelijoiden tulisi ehdottomasti harkita neuroverkko työkalujen käyttöä mekaanisten mallien ymmärtämiseen ja ratkaisemiseen.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS AND SYMBOLS

Abbreviations

AI	Artificial Intelligence
MBS	Multibody system
ML	Machine Learning
MSD	Mass-Spring-Damper
NN	Neural Network
ReLU	Rectified Linear Unit

Symbols

k	Spring constant
c	Damping constant
M	Mass matrix
P^i	Deformation coordinates of flexible body i
R^i	Position vector of reference model
θ^{E^i}	Array of euler parameters
q_f^i	Elastic coordinates vector of flexible body i
u_f^{Ni}	Deformation vector of node *N* in flexible body i
ψ_R^{Ni}	Translations of node *N* in flexible body i

SI- Units

t	time [s]
m	mass [g]
F	force [N]
v	velocity [m/s]

1 Introduction

1.1 Background

Artificial intelligence (AI) is one of the most discussed and researched topics in technology today. The use of AI has been researched for decades, but only in recent times has their development and deployment increased massively. There has been even said to be deep learning revolution on last decade. (Dean, 2019.) The level of existing AI programs enables their effective use for many applications. In recent years, various AI-based programs and applications have become available briskly, regardless of the field. The expertise of developed artificial intelligence has also grown exponentially. The use of artificial intelligence as a modeling tool, especially in the field of technology, seems to be a future trend. In the field of technology, one of the best artificial intelligence tools are neural networks (NN). Complex NNs and other deep learning models did not come into proper use until the 2010s (Dean, 2019, p.1-2). Deep learning, such as neural networks, are used especially to find patterns and models in data. Some pattern recognition problems can be solved using NNs even when they are unsolvable with traditional methods (Eberhart, 2014, p.2). Now, human tier AI, and even more advanced in some applications, already exists. It is only a matter of time before these super artificial intelligences will take over on areas such as field of mechanical engineering. It is not far-fetched that in the future most of the software used by researchers, engineers and students might work on some type of AI (Kevin, 1997, p.17-18). This is the reason why not only computer scientists should look up on neural networks and other types of AI.

1.2 Objectives and limitations

The topic of this work is to study the use of neural networks in understanding and modelling mechanical systems. NN models are tested with two examples made for this work, mass-spring-damper (MSD) system and deformable flexible multibody system. In this work, we will use custom-designed NN models, which are trained with data created by mechanical system solver. These solvers are also custom crafted for this work with Python (Van Rossum and Drake Jr, 1995). After the models are trained, they are used to predict different state of examples mechanical model. Finally, the prediction results are compared with the precise output values of the solvers. Results are then visualized to give better understanding of NNs accuracy. The aim of the work is to study the potential benefits of using NNs to solve these mechanical systems and to determine whether the NNs can be also used effectively for solving other systems of mechanical engineering. The aim is also to optimize NN as easily and accurately as possible. The work will also take a position on the ease of using the method for solving problems and the prospects for the future. In this work there will not be deep tutorials on how NN models are created since this work is not intended to teach creation of NNs. However, this work will give reader a good understanding about using NNs to solve mechanical systems, also other than examples shown.

2 Neural networks and mechanical systems

2.1 Why Neural networks?

Today there are multiple AI frameworks and many ways to create artificial intelligence models to complete a wide variety of tasks. For this work, NN is chosen to be a type of AI to be used especially for its future views and possibilities. Even that working with NN models might sound hard and complex, now there are lots of tools to ease the creation process. It is even possible to avoid most of the programming by using some pre-trained NN model, if found suitable for the project. And of course you could modify and retrain this pre-trained model to suit your needs better. The ability to use pre-trained models might empower the future spreading of NN tools. There has also been increasing effort on developing tools to create NNs with minimal coding. An example of this is Google's AutoML (He et al., 2019). The ability to create powerful NN tools with minimal effort will most likely gather more and more people to this field.

It is fascinating to think that a tool could solve automatically some mechanical system. This tool could only need few parameters as input and the rest would be done by the tool. These input parameters could only be some variables defining the state of the mechanical system. All this would be possible with NNs. Furthermore, NNs have the ability to do deep learning, which might be beneficial on this field, at least with complex mechanical systems. However, this work will not use deep learning as this is only meant to be an introduction to using NNs for mechanical systems.

2.2 Construction of NN and mechanical system models

To create a NN model, we need a Framework. AI frameworks that include NNs are offered in multiple different programming languages. For this work, the following tools were selected: Python programming language (Van Rossum and Drake Jr, 1995), Tensorflow (Abadi et al., 2015) and Keras (Chollet et al., 2015) frameworks. Python being well known general purpose programming language for many scientists and Tensorflow and Keras are both widely used and documented packages of Python for developing, training and testing of AI models. All this is possible since Tensorflow is huge library containing end-to-end possibilities also for NNs. Python programs for this work are mainly written in Jupyter notebook environment. Since NNs need training to be able to predict, this work uses custom-made mechanical systems to generate data. The example systems that generate training data are also created using Python. These systems return the most important data points in a matrix to train NNs. Before training data enters NN, it is also shuffled to improve generalizability. NN model size and architecture will be selected by trying different configurations. If not easily found, automated machine learning (ML) will be used. The model is also optimized to fit by adjusting different parameters like learning rate and number of epochs. Optimization is also done by modifying the loss-function. These adjustments are intended to stop NN to overfit, but still allowing it to learn the model. Overfitted NN does not describe the physically realistic model that should also be seen in the results. Loss-function and validation splitting will punish NN models learning so that there is just the right fit. At the end of the work, accuracy of the NN models are visualized. These results alongside with loss values and time used can be used to determine the actual usefulness of NN.

There are multiple different types of NNs. For this work, the simplest solution is to use fully connected layers to create the model (Battaglia et al., 2018, p.5-8). Fully connected networks search data patterns widely on all the data, which should be enough for this project. Creating a network with fully connected layers creates an architecture called multilayer perceptron. Fully connected layers will be implemented using Keras dense layers. Number of hidden layers and number of nodes inside the layers are being chosen with carefully testing them. The best performance is met with the lowest number of layers and nodes. Therefore, usually it is wisest to start with the smallest possible model (Billings et al., 1992, p. 215; Ramchoun et al., 2021, p. 25). Like in many different parameters on NNs many are found right by testing. Values like learning rate, batch size and number of epochs are few of the parameters that needs to be found right. However, in many cases these values can be left as standard. NNs also need optimizer function to modify models parameters like weights and learning rate on the go. For this work, Adam optimizer was chosen for both examples (Kingma and Ba, 2015). In figure 1 the flow of the creation of NN model is shown in a simple manner.

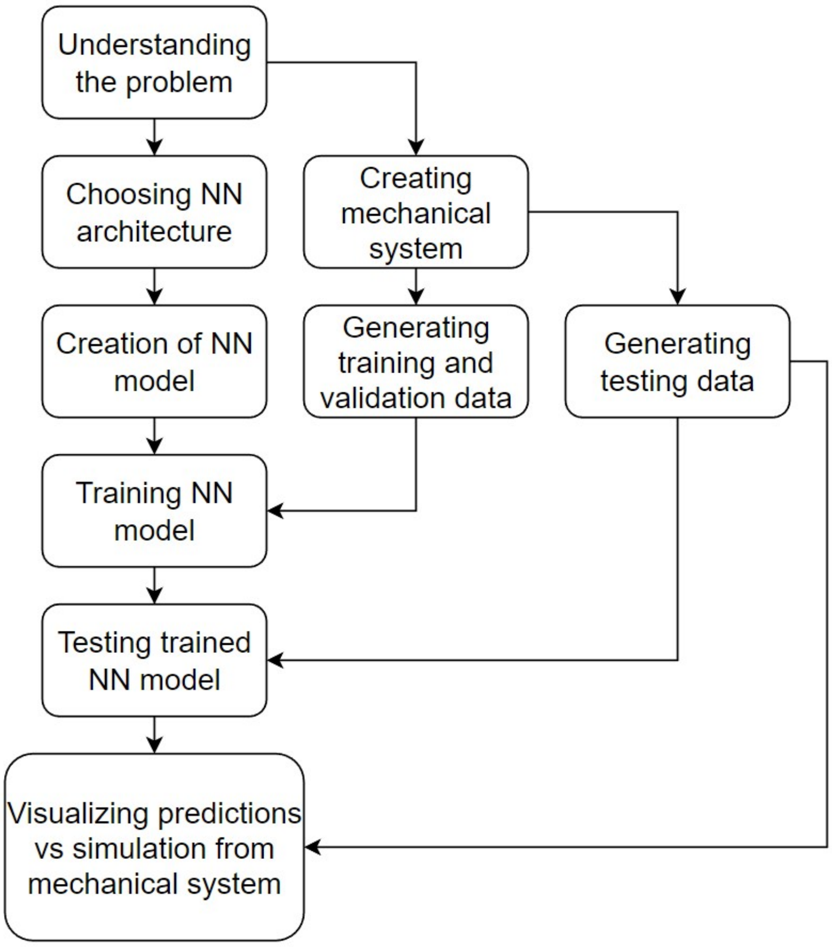

Understanding the problem
Choosing NN architecture
Creating mechanical system
Creation of NN model
Generating training and validation data
Generating testing data
Training NN model
Testing trained NN model
Visualizing predictions vs simulation from mechanical system

Figure 1. Flow of the NN model creation for this work simplified

2.3 Mass-Spring-Damper system

Mass-Spring-Damper system is one of the most used models in mechanical engineering. This is because in many cases, mechanical systems or parts of them can be seen as MSD systems. Therefore, MSD system is excellent way to see if using NNs as a tool is worth it on this field. Mechanical MSD model used for this example is shown below as figure 2.

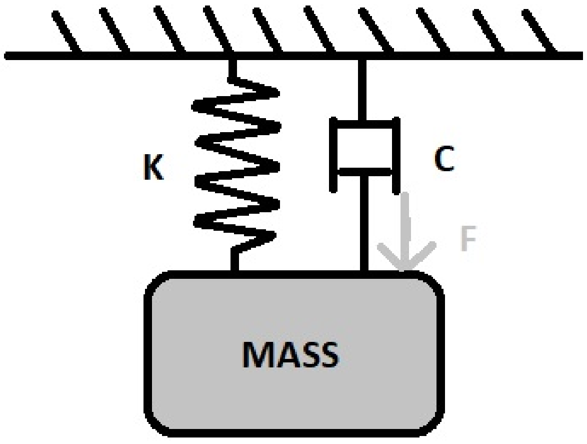


Figure 2. MSD Model visualized

In figure 2 K shows the spring and C the damper. This system includes external force $F(t)$ that is applied to the mass. MSD system can be defined with newtons second law as shown in equation 1. Also m defines the mass, k the spring constant and c damping constant.

$$F(t) - c\dot{x}(t) - kx(t) = m\ddot{x}(t) \tag{1}$$

Position of the mass can be seen from $x(t)$, velocity from the first derivative and acceleration from the second derivative. To solve these equations with general purpose first order ordinary differential equation solver (ODE), velocity will be defined as in equation 2. The function used for solving this work's MSD system can be seen in equation 3. Where x is position and v velocity of the mass. This first order equation is reformulated from equation 1.

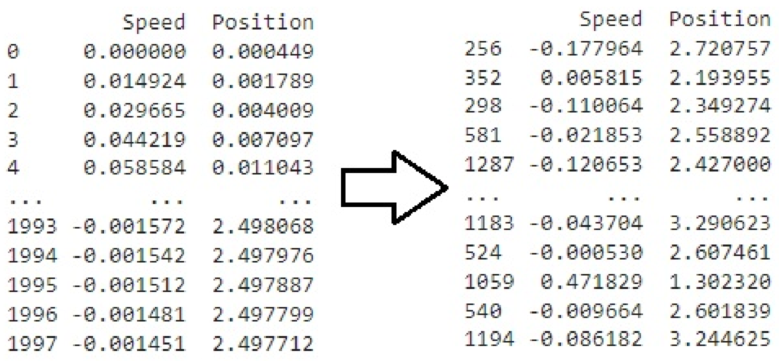
$$\dot{x} = v \tag{2}$$

$$\dot{v} = \frac{F - cv - kx}{m} \tag{3}$$

For this work we want to solve for position x. Solving equation 3 for x gives us equation:

$$x = -\frac{m\dot{v} + cv - F}{k} \tag{4}$$

MSD system for this work is created with Python to generate data for training the NN model. The system can be easily solved using Scipy packets Solve-ivp function (Halvorsen, 2020). This solver will solve position and velocity of mass on each timestep. These calculations are done following previously shown equations. When creating a NN model, it is important to first think what are you going to predict and how? In this example, we want to predict position of the mass. Then we can compare model's accuracy with data created by the generator with different parameters. To predict the position, we need to input enough data for the model to understand the characteristics of the system. In this example, we are going to input velocity and position as they form the state of the MSD system. NN models output will be position shifted by constant time. The shifting is needed on the output, so the model tries to predict the next step of the system. Without time shift, output will just be the same as input. Training and validation data will be then randomly shuffled, which is shown in figure 3



```
         Speed  Position
0     0.000000  0.000449
1     0.014924  0.001789
2     0.029665  0.004009
3     0.044219  0.007097
4     0.058584  0.011043
...        ...       ...
1993 -0.001572  2.498068
1994 -0.001542  2.497976
1995 -0.001512  2.497887
1996 -0.001481  2.497799
1997 -0.001451  2.497712
```

```
         Speed  Position
256  -0.177964  2.720757
352   0.005815  2.193955
298  -0.110064  2.349274
581  -0.021853  2.558892
1287 -0.120653  2.427000
...        ...       ...
1183 -0.043704  3.290623
524  -0.000530  2.607461
1059  0.471829  1.302320
540  -0.009664  2.601839
1194 -0.086182  3.244625
```

Figure 3. Training data that is being fed to NN model is randomly shuffled

The next step is to create the NN model. Since we have quite simple example, number of hidden layers and nodes should stay relatively low. It is a good idea to test the model's performance with different sizes and architectures. Even with only few hidden layers, NNs can solve great amount of problems (Hornik, 1990). NNs also need activation functions to help nodes from diverging. For that, Rectified Linear Unit (ReLU) and softplus functions were found to be best with similar projects. Softplus was chosen to be used due to its smoothness, but offers similar or even slightly better performance than with ReLU (Ramachandran et al., 2017, 7-8).

For this example, it is not necessary to input multiple datasets for the model to get good results. Nonetheless, it will definitely help when testing the predictions on wider scale. On this work, two different datasets combined are used to train the model. These datasets only differed with starting position of the mass. Parameters for generating data for these two datasets are shown in table 1. After training the model with datasets, results are used to predict position of the MSD system with different external force applied to the mass. With different force applied to mass, the system will behave differently, which creates realistic prediction chance. If NN performs predictions well enough, other parameters like external force can be altered for testing runs. At the end, predictions are visualized with position results solved with solve-ivp function.

Table 1. Variables used for solving MSD system

Simulation	Mass	External force	Spring constant (K)	Damping constant (C)	start position
1	20 kg	5 N	2	4	0
2	20 kg	5 N	2	4	0.2 m

2.4 Deformable cantilever beam

Another example we are looking on this work is deformation of a cantilever beam when constant force is applied to the free end of it. Deformations of beams are really known problems for many engineers. Solving deformations on different systems can also be surprisingly complex. There are lots of ways to model and solve the deformations, and it would be interesting to see how NN would do in such an example. These are the reasons why this example is chosen for this work. A simple model of this example is shown in figure 4. Like in MSD example, we are using Python to create a generator function to create training data for the NN model. This time, however, the model is more complex. However, we are still solving equation of motion like in MSD system, only this time on multibody system.

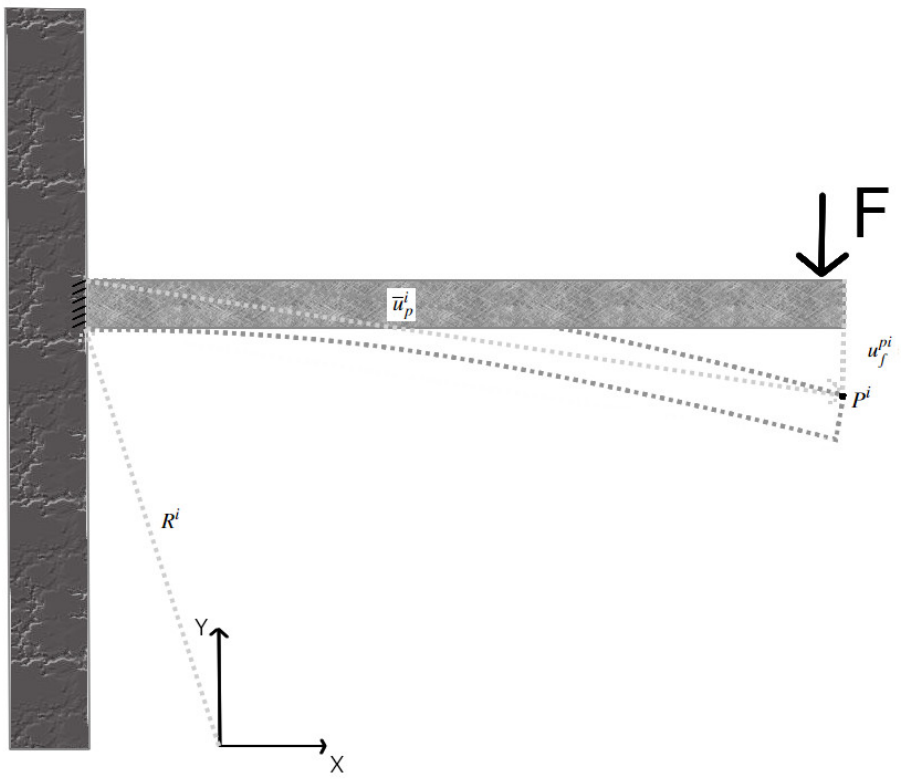


Figure 4. Deformable cantilever beam visualized

For modeling the deformed beam in this example, floating frame of reference (FFR) method is used. With FFR method, flexible body deformations are described by respecting frame of reference (Shabana, 2013). It is efficient and accurate on linear deformation problems, like this one. For this method, a finite element model is firstly created. These models represent a continuous body as a set of finite volumes that are called elements. For this example, we represent the beam with 40 elements. Connecting the elements there are nodes *p*. The first node of the first element is clamped to wall and the last node is where force is applied. One way to solve deformed position of a particle of body i in multibody system (MBS) can be shown as

$$r_p^i = R^i + A^i \overline{u}_p^i \tag{5}$$

Where R^i is reference frame's position vector, A^i is body's rotation matrix and $\overline{u}_p^i$ is sum of undeformed and deformed position vectors of position node p of body i. This sum together tells new position of the node. This position is expressed in the global coordinate system (Korkealaakso, 2009, p.23-25). Body coordinates are represented as follows

$$P^{i^T} = \left[R^{i^T} \theta^{E^{i^T}} q_f^{i^T} \right]^T \tag{6}$$

Where Euler parameters are θ^{E^i}. Elastic coordinates vector are given by q_f^i. For this work, only y-coordinate deformation is needed. This deformation vector u_f^{pi} will output positions that can be then compared between predictions and ones solved by solve-ivp. The deformation vector u_f^{pi} can be solved as shown below

$$u_f^{pi} = \psi_R^{pi} q_f^i \tag{7}$$

Where ψ_R^{pi} contains translations of particle p in flexible body i as modal matrix. To reduce model size, created system uses Craig-Bampton method. With this method, both stiffness and mass are taken in account. Using these equations and methods, the generator will solve the beam system. Data generator will output full state of the model on each time step using multi body dynamics (MBD) calculation. For the model to work broadly, we would want to have multiple datasets to train the model. These datasets are then grouped together and placed as input to NN model. The flow of data can be seen in figure 5. Also, the force applied will be placed on the input, since it will be the one to distinguish multiple datasets from each other. This force array should make it easier for the model to understand differences with datasets.

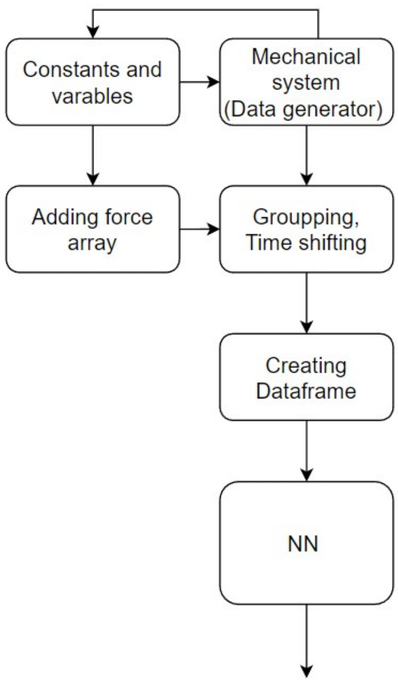


Figure 5. Flow of generated data on deformable cantilever beam example

For simplicity, all values are kept the same except the applied force on the beam. If the NN model gets good results easily, the model can be tested with multiple changing variables. These constant variables used for simulations are shown below

Table 2. Parameters used for calculating deformation of the cantilever beam

Beam length	Thickness	Density	E	Poissons ratio	Elements
1000 mm	20 mm	7801 kg/m	2e11	0.3	40

In figure 6 you can see four simulations plotted whose data are being fed later to the model. Simulation lasts 0.6 seconds, since after that, every simulation's deformations remain constant. Each of these datasets contains *n* a number of time steps of the full state matrix of the beam. The number of timesteps *n* is chosen as low as possible for NN model to still give good predictions. If the number of timesteps is chosen to be too large, it will just make the training take longer with same results as with *n*. The full state matrix contains values like velocities and can be used to calculate many things like rotations on different points. In this example however, we only use the essential parameters to calculate displacement.

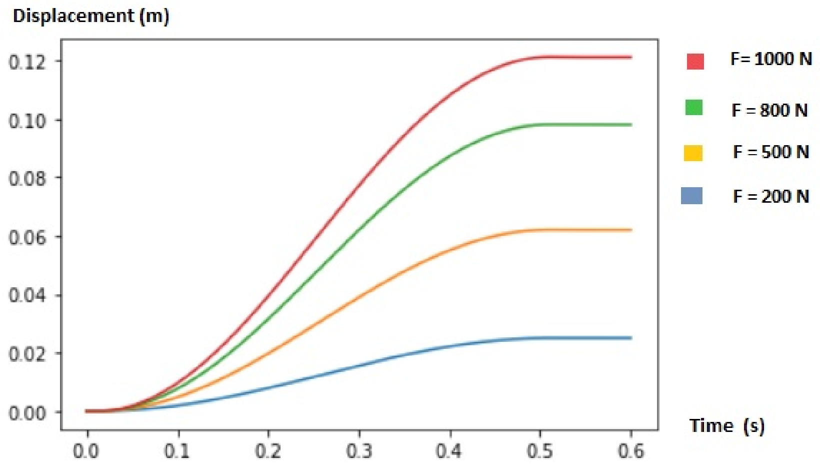


Figure 6. Four simulations that are used to train NN model

The created NN model itself is vastly different than in the previous example. This time we have over fifty inputs and over ten thousand timestep matrices fed to input. These are matrices are again the ones to store data of current timesteps state of the system. Because of the larger scale also the hidden layers will be larger. Still, the number of neurons should be restrained. Too large hidden layers could cause numerical problems. As stated by Heaton (Heaton, 2008, p. 159) ”The number of hidden neurons should be less than twice the size of the input layer.” This rule among the two others given by Heaton create good range to find perfect amount of neurons for hidden layers. For the activation function, ReLU is chosen since it has been said usually to work the best with other dynamical models (Hegedüs et al., 2021). However, softplus activation function will also be tested since like said before it can in some cases work better. This could be because softplus can be seen as a softer version of ReLU (Ramachandran et al., 2017). With thousands of training samples, it could be wise to implement dropout layers between hidden layers. Dropout layers will drop out some of the training samples to fight overfitting.

After the training, the model is used to predict the state matrix with different force applied to the beam. All the other parameters are kept the same for simplicity. Prediction will also output the state matrix, which is then used to calculate deformation position on each time step. To get comparative results we also calculate the state matrix with MBD calculations which is then also used to calculate the deformation with same functions as prediction. The MBD calculated state matrix is the same matrix used as input for prediction. Like in the previous example, results are then visualized by plotting.

3 Neural Networks as a tool for approximating mechanical systems

3.1 Mass-spring-Damper system

In this example, MSD system was created and solved with scipy solve-ivp function (Virtanen et al., 2020). Solved MSD system model gave data for NN to train, validate and later to test. NN model was tested with multiple different configurations, starting from the simplest forms of NN models. Softplus and ReLU activation functions were found to be similar. Main parameters of the best model found are shown in Table 3.

Table 3. Main parameters of working NN model for MSD system

Hidden layers	Epochs	Learning rate	Batch size	Optimizer	Activations
2	150	0.001	32	adam	softplus

The NN model had two inputs, which were velocity and position of the mass. Output of the NN had only the position or in this case the displacement of the mass. After testing with multiple configurations, the model did not need more than two hidden layers. The full NN model structure that was used for MSD system can be seen in Figure 7.

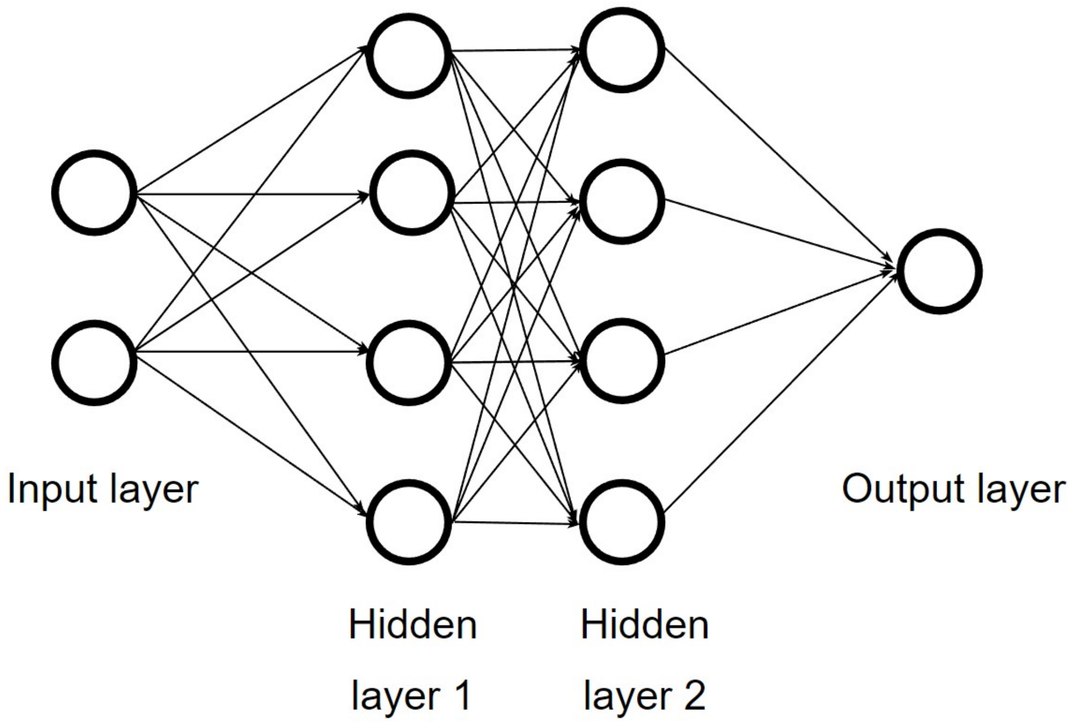


Figure 7. NN model that was used for MSD system

Training NN model was successful, as seen from predictions in figure 8. Model was able to learn the MSD system quickly in few seconds with 150 epochs. The trained model was able to predict the position for different external force inputs. A working network model was easy to find and did not take long to perfect. Since the predicted MSD system is quite simple, NN will produce good results with many different configurations. One big configuration variable encountered was the way of joining different datasets together, which later would be inputted to the model to train it. Efficiently time shifting both sets were causing some problems on creation of this work.

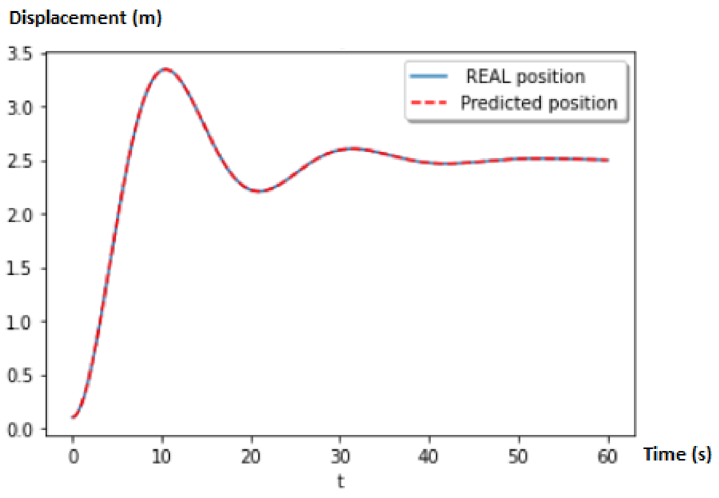


Figure 8. Predicted and real positions of MSD system in chart

Results were not only good with various different external forces inputted, but results were even accurate when the starting position of the mass was altered to the testing set. This shows that NN was able to learn the model enough to understand how different parameters alter the outcome. There are plenty of applications that are associated with spring-mass systems. Therefore, these results gives good understanding of usability of NN tools on simple mechanical models.

3.2 Deformable cantilever beam

The idea of this cantilever beam example was to give a bit of depth to testing NNs on mechanical systems. The First NN model version of this example was created in simple manner. Instead of predicting the full state of the model, it predicted the displacement directly. This version was able to perform predictions accurately. However, this was not a good way to solve this problem in this case because the model required the displacement also as input. This might create a case where the NN model is not really understanding the physics behind the problem, but understand only some correlations between other parameters and displacement. To make the NN model really understand the mechanical system, it should predict the full state of the system. This way NN model predictions could even be used to other values like rotations. Second version being more complex, there were some problems on building the NN. Now there were 50 inputs and outputs versus in MSD system example there was only few. There were problems creating a model that would learn enough from the data but still not being overfitted. Surprisingly, it was easier to find a model that was getting predictions close and didn't take force as an input. The NN model without the force is shown as figure 9.

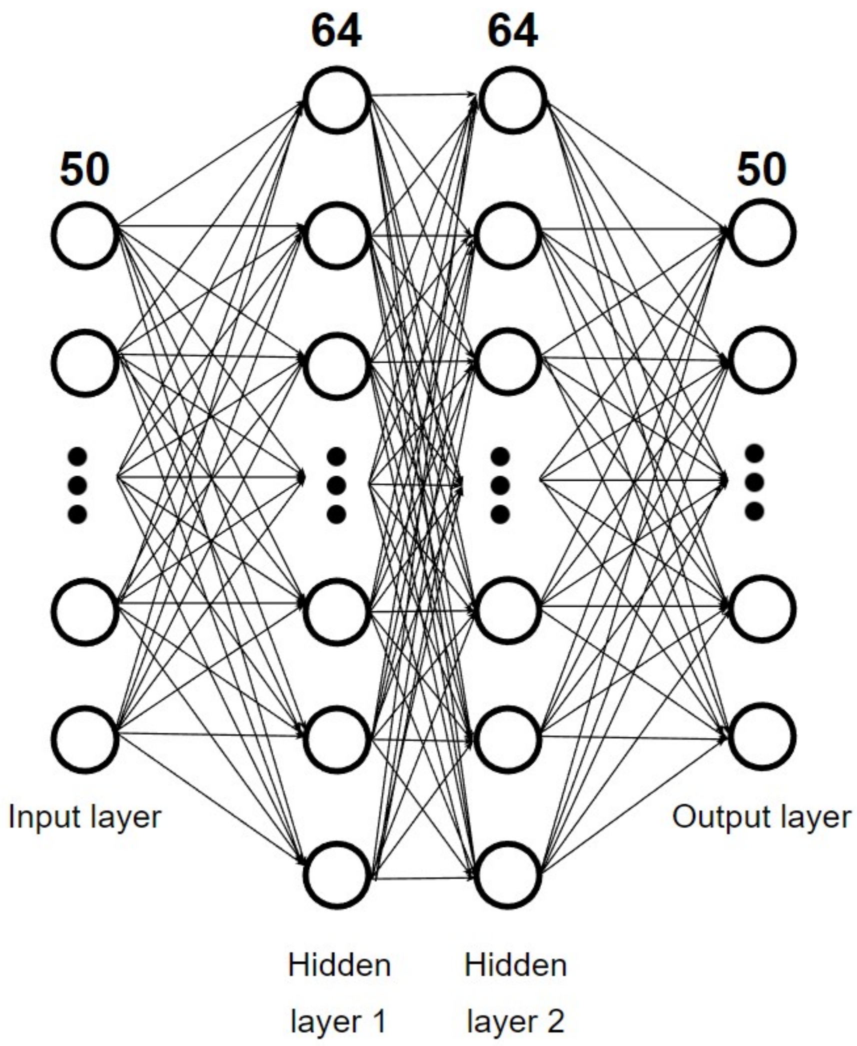


Figure 9. NN model where force was not included

This NN model likely succeeded because the force array was greatly different from other arrays on the input, and getting rid of it unified the whole input. This big difference in input probably made the deformable beam system harder to understand. After multiple tests, it was found that this ”forceless” model was not always giving reliable results. An example of these unreliable results is show in figure 10. These results were not terrible though, and it gave the impression that parameters are approaching the perfect values.

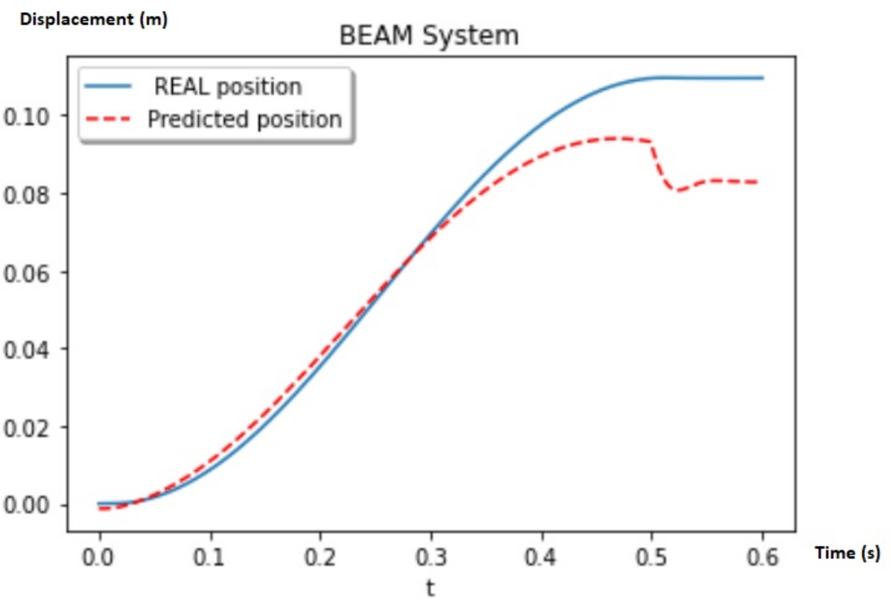


Figure 10. Forceless NN model was not always able to predict accurately

Since the found NN model was not able to predict accurately on every run, newer models were tried. Different models were tried by modifying parameters like learning rate, number of hidden layers and number of neurons on hidden layers. After using tens of hours trying to find the perfect NN model, it seemed better to try something new. One ”newer” way to find an accurately working NN model is to use some type of automated machine learning (ML). The idea of auto ML is to find good configuration for a machine learning model without the user having to do the hard work. For this example, AutoKeras (Jin et al., 2019) named auto ML was tried. Auto-Keras tries different parameters and architectures for NN model and compares those to previous attempts. At the end, the best found model is saved and shown to the user. To find accurate model for complex systems will usually take a lot of time. On the other hand, user can be absent while Auto ML does the job. Using these automated tools could also allow beginners to reach high-end models without having to extensively study the field of ML. For this work with limited time, Auto-Keras didn’t straight find a super accurate model, but it gave a good direction on where one may find one. Before Auto-Keras models that were tried used 50-128 nodes on hidden layers. Auto-Keras showed that even with 32 nodes on hidden layers might be enough. In the end, a successful NN model was found, even normalized. This NN model didn’t need dropout layers. This is mostly because the NN model got so large dataset as input and the model itself was relatively small. With larger input dataset, speed performance of training was obviously worse than in MSD system example due to size difference. In Figure 11 prediction results of the working NN model are plotted.

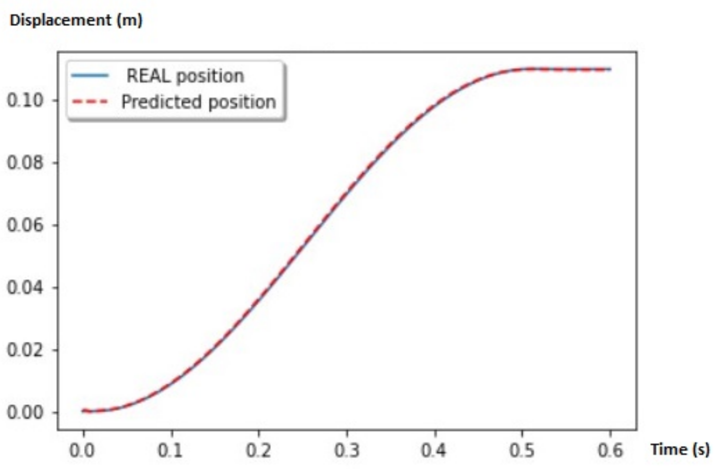


Figure 11. Predicted and generator solved positions of the end of cantilever beam

The best model that was found used two hidden layers with 32 nodes. Models that were tried that had more hidden layers seemed too complex and results were bad. Number of nodes for hidden layers were tested, ranging from 16 to 512. Other parameters used for this NN model are shown in table 4. This final models structure can be seen in figure 12.

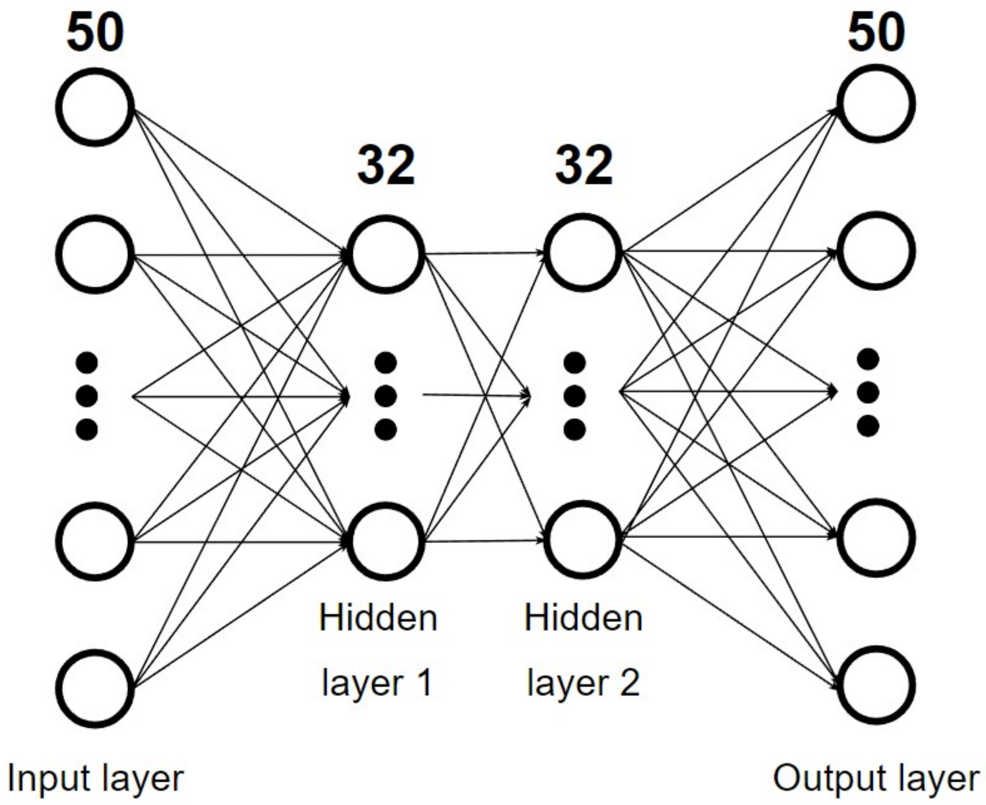


Figure 12. Final NN model for beam deformation example

Table 4. Main parameters of working NN model for beam deformation system

Hidden layers	Epochs	Learning rate	Batch size	Optimizer	Activation
2	100	0.0001	32	adam	ReLU

4 CONCLUSION

This works goal was to see if NNs could give benefits on using them as way to understand mechanical systems. It was also important to see if these NNs would give good results without enormous effort. This work had two examples to test usage of custom-made NNs. These example systems were chosen by their frequent appearance in the field of mechanical engineering. As seen from the results, NNs were able to predict example systems with good accuracy. With MSD system, the model was easy to make and had only few difficulties when being developed. In contrast, however, the beam deformation system caused many problems when creating the NN model. Total time used for the beam example was also vastly more than in the MSD system. NN for beam deformation example had to undergo hundreds of different configuration tries before it started giving accurate predictions. After all, both examples NN models were able to predict results with variety of different inputs. This tells about good understanding on the systems behind the examples. On both examples, NN models that were found best were simple, with only having two hidden layers. You can only imagine possibilities of NN models that have dozens or even hundreds of hidden layers. Despite keeping in mind that the simplest possible network will always be better than a larger one. On this work there was also taste of Automated ML. With Auto-Keras used on last example, it showed that it can be easily useful. With more patience and time, Auto-Keras might have found even better configuration for the NN that was used.

Field of AI is expanding to every sector rapidly. This work shows that today you can create simple NNs with fairly small effort, even with only having a bit of knowledge on NNs and AIs beforehand. This is done possible by easily learnable AI- frameworks and libraries. Although on works like this, previous experience on Python language is beneficial, it seems unnecessary to be able to create NNs like in this work. Actually, more programming is most likely needed on other parts of the work than the NN, like in this work. So, a user without programming knowledge beforehand might face an overall longer project. That said, we come cross to an important factor when using NNs as a tool, Time. Time is one of the key factors when looking on using something effectively as tool or method. Even if some method of solving a problem is really accurate, if it takes absurdly long, in many cases it will get overthrown by a faster yet less accurate method. With MSD example, the NN tool was effective also on the time aspect. However, with the beam deformation example, it took a great time to get it working with good enough accuracy. Therefore, for people without experience in programming, it is not fully recommended using NNs when solving complex problems. Of course also inexperienced people might get more complex projects to work with luck, but possibility of errors will just rise when projects get harder. More errors corresponds to more time used on project. Where is the line then? What is too complex for inexperienced scientists? Still, NNs should be looked as a massive opportunity. After every project, users expertise on NN and other programming will just increase and that will allow efficient usage of NNs on more complex systems next time. Also, nowadays most students, researchers, engineers and even hobbyists already have experience on some kind of programming language. This work suggests every scientist and student to consider using NN based method on learning different systems. Most of the scientist and researchers will likely do this or use some other type of AI in the future, if not already.

LIST OF REFERENCES

Abadi, M., et al. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. url: `https://www.tensorflow.org/`. Software available from tensorflow.org.

Battaglia et al. (2018). Relational inductive biases, deep learning, and graph networks. *Graph networks*, pp. 5–8. doi:10.48550/arXiv.1806.01261, url: `https://arxiv.org/abs/1806.01261`.

Billings, S.A., JAMALUDDIN, H.B., and CHEN, S. (1992). Properties of neural networks with applications to modelling non-linear dynamical systems. *International Journal of Control*, 55(1), pp. 193–224. doi:10.1080/00207179208934232, url: `https://doi.org/10.1080/00207179208934232`.

Chollet, F. et al. (2015). *Keras*.

Dean, J. (2019.). The Deep Learning Revolution and Its Implications for Computer Architecture and Chip Design. *arXiv*. doi:10.48550/arXiv.1911.05289.

Dean, J. (2019). The Deep Learning Revolution and Its Implications for Computer Architecture and Chip Design. *arXiv*. doi:10.48550/arXiv.1911.05289.

Eberhart, R. (2014). *Neural Network PC Tools: A Practical Guide*. Elsevier Science. ISBN 9781483297002.

Halvorsen, H.P. (2020). Mass-Spring-Damper System with Python. *MSD*. url: `https://www.halvorsen.blog/documents/programming/python/resources/powerpoints/Mass-Spring-Damper%20System%20with%20Python.pdf`.

He, X., Zhao, K., and Chu, X. (2019). AutoML: A Survey of the State-of-the-Art. *arXiv*. doi:10.1016/j.knosys.2020.106622.

Heaton, J.T. (2008). *Introduction to Neural Networks for Java, 2nd Edition*. Heaton Research, Inc. ISBN 1604390085.

Hegedüs, F., Gáspár, P., and Bécsi, T. (2021). Fast Motion Model of Road Vehicles with Artificial Neural Networks. *dynamicalex*, pp. 12–17. doi:10.3390/electronics10080928, url: `https://www.mdpi.com/2079-9292/10/8/928`.

Hornik, K. (1990). Approximation Capabilities of Muitilayer Feedforward Networks. *hornik*, p. 2. url: `https://web.njit.edu/~usman/courses/cs677_spring21/hornik-nn-1991.pdf`.

Jin, H., Song, Q., and Hu, X. (2019). Auto-Keras: An Efficient Neural Architecture Search System. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1946–1956.

Kevin, G. (1997). *An Introduction to Neural Networks*. Routledge; 1st edition (5 Aug. 1997). ISBN 1857285034.

Kingma, D.P. and Ba, J.L. (2015). ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION. *adam*. doi:10.48550/arXiv.1412.6980, url: `https://arxiv.org/abs/1412.6980`.

Korkealaakso, P. (2009). *REAL-TIME SIMULATION OF MOBILE AND INDUSTRIAL MACHINES USING THE MULTIBODY SIMULATION APPROACH*. Master's thesis. LUT University, 23-25 p.

Ramachandran, P., Zoph, B., and Le, Q.V. (2017). SEARCHING FOR ACTIVATION FUNCTIONS. *Activations*, pp. 1–9. doi:10.48550/arXiv.1710.05941, url: `https://arxiv.org/abs/1710.05941`.

Ramchoun, H., et al. (2021). Multilayer Perceptron: Architecture Optimization and Training. url: `https://reunir.unir.net/bitstream/handle/123456789/11569/ijimai20164_1_5_pdf_30533.pdf?sequence=1`.

Shabana, A.A. (2013). *Dynamics of multibody systems*, 4th edn. New York: Cambridge University Press.

Van Rossum, G. and Drake Jr, F.L. (1995). *Python tutorial*. Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands.

Virtanen, P., et al. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17, pp. 261–272. doi:10.1038/s41592-019-0686-2.