Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Simulation of muscle movements with GNNs

Niklas Kleinhans

Course of Study:	Informatik
Examiner:	Prof. Dr. rer. nat. Kurt Rothermel
Supervisor:	Michael Schramm, M.Sc.
Commenced:	January 27, 2022
Completed:	July 27, 2022

Abstract

In the last few years Graph Neural Networks (GNNs), a family of neural architectures used for irregularly structured data, gained a rising interest of the graph processing research community. Previous work has shown the potential of GNNs in simulations. In the field of bioinformatics the simulation of human body movements is of great interest. State of the art approaches used for simulations, like the Finite Element Method (FEM), are very time and resource consuming. The human body, in particular the muscle system, can be represented in a graph structure. This offers a great potential of applying Deep Learning approaches for graph structures on muscle data to optimize the simulation.

This work presents a proof of concept to apply GNNs on muscle data and find effective network properties. A Deep Learning Feed Forward Neural Network combined with a Graph Convolutional Network (GCN) is used to learn the deformations of the muscles. The model can predict every node inside the muscle. The results are compared to an existing approach which is using a Feed Forward Neural Network to predict a fixed set of nodes inside a muscle.

Zusammenfassung

In den letzten Jahren ist das Interesse an Graph Neuronalen Netzen (GNNs) in dem Forschungsbereich zur Graphverarbeitung immer weiter gestiegen. GNN sind neuronale Architekturen die zur Verarbeitung von unregelmäßig strukturierten Daten verwendet werden. Vorhergehende Arbeiten haben das Potential von GNNs für Simulationen, die sonst beispielsweise durch Finite Elemente Methoden umgesetzt werden, gezeigt. Im Bereich der Bioinformatik ist das Simulieren von Bewegungen des menschlichen Körpers von großem Interesse. Aktuelle Verfahren, wie die Finite Elemente Methode, die oft für Simulationen angewendet werden, sind oft sehr zeit- und ressourcenaufwändig. Der menschliche Körper, im speziellen das Muskelsystem, kann durch eine graphstruktur dargestellt werden. Das bietet ein großes potential Deep Learning auf Muskeldaten anzuwenden um die Simulationen zu optimieren.

In dieser Arbeit wird ein Proof of Concept vorgestellt um GNN auf Muskeldaten anzuwenden und effektive Netzwerkkonfigurationen zu finden. Ein Deep Learning Feed Forward Neuronales Netz kombiniert mit einem Graph Convolutional Network wird verwendet um die deformation eines Muskels zu lernen. Das Model kann jeden Knoten in dem Muskel auswerten. Die Ergebnisse werden mit einem bestehend Ansatz, in dem ein Feed Forward Neuronales Netzwerk zum vorhersagen von einer fest definierten Knotenmenge verwedet wird, verglichen.

Contents

List of Figures

List of Tables

Listings

Acronyms

Adam Adaptive Moment Estimation. 32

CNN Convolutional Neural Network. 7

ConvGNN Convolutional Graph Neural Network. 16

FEM Finite Element Method. 3

GCN Graph Convolutional Network. 3

GEN Graph Element Network. 7

GNN Graph Neural Network. 3

MPNN Message-Passing Neural Network. 20

MSE Mean Squared Error. 8

NN Neural Network. 18

PDE Partial Different Equation. 15

ReLU Rectified Linear Activation Function. 30

SFT Spatial Function Transformation. 20

1 Introduction

Understanding the human body in particular the muscle activities regarding human body movements is a major field of research in bioinformatics. In order to make research on well structured and meaningful data possible, in the last years many different simulation techniques were introduced like neuromusculoskeletal tracking to simulate forward simulations of human movements [SP07] or numerical calculations by using the FEM [AG09], just to mention few of them.

Due to the rapid growth of hardware and computing power the simulations became better and faster, but they are still bounded in calculation time and high resource costs. This fact makes it difficult to use these kind of simulations in real time applications or anything similar.

Due the increasing computational power although deep learning approaches found there way into different kind of applications. The great advantage of deep learning is that the most approaches needs a lot of computational power while training but not on inference. They can map the input of a function without the knowledge of any mathematical formulation [MMC20]. Therefore it is possible to predict fast without a huge amount of hardware [Ayo10].

The simulation of the muscle system is part of continuum-mechanical simulations and is a very resource-consuming calculation. To benefit from the advantages of machine learning many approaches have been applied to simulate muscle movements. Deep Learning approaches are very successful on different kind of research fields, for example image classification, speech recognition or object detection. As well as for graph structures as used on e-commerce data to represent the interactions between users and products [WPC+21]. This promising result in the field of graph structures leads to new ideas of using Deep Learning architectures on these structures. As an example the muscle system of the human body and every muscle itself can be represented in some kind of graph as well. This leads to the assumption that Deep Learning can be applied to simulate body movements.

The work of Alet et al. [AJB+19] has presented an approach for solving PDEs using GNNs. GNNs is a family of neural architecutre for irregularly structured data, like graph representations. Therefore this work is a promising approach for optimizing muscle movement simulations. In their experiments on simple 2-dimensional Partial Different Equation (PDE) problems they achieved promising results.

In this work, a proof of concept is presented to verify if graph neural networks can be applied to muscle data by using the approach of Alet et al. [AJB+19] to learn the deformation of the muscle due the human body movement. To do this a FEM based simulation is used to generate a set of train data, containing a activation-driven musculoskeletal system representation of the upper limb. Every activation represents the state of the upper limb system. This data is encoded into a higher dimensional space and brought into a graph structure. With the use of a GCN a global solution is derived and decoded into the output space with the final node state prediction. The resulting Model can predict the deformation for every node inside the muscle.

The concept is compared to a Deep Learning approach which uses a Feed Forward Neuronal Network to predict the new node positions of a fixed set of nodes. Different variations of hyperparameter settings are tested to get a nice understanding of the model efficiency.

1.1 Thesis Organization

This work is structured as follows. Chapter 2 introduces all the background knowledge required for this work. Including the idea behind GNNs, and the Convolutional Graph Neural Network (ConvGNN) used in this work. Furthermore the underlying core technique used to generate train data is explained.

In Chapter 3 the problem we designed a concept for is described in detail and the challenges this work deals with are introduces.

Chapter 4 presents related work using Machine Learning approaches in the field of bioinformatics.

In Chapter 5 the concept of using GNNs for simulating the deformation of muscles is described.

Chapter 6 contains different experiments applied on muscle data by using the concept. The results are discussed and compared to another comparable Machine Learning approach.

Chapter 7, the last chapter, concludes the work with a summary of its result as well as possible ideas for future work.

2 Background

To implement the concept presented in this work, different approaches in the field of simulation, bioinformatics and Machine Learning are applied. Therefore, in this chapter some helpful background knowledge is described, especially the concept behind GNNs and the current common approaches in simulating large nonlinear distortions of three dimensional problems.

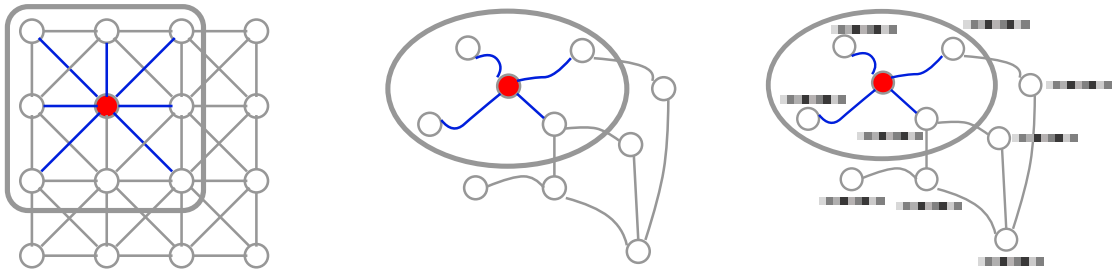
2.1 Finite Element Method (FEM)

The application of the FEM can be found in many different fields of complex problem definitions. It is used for solving problems in the field of aircraft construction, ship building, for solving heat conduction or fluid mechanic problems as well as for simulation of muscle deformations in the field of bioinformatics, just to mention a few areas of applications [Wri13].

In general the FEM is an approximation method. It finds a numerical solution of a PDE allowing the division of a large complex system into smaller simpler parts. These parts are called the finite elements. An example might be a large 3-dimensional object divided into a finite set of tetrahedrons. After dividing the system into a finite set of elements, the whole system is modeled with the set of approximation functions of every finite element. Finally the new system of equation is solve with the FEM approximation due minimizing an error function [KW91].

Many simulations are realised in a numerical way by solving the underlying PDEs, hence FEM solver are often used to simulate different problems. The core Challenge to simulate complex objects is the time and resource-consuming calculations behind FEM solvers. One reason for the complexity of solving FEMs is the high amount of possible parameters of every approximation function. There are several techniques to reduce the complexity and solving the approximation faster, but for example in real time applications these approaches are still too slow and resource-consuming [MMC20].

Specifically the calculation, in the field of continuum-mechanics that deals with the motion of deformable objects in response to external manipulation, is very complex. This makes it very difficult to use FEM simulation for analytical evaluations [Wri13]. For this reason the research community is very interested in the use of Machine Learning approaches for continuum-mechanics simulations. To apply one of this approaches the data of the upper arm muscles are trained with an architecture of different Machine Learning techniques. The training data for optimizing the parameter in the Machine Learning architecture is generated out of a simulation with FEM solvers. The simulation evaluates a set of data points in an euclidean space with coordinate information representing a muscle structure. This data points and the relations between them can be represented in a graph. The structure of a graph and how Machine Learning can be applied on this kind of data representation is explained in the next chapter.



(a) Images Structure **(b)** Graph Structure **(c)** Graph with Features

Figure 2.1: The illustration of a graph represented in different forms. In the first illustration 2.1a the graph has a fixed structure equal to a graph representation of an image. figure 2.1b shows an unstructured graph representation. The last figure 2.1c shows the node enrichment with an feature vector. Every figure contains an illustration of a convolutional step [WPC+21]

.

2.2 Graph Neuronal Networks (GNN)

The number of applications, in which data is represented in the form of graphs is increasing. That is why there is a rising interest in finding new solutions solving tasks on graphs. Amongst others, there are promising approaches in the field of Deep Learning [WPC+21].

In general a graph $G(V, E)$ is defined by a set of nodes (V) and edges (E). Every node is connected to another node with an edge. The core challenge of applying Deep Learning methods on graph data is the irregular structure, the occurrence of different variations with undefined numbers of edges and nodes and the sometimes existing correlation between nodes and spatial locations [AJB+19].

An example for a very simple graph structure is an image represented as some type of graphs as shown in figure 2.1a. Every pixel represents a node and the nearby pixels are connected with edges. The properties of this image graph structure is that every neighbor of a node is ordered and has a fixed size. This property allows reusable functions to be applied iterative on the dataset. This is exactly what happens in the so called CNNs.

CNN is a common architecture in Image Processing to classify images by reducing the complexity and dimension of an image. In most cases this is done with a combination of convolution, pooling and fully-connected Neural Network (NN) layers as shown in figure 2.2. The convolutional layer uses learnable kernels or filters with fixed sizes. These filters are applied on every pixel and calculate a new corresponding value based on the current and neighbor pixel values. An example would be applying a $3x3$ Matrix (called the filter) on every pixel and its neighbors across each channel and calculating the weighted average [WPC+21]. A channel extra dimension in the grid tensor. For example in an image this is in the most cases in 3-dimension for an rgb representation. The pooling simply reduces the dimensionality for example by using the max value of 4 neighboring pixels. Both layers just work in case of a fixed graph structure as shown in Figure 2.1a [ON15].

As mentioned before, in contrast to the calculation on this simple graph structure as done with CNN, many graph structures are unordered and can vary in their sizes as shown in figure 2.1b. To handle this kind of graph structures new approaches have been defined like GNNs.

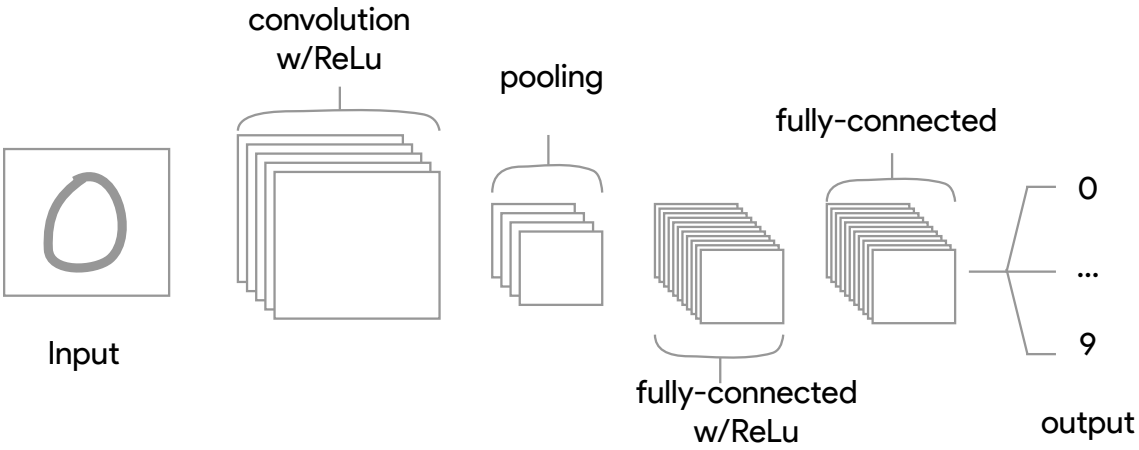


Figure 2.2: Architecture of a CNN with a convolution, pooling and fully connected network. [ON15]

GNNs are applied in settings where nodes can be objects or part of an object [AJB+19]. There are different kind of GNNs. In this work we are focusing on spatial-based ConvGNNs.

2.3 Convolutional Graph Neuronal Network (ConvGNN)

Similarly to the idea of convolutions in a CNN the spatial-based graph convolutions convolves the central nodes representation with its neighbors representation to derive the updated representation for the central node as shown in figure 2.1b [WPC+21]. The output of the ConvGNN in general is the same graph with the influence of the neighbor nodes calculated in every node. It brings the convolution steps as done in CNNs from grid data to graph data Wu et al. [WPC+21].

In figure 2.1c is shown that every node in the input can be represented with a vector containing some metric informations, mostly called a feature vector. Calculation on this structure is done in different ways. This depends on the goal of the graph processing. One common example is the GCN approach which is motivated by CNNs.

The goal of a GCN is to learn features on a graph by propagating the node inforamtion through the graph with convolution steps. The graph $G = (V, E)$ is defined by:

- The features $v_i \in V$ for every node, which can be written in a feature Matrix $X \in \mathbb{R}^{NxD}$ with N = count of nodes and D = size of input feature vector
- The representation of the graph. Most common is a adjacency matrix A, with $A \in \mathbb{R}^{NxN}$ and for every edge between two nodes the entry a_{ij} of node x_i and x_j is increased

The Output of every convolutional step is a node-level Matrix $Z \in \mathbb{R}^{NxF}$ with F = count of output Features. The propagation rule for a multi-layer GCN is defined by:

$$f(H^{(l+1)}, A) = H^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}\tilde{H}^{(l)}\tilde{W}^{(l)}) \tag{2.1}$$

Here, $\tilde{A} = A + I_N$ is the graph representation with the adjacency matrix for undirected graph G including self connections. I_N is called the identity matrix. D is the diagonal degree Matrix and is used to normalize A. With a symmetric normalization and the calculation of $\tilde{D_{ii}} = \sum_j \tilde{A}_{ij}$ we get the Term $\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$. $\tilde{H}^{(l)}$ is the result of the previous Layer with $H^{(0)} = X$ and $W^{(l)}$ is the trainable weight matrix. $\sigma(\cdot)$ denotes an activation function, e.g. $ReLU(\cdot)$ [KW17]

Applying multiple steps of convolution due adding multiple Layers, the convolutional steps although they can be seen as some kind of Message Passing in the graph. In every step the influence of the neighbours is represented in the new node features and therefore the influence of the nodes is passed through the graph.

In a Message-Passing Neural Network (MPNN) this is done with a small reusable neuronal network module for every node. The idea of a reusable function like the NN module is similar to the idea of reusable filters in a CNN Alet et al. [AJB+19]. Formally a MPNN is defined with: $\langle V, E, m_e, m_v \rangle$ with the given set of nodes V and edges E and the two types of NNs. The edge module NN m_e is used to convert the hidden values into a message and pass it to the target node of an edge (Message Passing). The node module NN m_v is calculating a new node value based on the current node value and the incoming massage due the Message Passing [AJB+19].

In a MPNN T steps are applied to calculate final nodes states. The initial state is defined with $h_1^0, \ldots, h_r^0$ and the final state $h_1^T, \ldots, h_r^T$ is achieved by applying:

1. use edge module m_e to compute message

$$m_{ij}^{t+1} = m_e(h_i^t, h_j^t) \tag{2.2}$$

2. aggregate message

$$u_j^{t+1} = \sum_{(i,j)\in E} m_{ij}^{t+1} \tag{2.3}$$

3. update hidden states

$$h_j^{t+1} = m_v(h_j^t, u_j^{t+1}) \tag{2.4}$$

This process is a very adaptive way to apply the idea of passing informations through a graph as its done on grid data with CNNs on a GNN. This process is used in the following approach of Alet et al. [AJB+19]. This approach is used and adapted for the proof of concept presented in this work.

2.4 Graph Element Networks (GEN)

The core idea behind GEN is to model a relation between an input function defined over an input space and an output function defined over the same space. This is done with a computational process defined over a graph using a MPNN. This makes it possible to solve Spatial Function Transformation (SFT) problems.

A SFT is defined by a mapping between a sequence of functions $f = (f_1 \dots f_c) \in \mathbb{F}$ to the sequence of functions $g = (g_1 \dots g_{c'}) \in \mathbb{G}$. Everything is defined over the metric spaces $\mathbb{I}_1, \dots, \mathbb{I}_c, \mathbb{O}_1 \dots, \mathbb{O}_{c'}$ and $\mathbb{X}$. The metric space $\mathbb{X}$ must be bounded. Functions f are defined over a set of data points $(x \in \mathbb{X}, i \in (1 \dots c), s \in \mathbb{I}_i)$ and are mapped from $f_i : \mathbb{X} \to \mathbb{I}_i$. The functions g contain a set of queries with location and dimension points $(x \in \mathbb{X}, j \in (1 \dots c^{'}))$ with the mapping definition $g_j : \mathbb{X} \to \mathbb{O}_j$ and the resulting output values $s \in \mathbb{O}_;$ [AJB+19]

This can now be defined as a problem. There is a set of known samples what in an SFT problem is a a set of data to describe the functions for f. Based on this samples a set of queries (functions g) could now be answered because of the known samples. With regard to the problem statement in chapter 3 the samples could be seen as a set of known nodes in the muscle data and for the other functions all other points should be queried.

To solve this kind of Problems Alet et al. [AJB+19] defined the a structure they call GEN. A GEN extends the SFT definition as follows:

A GEN is a tuple $\langle \mathbb{X}, \mathbb{F}, \mathbb{G}, \mathbb{L}, N, E, e, d, m_e, m_v, T \rangle$

- The spaces of the SFT $\mathbb{X}, \mathbb{F}, \mathbb{G}$
- A latent metrix space $\mathbb{L}$
- N is a finte set of n nodes, where each node v_l is placed at location x_l and v_l is the feature vector representation
- $E \subseteq NxN$ a finite set of edges between nodes
- $e = e_1, \dots, e_c$ is a sequence of encoders, where e_i is a mapping from $\mathbb{I}_i \to \mathbb{L}$ represented as a feed-forward network with weights W_e
- $d = d_1 \dots, d_{c'}$ is a sequence of decorders, where d_j is a mapping from $\mathbb{L} \to \mathbb{O}_j$ represented as feed-forward neuronal network with weights W_d^j
- m_e and m_v are the ede and vertex modules of a GNN defined in the latent space $\mathbb{L}$
- T is the number of rounds of message-passing done by the GNN

In addition to the defined metric spaces of the SFT Alet et al. [AJB+19] introduced a latent space $\mathbb{L}$. This space is used to derive a global solution by learning a simple local model. This is done by using the MPNN approach in the latent space $\mathbb{L}$. First the encoder brings the input data of the metric space $\mathbb{X}$ into the latent space like its shown in figure 2.3. Every node i consists of the coordinate information x_i and the feature vector e_i. The coordinate information x_i is used to calculate the influence of the input node to a node representation in a given graph by calculating the distance to the surrounding graph nodes. This results in a node state for every graph node. Using this local node state the Message Passing is applied by convolving the graph T times with a GCN. Due to this the nodes state is passed through the graph and is represented in the latent state. Like described before this is similar to the Message Passing process in a MPNN. Finally the decode and the resulting graph in latent space is used to decode the query points to the output space $\mathbb{G}$.

The Parameters of the encoders, decoders and the GCN including the node position of the graph itself is learned from data. The data consists of many (f, g) pairs containing the input data samples in f and querys with targets in g. Formally the dataset is described with:

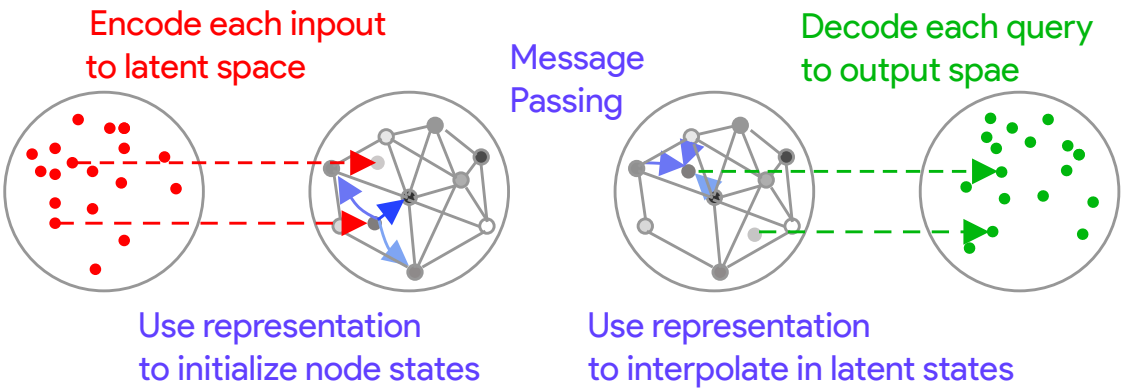


Figure 2.3: The Process of the GEN approach from Alet et al. [AJB+19]. First the node positions x_i and the feature representation v_i is encode into a latent space, then threw Message Passing interpolated in latent the space and finally each query is encoded to the output space.

$$\left\{\left\langle \mathcal{D}_p^{inp}, \mathcal{D}_p^{out} \right\rangle_{p=1}^{P}\right\} \tag{2.5}$$

$$\mathcal{D}_p^{inp} \in \prod_{l_1=1}^{L_1} \left(\mathbb{X} \times (1 \ldots c) \times \mathbb{I}_{i_{l_1}}\right) \tag{2.6}$$

$$\mathcal{D}_p^{out} \in \prod_{l_2=1}^{L_2} \left(\mathbb{X} \times (1 \ldots c^{'}) \times \mathbb{O}_{j_{l_2}}\right) \tag{2.7}$$

Based on this the data for the concept presented in this work is constructed like described in chapter 5.2. In the following Chapter related work in the field of Bioinformatic and the usage of GNN is presented.

3 Problem Statement

The complexity of FEM calculations is a common challenge to solve in simulation. The numerical calculation of the underlying PDE is a very time and resource consuming task. There are many approaches trying to reduce the need of resources for these simulations. Although real time simulations is of great interest for computer-aided surgery for example [MMC20].

Due to the fact that trainend Deep Learning models in most cases are very resource efficient in inference, it is of great value to use Deep Learning approaches in simulation. Among others in the simulation of continuum-mechanical skeletal muscle models.

In this work the Deep Learning architecture of the work of Alet et al. [AJB+19] is used to apply a GNN to muscle data with the goal to present a proof of concept that learns the muscle data and finally simulate the deformation. The work core challenges are creating a meaningful dataset of muscle data to generate a graph representation and to adapt the approach of Alet et al. [AJB+19] to the created dataset.

The data generated by a FEM based simulation is used with different metrics and is represented in a structure in order to achieve a general representation of every muscle state for every move.

To adapt the approach of Alet et al. [AJB+19] the challenge is to find a 3-dimensional mesh structure that represents the muscle system and fits to the calculation due the message passing process to find the global solution out of local informations in the data. The node values of the simulation needs to be presented in the Mesh of GNN. As a result of the new dimension the representation function r mapping the simulation nodes to the Mesh nodes needs to be defined. Although the convolutional steps needs to be adapted to get the approach work on the new structure.

4 Related Work

This work focuses on the use of GNNs to simulate muscle deformation. As far as known this work is one of the first approaches to apply GNNs on muscle data. Nevertheless this work can be categorized in the field of simulation optimization for muscle deformation and the use of Machine Learning in the field of bioinformatics in general. There are different kind of approaches with valuable results to improve FEM based simulation and apply Machine Learning to problems in this research area.

As an example in the work of Mendizabal et al. [MMC20] they present an approach to simulate non-linear material very fast and accurate by using an artificial neuronal network to represent the relation in the data in a smaller dimension. The used network in this work is based on the u-net architecture of Ronneberger et al. [RFB15]. The input of the network is a tensor with the amount of nodes of a Mesh used for FEM solvers, containing the forces applied to the mesh. As output of the u-net they get the corresponding displacement of every node. They trained this network with random forces and calculated ground truth out of a FEM simulation [MMC20].

They prove their results on different kind of shapes like cantilever beam or L-shaped object and getting good results based on the FEM generated data they use for training and validation. They also benchmark the approach to state-of-the-art techniques like proper orthogonal decomposition (POD). The idea behind POD is to obtain lower dimensional approximation of higher dimensional processes [Cha00]. The core constrain in this work is, that the network only works for a fixed defined geometry. The proof of concept presented in this work is able to work on random graph structures.

In the Work of Fout et al. [FBSB17] they use approaches in the field of graph convolutions and adapt them to the 3-dimensional grid crystal structure of proteins. The functionality of Proteins is based on a very complex network with interaction between all these proteins. In the work of Fout et al. [FBSB17] they try to predict the protein interfaces through which the proteins communicate.

They adapted different kind of convolutions with a variation of convolution steps, for example a diffusion or a simple node average and compare the results. The experiments resulted in a state-of-the-art performance and proved the effectiveness of convolutions on protein graph structures. Since they rely on the use of convolutions on graphs they can also learn on diverse graph structures and are not bound to one geometry. Therefore the experiments could also be adapted to other problems

In the Work of Duvenaud et al. [DMA+15] they use a Neuronal Network to learn molecular fingerprints by replacing the common way of using static fixed-size feature vectors with dynamic graph structures. In the graph every node represents an individual atom and the edges represents the bonds. On this structure they apply multiple steps of convolutions on the graph itself and finally combine every information in the nodes states with a pooling function, very similar to the CNN or GCN calculations explained in chapter 2

All of this works using state of the art Machine Learning approaches on complex data structure and learn a global solution for the specific problem. In the work of Alet et al. [AJB+19] they presented a solution to learn a global solution out of local data points. This is possible cause of the use of a graph processing inside an introduced latent space explained in chapter 2. They also described there solution as a possibility to solve different kind of PDEs similar to numerical solutions like FEM. To proof the approach Alet et al. [AJB+19] demonstrate the efficiency of the Model by solving several PDE problems. For example they model the propagation of the heat in two-dimensional "houses". The houses vary in the heater and cooler position and the exterior temperature. The goal is to predict the heat on every coordinate in the house for every possible positioning and architecture (global solution) out of a set of possible architecture examples (local solution).

Like explained in chapter 3 the calculation of the deformation for muscle data is very complex and time consuming. The deformation is a complex PDE and often solved with FEM solver. Since the promising results in the work of Alet et al. [AJB+19] for solving PDE problems, this work will be adapted to the problem of chapter 3. The defined data structure used to apply the adapted approach of Alet et al. [AJB+19] and the necessary adjustments are described in the following chapter.

5 Concept

The deformation of muscle systems is mostly evaluated by using FEM based simulation which is approximating the underlying PDE. Due to the promising result of the approach of Alet et al. [AJB+19] solving general PDEs in 2-dimensional spaces, this work adapts the introduced GEN and applies the architecture on 3-dimensional muscle data. Therefore the data structure and the adaption of the GEN approach is described.

5.1 Data generation

The data used in this work is based on the „Visible Human Project “of Ackerman [Ack21]. The idea behind this project is to map all the textual printed informations about functional-physioligical knowledge with some image data of structural-anatomical knowledge [Ack21]. In the last years they published a set of 3-dimensional human body images.

With this dataset the work of Valentin et al. [VSPR18] is applied to simulate forward-dynamics of the upper limb. They presented a activation-driven musculoskaletal system model that uses 3-dimensional, continuum-mechanical skelatal muscles models to calculate skelatal muscle forces in which the muscle activations are determined based on a constraint optimization problem. Due the computaion of spars grid surrogates with hierarchical B-splines, they achieved a drastic reduction of computation time for the optimization processes [VSPR18].

By using the work of Valentin et al. [VSPR18] and some support points based on a FEM calculation on top of the „Visibal Human Project “the data for the training and the evaluation is generated.

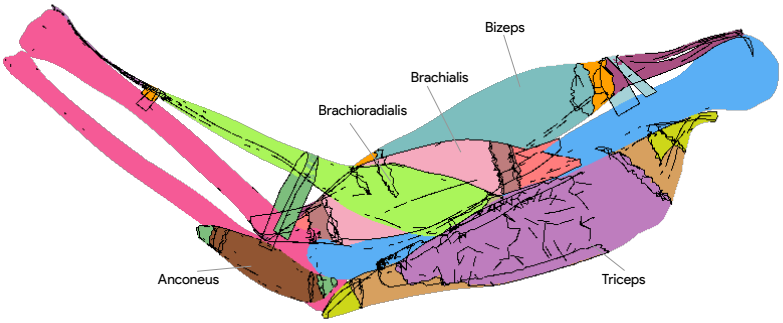
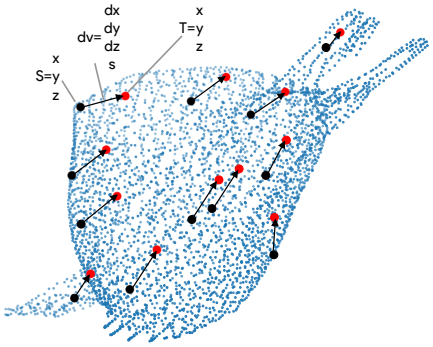


Figure 5.1: Muscle structure of the upper arm with the 5 muscles brachioradialis, brachialis, bizeps, anconeus and triceps



(a) Biceps with system activation zero

(b) Biceps with system activation one

Figure 5.2: The Node representation of the biceps with different activations in the complete arm system. Figure5.2a shows the biceps with a 0% activation of all 5 muscles and figure 5.2b with 100%. The black nodes are example coordinates of the points used for the input dataset $\mathcal{D}_p^{inp}$. The blue Coordinates are examples for the query dataset $\mathcal{D}_p^{out}$. Figure 5.2a although illustrates the initial coordinates S, target coordinates T and the direction vector d_v used as feature

For simplification the work is focused on the biceps muscle and the upper limb arm system with the muscles anconeus, biceps, brachialis, brachioradialis and triceps as shown in the figure 5.2.

5.2 Data structure

The system is a activation-driven musculoskeletal system model [VSPR18]. The complete state of the system will be described with an activation vector $a \in \mathbb{R}^5 : 0 \geq a_{[i]} \leq 1$. Every entry in the 5 dimensional vector describes the activation of a specific muscle between 0 and 1. In case of a 100% contraction for every of the 5 muscles the activation vector is $a = (1, 1, 1, 1, 1)$ and for 0% every vector entry is represented by 0. For example the biceps muscle in figure 5.2b is $a = (1, 1, 1, 1, 1)$ because every muscle in the system is contracted with 100% in contrast to the muscle shown in figure 5.2a which is given by a activation vector $a = (0, 0, 0, 0, 0)$

The goal in this work is to apply the GEN approach of Alet et al. [AJB+19] to learn this muscle system and evaluating every deformation of the muscles with respect to the activation. Therefore a dataset with input samples $\mathcal{D}_p^{inp}$ and query samples $\mathcal{D}_p^{out}$ similar to the structure in formula 2.5 of chapter 2 is generated.

With the simulation a set of activations and the corresponding node values are calculated. Every data pair p_i in the dataset $\left\langle \mathcal{D}_p^{inp}, \mathcal{D}_p^{out} \right\rangle_{p=1}^{P}$ consists of a set of nodes correlated to an initial and a target activation. For example the dataset $\left\langle \mathcal{D}_1^{inp}, \mathcal{D}_1^{out} \right\rangle$ could be calculated due the deformation from initial activation $\vec{a}_{init} = (0.0, 0.0, 0.0, 0.0, 0.0)$ to target activation $\vec{a}_{target} = (1.0, 1.0, 1.0, 1.0, 1.0)$ with the direction vector of the changed node position. The node representation is a combination of the position of the node at initial activation $\vec{a}_{init}$ with (S_x, S_y, S_z) and the 4 dimensional feature vector (d_x, d_y, d_z, s) with the direction d_x, d_y, d_z and s the scale factor of the direction vector. The relevant coordinates and calculated direction value is illustrated in figure 5.2a. For every system configuration described by a given activation and the deformation affected by the

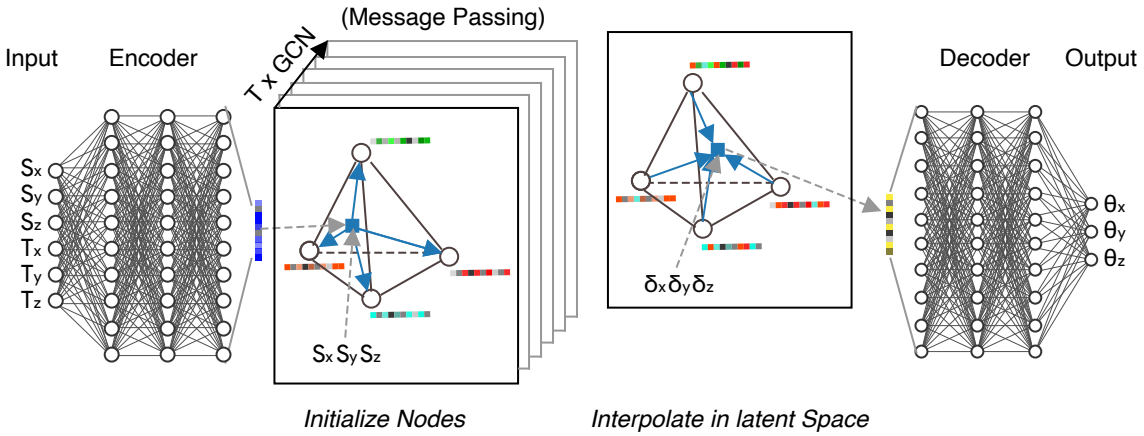


Figure 5.3: The adapter GEN architecture. Similar to the illustration of Alet et al. [AJB+19] in figure 2.3. The figure shows the fully connected networks of the encoder/decoder and the Latent space $\mathbb{L}$ with the Message Passing due the convolutions and the representer function r which is mapping the nodes of the simulation to the nodes in the GNN graph structure.

activation change, every $\mathcal{D}_p^{inp}$ consist of a set of Nodes with the combination of the described entries coordinates+feature vectors $(S_x, S_y, S_z, d_x, d_y, d_z, s)$. The $\mathcal{D}_p^{out}$ is defined with a similar structured vector, but contains a different set of nodes $(\delta_x, \delta_y, \delta_z, d_x, d_y, d_z, s)$. The result is a a set of nodes with coordinates and features in the input set $\mathcal{D}_p^{inp}$ containing a local description of the current activation and a complete different set of nodes in the query set $\mathcal{D}_p^{out}$ with query coordinates and the expected result.

This structure ensures, that the model learns the current activation state out of data with a minimal set of nodes. As an example the model is trained in the input dataset with the black nodes in figure 5.2a and is finally able to evaluate any other node for example the blue nodes in figure 5.2a.

The features used in the node representation can vary. Possible features could be the direction vector as just described, the absolute node position of the target activation, the force acting on a particular node at target activation, or anything similar. Although the features in the input and output do not have to match. It is possible to use the direction vector to learn the local state in the input data and querying the forces at the the output. In Chapter 6 different kind of features are used to prove the concept. For simplification in this chapter the target coordinate is used as feature vector.

5.3 Mesh and Graph processing

In contrast to the approach of Alet et al. [AJB+19] the coordinates of the input and query data is in 3-dimensional space, like explained in the chapter 5.2. Therefore the mesh and used space needs to be changed.

To increase the dimension of the underlying graph for the GNN processing a random mesh with delaunay triangulation is used.

The Delaunay Triangulation solves the Problem of triangulating a random set of points with the property that the triangle (2D) or the tetrahedron (3D) contains no point of the Node set in its interior [LS80]. In this work the scipy implementation [VGO+20] with the Qhul algorithm of Barber et al. [BDH13] is used. This Implementation algorithm automatically generates a set of edges with the described properties.

With the given data structure and the graph in 3-dimensional space, the GEN can be trained and the data can be processed through the architecture shown in figure 5.3. Everything starts with the encoder to get the input data in space $\mathbb{X}$ to the latent space $\mathbb{L}$. This is done with the encoder implemented as a fully-connected Feed Forward Neural Network, similar to the implementation of Alet et al. [AJB+19] shown in Listing 5.1. In line 10 some linear layers are added based on the defined amount of layers. In line 13 to 18 the data is propagated through the network. Finally the Rectified Linear Activation Function (ReLU) is applied at the last layer in line 17. The ReLU is an activation function mapping every negativ value to zero and only outputs the positiv values.

```
1  import torch
2  from torch import nn
3  import torch.nn.functional as F
4
5  class Net(nn.Module):
6      def __init__(self, dims):
7          super(Net, self).__init__()
8          self.layers = nn.ModuleList()
9          self.dims = dims
10         for i in range(len(self.dims)-1):
11             self.layers.append(nn.Linear(self.dims[i], self.dims[i+1]))
12
13     def forward(self, x):
14         for i in range(len(self.layers)):
15             x = self.layers[i](x)
16             if i+1 < len(self.layers):
17                 x = F.relu(x)
18         return x
```

Listing 5.1: Encoder/Decoder implementation

After propagating the data from input space $\mathbb{X}$ to latent space $\mathbb{L}$ the nodes of the input data with the feature vector in latent space $\mathbb{L}$ needs to be mapped from input position (S_x, S_y, S_z) to a node represented in the defined GNN mesh. This is done with the representer function $r : \mathbb{X} \rightarrow \mathbb{R}^3$. The representer function is calculating the influence of every input data node to every node in the GNN mesh with respect to node distance. In figure 5.3 this process is illustrated in the „Initialize Nodes “step.

The implementation of the representer function is shown in listing 5.2. The pseudo squared error distance is calculating the node distances in line 2 and the results are mapped in a range between 0 and 1 with the softmax function. The resulting scores represents the influence of every input node on every GNN mesh node.

```
1 def repr_fn(mesh_nodes, input_nodes):
2     pseudo_dist = (torch.norm(input_nodes, dim=1)**2 - 2*torch.mm(mesh_nodes,
  input_nodes.t()))
3     scores = F.softmax(pseudo_dist, dim=1)
4     return scores
```

Listing 5.2: The representer function implementation

To propagate the feature information of the input node through the graph a GCN is applied for T steps as illustrated in figure 5.3. The GCN implementation used in this work is the implementation of Fey and Lenssen [FL19] based on the Architecture of Kipf and Welling [KW17].

At this point the GNN mesh contains the new node states in form of updated feature vectors. The model is now queried with the query points of the output dataset. This is done by applying the representer function again on the query points, but this time the aggregated features are appended to the coordinates of the query points. This means the representer function calculates, based on the distance between query node and mesh node, the influence of every GNN mesh node to the query node $(\delta_x, \delta_y, \delta_z)$ and creates a node representation in the latent space $\mathbb{L}$. Finally the decoder is used to project the node in latent space $\mathbb{L}$ to the output space $\mathbb{O}$.

The pseudocode of the adapted GEN implementation is shown in listing 5.3. For simplification the pseudocode does not show the detailed implementation of training techniques like batching. In line 2 the input is encoded into latent space $\mathbb{L}$. The influence for the GNN mesh nodes as shown in listing 5.2 is done in line 3 and is applied on the graph with a batched matrix multiplication in line 4. Line 6 to 9 is applying the convolution steps with the GCN to a graph copy. The result of the Message Passing is applied to the node features of the graph in line 10. The last lines 12 to 14 is calculating the final result by calculating the influence of the GNN mesh nodes to the query nodes and decodes the nodes from latent space $\mathbb{L}$ to the output space $\mathbb{O}$

```
1  def GEN_3D (encoder, decoder, gcn, input, query, graph, mesg_steps)
2      latent_input = encoder(input)
3      scores = repr_fn(graph.pos, input)
4      graph.x = torch.bmm(torch.transpose(scores, 1, 2), latent_input)
5
6      tmp_graph = graph
7      for step in range(msg_steps):
8          tmp_graph.x = self.layer_norm(tmp_graph.x + gcn(
9              torch.cat((tmp_graph.pos, tmp_graph.x), dim=-1), tmp_graph.edge_index))
10     G.x = tmp_graph.x
11
12     scores = repr_fn(G.pos, query)
13     latent_output = torch.bmm(scores, G.x)
14     output = dec(latent_output)
15
16     return output
```

Listing 5.3: Pseudocode of the adapted GEN of Alet et al. [AJB+19]

To train the model, the parameters of every component needs to be optimized. A common way in Machine Learning is to use a Gradient Descent algorithm. The Gradient Descent is optimizing the model by updating every parameter in the opposite direction of the gradient. In this work the Adaptive Moment Estimation (Adam) algorithm [KB15] is used for optimizing all trainable components namely the GCN, the NN of the encoder/decoder, and the node positions of the Mesh. The Adam is a Gradient Descent algorithm with the core idea of using the gradients calculated in previous steps for the optimization in the current step [Rud16].

In summary this concept contains the definition of the dataset, the adaption of the GEN approach of Alet et al. [AJB+19] and the description to optimize the model. The result is a approach to apply a GNN on muscle data. In this work the concept is also be proven in chapter 6

6 Experiments

This chapter aims to prove the concept described in chapter 5 and shows the effectiveness on the simulation data described in Section 5.1. Furthermore the concept will be compared with another approach to learn the deformation of a muscle system. Therefore the dataset used for training and validation is described, the benchmarking model architecture is shown, and the results of the experiments are discussed.

6.1 Description

Dataset In the experiments a dataset structured like the setup in Equation (2.5) is used. 32 random activations are evaluated with different activation steps. An activation step is the granularity of the used values in the activation vector e.g 0.1 with a random set of activations $\{(0.1, .2, .0.2, 1.0.0.8), (0.4, .3, .0.1, 1.0.1.0), \dots\}$. The dataset contains 4027 nodes for each of the 32 activations and is splitted into a train- and testset (80/20) over the different initial activations. For the D^{inp} and D^{out} a diffent set of nodes is used. For example 9 random nodes in D^{inp} and 4018 in D^{out}.

Benchmark Model To compare the result on the given dataset a Feed Forward Neural Network illustrated in figure 6.1 is used to benchmark the concept presented in this work. The benchmark model contains 5 hidden layers with an exponential growth of the node count for every layer. This creates an architecture with an output of 2809 nodes out of a 5-dimensional activation input. The model is trained with data of the described simulation in chapter 5.1. As input it gets an activation $a \in \mathbb{R}^5$ and the output is a list of nodes with the given coordinates for the requested activation. The model is trained on a fixed set of 2809 nodes. Therefore the output is limited to this set. This nodes are all on the surface of the muscle and are a subset of the 4027 nodes. The activation trained into

Table 6.1: The different metrics used in the experiments

Metric	Description
Coordinates	The Coordinates of the node position
Direction Vector	The direction vector describing the movement of a node caused by the deformation
Maximum Distortion Criterion	With the the simulation described in chapter 5.1 a symmetrical stress tensor for every node is calculated. To compare the results the Maximum Distortion Criretior calculates a scalar to represent the forces. It is although called the von Mises yield criterion [Mis13].

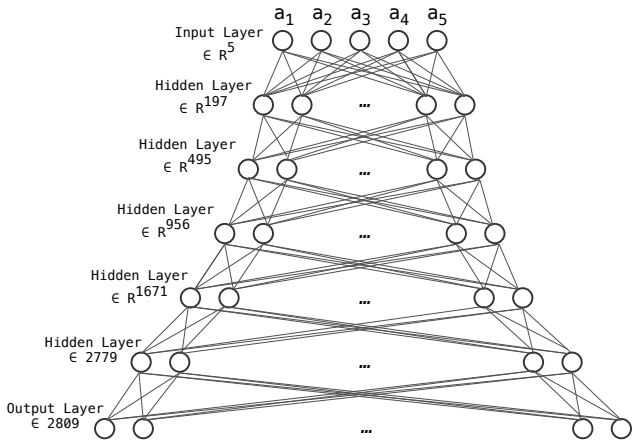


Figure 6.1: An illustration of the Layer architecture of the Feed Forward Neuronal Network used to benchmark the concept presented in this work. Five Hidden Layers with an exponential count of nodes for every Layer.

the benchmark model is a subset of 1743 support activations described in chapter 5.1. To get a meaningful comparison only the fixed set of nodes without the trained support activations are used for the comparison with the benchmark model.

Metric and Variations In Table 6.1 the different metrics used for training and evaluation are described. Although different mesh sizes, different amount of nodes for D^{inp} and D^{out} and different Layers of the encoder and decoder is evaluated to get a nice understanding of the effectiveness of the concept.

6.2 Execution

To run the experiment the code of Alet et al. [AJB+19] is extended. For training two NVIDIA RTX 3090 with an AMD Ryzen 9 and 128 gb RAM is used. To visualize the results mostly the matplotlib [Hun07] library is used. For Monitoring the pytorch libary [PGM+19] is extended with the Tensorboard dashboard [MAP+15].

6.2.1 Learning Target Coordinates

In this experiment a set of 32 activations with 9 nodes for the D^{inp} and 4018 nodes in the D^{out} dataset is used. The activations are evaluated with a activation step of 1.0. Therefor it is a set of $\{(0.0, 0.0, 0.0, 0.0, 0.0), (1.0, 0.0, 0.0, 0.0, 0.0), (0.0, 1.0, 0.0, 0.0, 0.0), \dots\}$ activation vectors. As metric the target coordinates are used like described in table 6.1. Therefor every dataset contains the corresponding nodes with the vector $(S_x, S_y, S_z, T_x.T_y, T_z)$.

The encoder is initialized with an input layer of dimension 6, 2 hidden Layers of dimension 32 and an output Layer with output dimension 32. The decoder is getting the features vectors of dimension 32 with the coordinates of dimension 5 as input. That is why the decode is initialized with an input layer of dimension 35, two hidden Layer of dimension 32 and an output layer of dimension 3 with the resulting target coordinates in the ouput.

The model with the GCN and the encoder/decoder is optimized with the Adam optimizer and a learning rate of $1e^{-4}$. Two different Graphs are initialized with a node count of $2^3 = 8$ and $2^7 = 343$. The Graph optimizer is also the Adam optimizer with a learning rate of $1e^{-1}$. The loss is calculated with the MSE. For benchmarking the max error is calculated too.

Result Discussion

After 990 Epochs the optimal error in mm is 5.72 on the complete query test dataset of 4018 nodes. To compare the results with the benchmark model the reduced node set of 2809 nodes are evaluted and achieves an error of 7.0 in mm. The benchmark model achieves an accuracy of 2.8 mm. The GEN also scores slightly worse in maximum error. The maxerror for the adapter GEN is 25.6mm and for the benchmark model 10.79 mm.

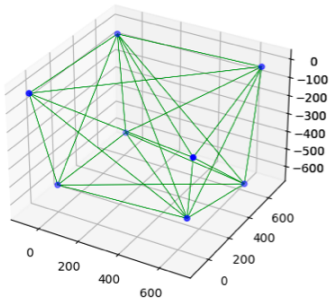
The result means, that the model can evaluate every queried node in the muscle with a given set of 9 nodes to describe the muscle activation, with a precision of 5.72 mm. In contrast to the benchmarking model, this model is able to evaluate a fixed set of nodes with a precision of 2.8mm.

Figure 6.2 shows some illustrations of the results. The adapted GEN is initialised with the mesh in figure 6.2a and 6.2b. After 990 the mesh in figure 6.2e distributed the nodes over the entire range of all possible muscle activations and corresponding node positions. The smaller mesh 6.2d distributed itself the same way. Both meshes results in the same granularity of 5.8 mm for the 4018 nodes. This means, that there is no need for a huge mesh for this kind of problem and dataset.

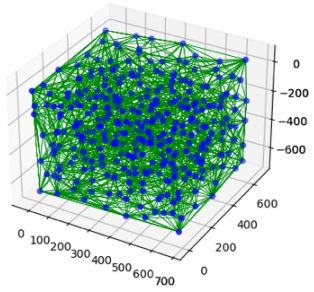
In general the mesh needs is to train a global solution out of local informations, in that case the different initial activations. This makes it possible to query every node for every activation. The expected behavior of the mesh is, that the nodes of the mesh accumulate at interesting places, for example areas on the muscles with a huge change of the node position. This can be seen in figure 6.2e. The nodes accumulate at this position in the coordinate space with the largest mass of the muscle. Therefore, the highest change can be perceived by the muscle movement at this area. But also the direction of the movement equals very much in this area. For that reason there is no need for a huge mesh.

In figure 6.2g the granularity of the benchmark model is shown with red nodes for high errors and white nodes for small errors. The heat map shows, that the nodes with high Errors are distributed over the complete muscle. In contrast figure 6.2f shows, the result of the adapted GEN model. In general the max errors are higher then the benchmark model, but the errors focus on the tendon insertion on the biceps.

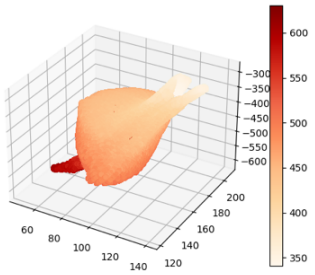
This experiment shows, that the concept is able to learn the deformation out of the simulation data similar to the benchmark model with some lost of granularirty but allows to evaluate any coordinate in the muscle and is not bound to a fixed set of nodes.



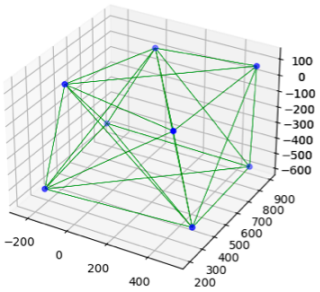
(a) initial Mesh with 2^3 nodes



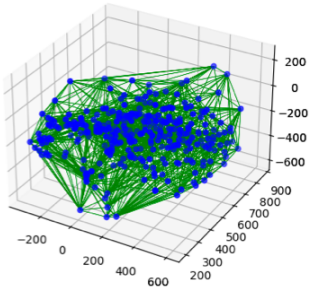
(b) initial Mesh with 7^3 nodes



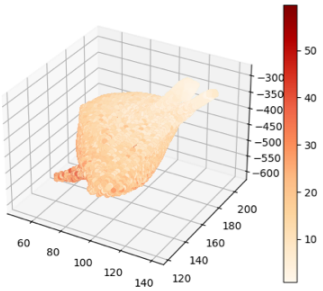
(c) heatmap at epoch 1



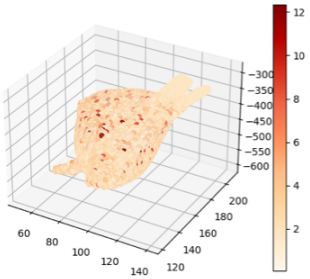
(d) resulting Mesh with 2^3 nodes



(e) resulting Mesh with 7^3 nodes

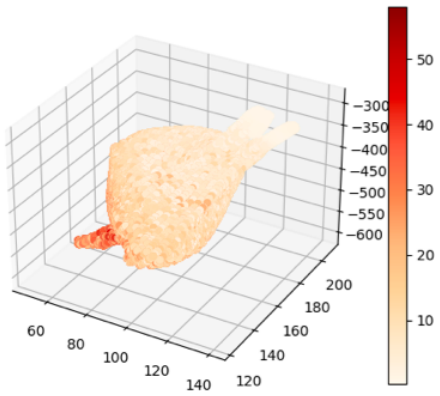


(f) heatmap at epoch 990

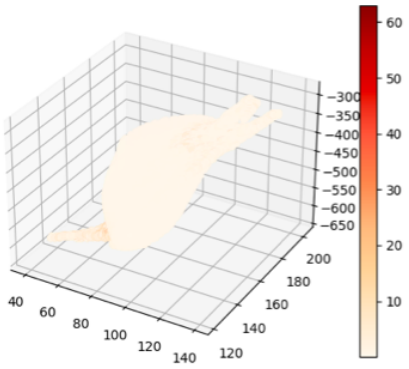


(g) heatmap of benchmark model

Figure 6.2: Graphical evaluation of the first experiment described in chapter 6.2.1. It shows the initial meshes with 7 nodes in figure 6.2a and 343 nodes in figure 6.2b and the corresponding results in figure 6.2d and 6.2e. Figure 6.2c and 6.2f is showing the learned accuracy in form of a heatmap by marking the highest error nodes in red and lower errors in white. The area of the legend and the axes adjust due to the high value changes. The last figure 6.2g shows the heatmap of the benchmarking model.



(a) The heatmap result by using direction vector as metric



(b) The heatmap result by using the Maximum Distortion Criterion

Figure 6.3: Different heatmap results for the metric variations

To show that the concept can also be used on on different kind of metrics, the following experiments are applied

6.2.2 Learning Direction Vector

The previous experiments have shown, that the model is able to learn the deformation with the target vectors as metric. The target vectors in the output can be seen as an absolute state for every activation. Hence, the evaluation of a activation is not effected by the initial activation. In this experiment the direction vector is used. Based on the initial activation every target activation results in a different direction vector. Therefore the model needs to learn, that every initial activation state combined with a target activation state results in a different system move.

Result Discussion

The results are very similar to the experiment in the chapter 6.2.1. The heatmap of the result is shown in figure 6.3a. Again, the least accuracy focusing at the tendon insertion on the biceps. The model achieves a granularity of 5.5mm. That means for a given start activation due 9 reference nodes the model can predict the direction vector for every node in the muscle with a granularity of 5.5mm.

The model can not be compared to the benchmark model, because the benchmark model is trained for predicting target node position. But the 5.5mm average accuracy is very similar to the result on the target vector.

6.2.3 Learning maximum distortion criterion

In the previous examples the deformations with the node position changes has been evaluated. An interesting value is also the force applied to certain points in the muscle. This forces can be represented in the form of the Cauchy stress tensor. This $3x3$ tensor consists of 9 entries which describe the stress at a point inside the muscle [Irg08]. The simulation used in this work can also be used to evaluate this Tensor. To simplify the comparison the Maximum Distortion Criterion is used like described in Table 6.1. This Criterion Projects the $3x3$ tensor to a single comparison value.

For this the von mises equation 6.1 [Mis13] is implemented. The $3x3$ Tensor is a symmetrical matrix, therefor only the diagnoal values $\sigma_x, \sigma_y, \sigma_y$ and the 3 symmetrical values $\sigma_{xy}, \sigma_{yz}, \sigma_{zx}$ ares used to calculate the comparison value.

$$\sigma^2 = \frac{1}{2}((\sigma_x - \sigma_y)^2 + (\sigma_x - \sigma_z)^2 + (\sigma_z - \sigma_x)^2) + 3(\sigma_{xy}^2 + \sigma_{yz}^2 + \sigma_{zx}^2) \tag{6.1}$$

Result Discussion

The achieved granularity on this metric results on an average value of 2.2 MPa. The distribution of the granularity is shown in figure 6.3b. Like seen before the most difficult part to predict is the tendon insertion. In a previous work of Kneifel et al. [KROJ] they used Gaussian Process Regression to evaluate the forces on a specific point in the muscle. They achieved an average value of 0.2 MPa and Maximum Error of 11.5 MPa.

6.2.4 Reducing Complexity

All the previous experiments are done with the same model konfiguration. The experiments have shown that the results could also be achieved by a model with less learnable parameters. Therefor in this experiment the size of the encoder/decoder is reduced and the calculations in the latent space $\mathbb{L}$ are done in smaller spaces. The results are compared and discussed.

Result Discussion

In figure 6.4 the training progress with the train curves is plotted for different encoder/decoder configuration. The legend describes every Layer of the fully connected network with the input and output. For example the green curve shows the Encoder with Input (input:6 - output:32) and Output Layer (input:32-output:32) and two additional hidden layer (input:32-output:32). For every epoch the MSE is shown. The GNN is initialized with a Mesh of 343 nodes.

The green line is the default configuration of Experiment 6.2. A encoder/decoder dimension from 32 to 16 with a reduction of the hidden layers doesn't affect the results very much (orange curve in figure 6.4). It only changed the progress of optimization with a smaller deflections in the loss values. This effect often occurs comparing two training curves with a different amount of trainable parameters. The underlying concept of the Adam optimizer is the stochastic gradient descent [KB15]. The idea of these concepts is to find a local minimum of a given function. Combined with

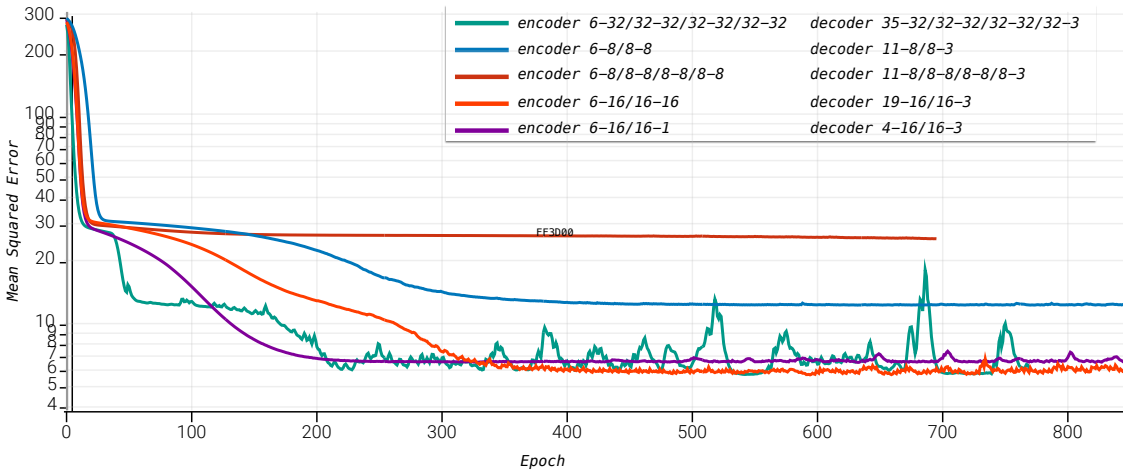


Figure 6.4: Different train curves with the epoch and the corresponding MSE. For every curve a different encoder/decoder konfiguration is used. In the Legend for every curve the layers with the corresponding input-output dimension is described

the adjustable learning rate, one of the hyperparameter of this model and the amount of trainable parameters the defflection of the learning curve can vary more or less. This is only one possible explanation. A train curve interpretation is complex. The work of Viering and Loog [VL21] gives some nice insights of interpreting learning curves.

Reducing the dimension of the hidden layers to a dimension of 8 results in higher loss values like shown in the red and blue curve. This leads to a necessary dimension of 8 in the encoder and decoder hidden layers.

Even for smaller encoders and decoders the mesh size doesn't matter a lot. Cause the results are always similar.

Finally the purple curve shows the training with a reduced Latent Space $\mathbb{L}$ to 1. This means, the feature v_i for every node on the Mesh is reduces to a single value. Therefore every node in the Mesh consists of its position x_i and a single value v_i.

In the curve can be seen a loss of accuracy compared to the green and orange curve with higher Latent Spaces. The encoder and decoder in the training with the reduced latent Space to 1 using layers with dimension 16. Therefore the curves in purple and orange equal in there defflection.

As a result of the different hyperparameter konfigurations it can be seen, that the encoder/decoder dimension as well as the latent space can be reduced to solve the problem on the given dataset, but there is a minimum of a latent space L with dimension 8 and a Layer dimension of the encoder/decoder of 16. Otherwise the results would be worse.

7 Conclusion and future work

The use of a Graph Neural Network (GNN) to simulate the deformation of a muscle in case of the upper limb movements was examined in this work. Therefore, a proof of concept was designed and applied to a set of muscle data due a FEM based simulation with different metrics.

The created concept is based on the work of Alet et al. [AJB+19] from wich the core architecture is used as well as the code which has been adapted to run the experiments.

As a result the data from the simulation has been represented in an input D^{inp} and output D^{out} dataset with the activation representation for the 5 muscles brachioradialis, brachialis, bizeps, anconeus and triceps in the input and the query nodes in the output dataset. As well as the GEN architecture of Alet et al. [AJB+19] which was adapted to a 3-dimensional GNN.

The experiments in chapter 6 has shown, that the concept can learn the deformation process of the biceps. The results has been compared with similar approaches to predict the node states. As an example a Machine Learning architecture which is able to predict the node position on a given acitvation vector on a fixed set of nodes was used for comparison. In all the experiments the concept could learn the deformation with a small loss of accuracy compared to the benchmark models, but with the advantage that the adapted GEN concept is not bounded to a fixed set of nodes or geometry.

In the last experiment the complexity of the architecture on the presented concept has been reduced in different ways by configuration of the hyper parameters, for example the amount of hidden layers or the dimension of the introduced latent space L. The result has shown that with a small set of trainable parameters the model can still learn the deformation.

This insights suggest, that there is a great use of GNN in the field of muscle movement simulation. To further develop this concept it could also be trained with data containing a significantly higher variance in the data, because the simulation data in this work has been very smooth. The results show that the model can possibly also be trained with the original data reduce the need of simulation values for the training.

To evaluate the generalisation of the model in the concept further experiments are planned. The model will be trained with a different set of activations and direct FEM data and the results of the models will be compared. Unfortunately, the data for this experiment were not yet available at the end of this work.

As another improvement of the concept different kind of convolutions could be applied to make the Message Passing more effective. Finally this could result in a very thin model which is able to run on small edge devices for the inference as example for the use on arm prosthesis

Bibliography

[Ack21] M. J. Ackerman. “The Visible Human Project”. In: *Studies in Health Technology and Informatics* 288.3 (2021), pp. 134–140. ISSN: 18798365. DOI: 10.3233/SHTI210988 (cit. on p. 27).

[AG09] N. Antoanela, D. Gherghel. “The application of the finite element method in the biomechanics of the human upper limb and of some prosthetic components”. In: *WSEAS Transactions on Computers* 8.8 (2009), pp. 1296–1305. ISSN: 11092750 (cit. on p. 15).

[AJB+19] F. Alet, A. K. Jeewajee, M. Bauza, A. Rodriguez, T. Lozano-Pérez, L. P. Kaelbling. “Graph element networks: Adaptive, structured computation and memory”. In: *36th International Conference on Machine Learning, ICML 2019* 2019-June (2019), pp. 321–331. arXiv: 1904.09019 (cit. on pp. 15, 18–23, 26–32, 34, 41).

[Ayo10] T. O. Ayodele. “Types of machine learning algorithms”. In: *New advances in machine learning* 3 (2010), pp. 19–48 (cit. on p. 15).

[BDH13] C. B. Barber, D. P. Dobkin, H. Huhdanpaa. *Qhull: Quickhull algorithm for computing the convex hull*. Astrophysics Source Code Library, record ascl:1304.016. Apr. 2013. ascl: 1304.016 (cit. on p. 30).

[Cha00] A. Chatterjee. “An introduction to the proper orthogonal decomposition”. In: *Current Science* 78.7 (2000), pp. 808–817. ISSN: 00113891. URL: http://www.jstor.org/stable/24103957 (visited on 07/19/2022) (cit. on p. 25).

[DMA+15] D. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, R. Gómez-Bombarelli, T. Hirzel, A. Aspuru-Guzik, R. P. Adams. “Convolutional networks on graphs for learning molecular fingerprints”. In: *Advances in Neural Information Processing Systems* 2015-January (2015), pp. 2224–2232. ISSN: 10495258. arXiv: 1509.09292 (cit. on p. 25).

[FBSB17] A. Fout, J. Byrd, B. Shariat, A. Ben-Hur. “Protein interface prediction using graph convolutional networks”. In: *Advances in Neural Information Processing Systems* 2017-December.Nips (2017), pp. 6531–6540. ISSN: 10495258 (cit. on p. 25).

[FL19] M. Fey, J. E. Lenssen. “Fast Graph Representation Learning with PyTorch Geometric”. In: 1 (2019), pp. 1–9. arXiv: 1903.02428. URL: http://arxiv.org/abs/1903.02428 (cit. on p. 31).

[Hun07] J. D. Hunter. “Matplotlib: A 2D graphics environment”. In: *Computing in science & engineering* 9.3 (2007), pp. 90–95 (cit. on p. 34).

[Irg08] F. Irgens. *Continuum mechanics*. Springer Science & Business Media, 2008 (cit. on p. 38).

[KB15] D. P. Kingma, J. L. Ba. "Adam: A method for stochastic optimization". In: *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings* (2015), pp. 1–15. arXiv: 1412.6980 (cit. on pp. 32, 38).

[KROJ] J. Kneifel, D. Rosin, R. Oliver, F. Jörg. "Low-dimensional Data-based Surrogate Model of a Continuum-mechanical Musculoskeletal System Based on Non-intrusive Model Order Reduction". In: *NOT PUBLISHED YET* () (cit. on p. 38).

[KW17] T. N. Kipf, M. Welling. "Semi-supervised classification with graph convolutional networks". In: *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings* (2017), pp. 1–14. arXiv: 1609.02907 (cit. on pp. 20, 31).

[KW91] K. Knothe, H. Wessels. *Finite elemente*. Vol. 2. Springer, 1991 (cit. on p. 17).

[LS80] D. T. Lee, B. J. Schachter. "Two algorithms for constructing a Delaunay triangulation". In: *International Journal of Computer Information Sciences* 9.3 (1980), pp. 219–242. ISSN: 00917036. DOI: 10.1007/BF00977785 (cit. on p. 30).

[MAP+15] Martın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, Xiaoqiang Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/ (cit. on p. 34).

[Mis13] R. v. Mises. "Mechanik der festen Körper im plastisch- deformablen Zustand". In: *Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse* 1913 (1913), pp. 582–592. URL: http://eudml.org/doc/58894 (cit. on pp. 33, 38).

[MMC20] A. Mendizabal, P. Márquez-Neila, S. Cotin. "Simulation of hyperelastic materials in real-time using deep learning". In: *Medical Image Analysis* 59 (2020). ISSN: 13618423. DOI: 10.1016/j.media.2019.101569. arXiv: 1904.06197 (cit. on pp. 15, 17, 23, 25).

[ON15] K. O'Shea, R. Nash. "An Introduction to Convolutional Neural Networks". In: (2015), pp. 1–11. DOI: 10.48550/ARXIV.1511.08458. arXiv: 1511.08458. URL: http://arxiv.org/abs/1511.08458 (cit. on pp. 18, 19).

[PGM+19] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, R. Garnett. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf (cit. on p. 34).

[RFB15] O. Ronneberger, P. Fischer, T. Brox. “U-net: Convolutional networks for biomedical image segmentation”. In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241 (cit. on p. 25).

[Rud16] S. Ruder. “An overview of gradient descent optimization algorithms”. In: (2016), pp. 1–14. arXiv: 1609.04747. URL: http://arxiv.org/abs/1609.04747 (cit. on p. 32).

[SP07] A. Seth, M. G. Pandy. “A neuromusculoskeletal tracking method for estimating individual muscle forces in human movement”. In: *Journal of Biomechanics* 40.2 (2007), pp. 356–366. ISSN: 00219290. DOI: 10.1016/j.jbiomech.2005.12.017 (cit. on p. 15).

[VGO+20] P. Virtanen et al. “SciPy 1.0: fundamental algorithms for scientific computing in Python”. In: *Nature Methods* 17.3 (2020), pp. 261–272. ISSN: 15487105. DOI: 10.1038/s41592-019-0686-2. arXiv: 1907.10121 (cit. on p. 30).

[VL21] T. Viering, M. Loog. “The shape of learning curves: a review”. In: *arXiv preprint arXiv:2103.10948* (2021) (cit. on p. 39).

[VSPR18] J. Valentin, M. Sprenger, D. Pflüger, O. Röhrle. “Gradient-based optimization with B-splines on sparse grids for solving forward-dynamics simulations of three-dimensional, continuum-mechanical musculoskeletal system models”. In: *International Journal for Numerical Methods in Biomedical Engineering* 34.5 (2018), pp. 1–21. ISSN: 20407947. DOI: 10.1002/cnm.2965 (cit. on pp. 27, 28).

[WPC+21] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, P. S. Yu. “A Comprehensive Survey on Graph Neural Networks”. In: *IEEE Transactions on Neural Networks and Learning Systems* 32.1 (2021), pp. 4–24. ISSN: 21622388. DOI: 10.1109/TNNLS.2020.2978386. arXiv: 1901.00596 (cit. on pp. 15, 18, 19).

[Wri13] P. Wriggers. *Nichtlineare finite-element-methoden*. Springer-Verlag, 2013 (cit. on p. 17).

All links were last followed on July, 2022.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature